RAL-91-056

Computing selected eigenvalues of sparse unsymmetric matrices using subspace iteration

by

I. S. Duff and J. A. Scott

ABSTRACT

This paper discusses the design and development of a code to calculate the eigenvalues of a large sparse real unsymmetric matrix that are the right-most, left-most, or are of largest modulus. A subspace iteration algorithm is used to compute a sequence of sets of vectors that converge to an orthonormal basis for the invariant subspace corresponding to the required eigenvalues. This algorithm is combined with Chebychev acceleration if the right-most or left-most eigenvalues are sought, or if the eigenvalues of largest modulus are known to be the right-most or left-most eigenvalues. An option exists for computing the corresponding eigenvectors. The code does not need the matrix explicitly since it only requires the user to multiply sets of vectors by the matrix. Sophisticated and novel iteration controls, stopping criteria, and restart facilities are provided. The code is shown to be efficient and competitive on a range of test problems.

Central Computing Department,
Atlas Centre,
Rutherford Appleton Laboratory,
Oxon OX11 0QX.

August 1993.

1 Introduction

We are concerned with the problem of computing selected eigenvalues and the corresponding eigenvectors of a large sparse real unsymmetric matrix. In particular, we are interested in computing either the eigenvalues of largest modulus or the right-most (or left-most) eigenvalues. This problem arises in a significant number of applications, including mathematical models in economics, Markov chain modelling of queueing networks, and bifurcation problems (for references, see Saad 1984). Although algorithms for computing eigenvalues of sparse unsymmetric matrices have received attention in the literature (for example, Stewart 1976a, Stewart and Jennings 1981, Saad 1980, 1984, 1989), there is a notable lack of general purpose robust software. The best-known codes are SRRIT (Stewart 1978) and LOPSI (Stewart and Jennings 1981). Both SRRIT and LOPSI use subspace iteration techniques and are designed to compute the eigenvalues of largest modulus. Many real problems, however, require a knowledge of the right-most eigenvalues. For example, common bifurcation problems involve computing the eigenvalue λ of largest real part (the right-most eigenvalue) of a stability matrix and then detecting when $Re(\lambda)$ becomes positive as the matrix changes (see example 2 in Section 3).

In the Harwell Subroutine Library, routine EA12 uses a subspace iteration method combined with Chebychev acceleration (Rutishauser 1969) to calculate the eigenvalues of largest modulus and the corresponding eigenvectors of a large sparse real symmetric matrix. There is no analogous routine in the Harwell Subroutine Library for the unsymmetric problem. In the NAG Library, routine F02BCF calculates selected eigenvalues and eigenvectors of real unsymmetric matrices by reduction to Hessenberg form, followed by the QR algorithm and inverse iteration for selected eigenvalues whose moduli lie between two user-supplied values. This routine is intended for dense matrices since all the entries of the matrix (including the zero entries) must be passed by the user to the routine, which stores the matrix as a two-dimensional array, making it unsuitable for large sparse matrices. Since it was the intention that EB12 should provide a code for unsymmetric problems that was analogous to the Harwell code EA12 for symmetric problems, EB12 uses subspace iteration techniques and in the design of the code we have not considered employing any of the other methods for computing selected eigenvalues and eigenvectors of large unsymmetric matrices such as Arnoldi's method and the unsymmetric Davidson's method which have been discussed recently in the literature (for example, see Saad 1980, Ho 1990, Ho, Chatelin, and Bennani 1990, and Sadkane 1991).

This paper describes the algorithms employed by EB12 and illustrates the use of EB12 on representative test problems. In Section 2 we introduce the algorithms and discuss some of the design features of the code. The results of using EB12 to find selected eigenvalues and the corresponding eigenvectors of a set of test examples taken from practical problems are presented in Section 3. These results illustrate the effect of varying the code's parameter values and demonstrate the superiority of Chebychev accelerated subspace iteration over simple subspace iteration for those problems where the right-most (or left-most) eigenvalues coincide with those of largest modulus. Concluding remarks are made in Section 4.

2 The algorithm

Let $\mathbf{A}$ be a real $n \times n$ matrix with eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$ ordered so that

$$|\lambda_1| \geq |\lambda_2| \geq ... \geq |\lambda_n|. \tag{2.1}$$

Let X be a subspace of dimension m with $m < n$ (in general, $m << n$). If $|\lambda_m| > |\lambda_{m+1}|$ then, under mild restrictions on X, as k increases, the subspaces $\mathbf{A}^k X$ tend toward the invariant subspace of $\mathbf{A}$ corresponding to $\lambda_1, \lambda_2, ..., \lambda_m$ (a proof is provided by Stewart 1975). The class of methods based on using the sequence of subspaces $\mathbf{A}^k X$, $k = 0,1,2,...$ includes subspace (or simultaneous) iteration methods. In the special case $m = 1$, the subspace iteration method reduces to the power method in which the dominant eigenvector of $\mathbf{A}$ is approximated by a sequence of vectors $\mathbf{A}^k \mathbf{x}$ $k = 0,1,2,...$ Subspace iteration methods are particularly suitable when the matrix $\mathbf{A}$ is large and sparse, since this class of methods only requires the multiplication of sets of vectors by $\mathbf{A}$.

Starting with an $n \times m$ matrix $\mathbf{X}_0$ whose columns form a basis for X, the subspace iteration method described by Stewart (1976a) for a real unsymmetric matrix $\mathbf{A}$ generates a sequence of matrices $\mathbf{X}_k$ according to the formula

$$\mathbf{A}\mathbf{X}_k = \mathbf{X}_{k+1}\mathbf{R}_{k+1}, \tag{2.2}$$

where $\mathbf{R}_{k+1}$ is an upper triangular matrix chosen so that $\mathbf{X}_{k+1}$ has orthonormal columns. This corresponds to applying the Gram-Schmidt orthogonalisation process to the columns of $\mathbf{A}\mathbf{X}_k$. It is straightforward to verify the relationship

$$\mathbf{A}^k \mathbf{X}_0 = \mathbf{X}_k \mathbf{R}_k \mathbf{R}_{k-1} ... \mathbf{R}_1, \tag{2.3}$$

so that the columns of $\mathbf{X}_k$ form a basis of $\mathbf{A}^k X$. Convergence of the i th column of $\mathbf{X}_k$ to the i th basis vector of the invariant subspace of $\mathbf{A}$ corresponding to $\lambda_1, \lambda_2, ..., \lambda_m$ is linear with convergence ratio $\max\{|\lambda_i/\lambda_{i-1}|, |\lambda_{i+1}/\lambda_i|\}$, which may be intolerably slow. A faster rate of convergence may be achieved by performing a "Schur-Rayleigh-Ritz" (SRR) step (Stewart 1976a) in which

$$\mathbf{B}_k = \mathbf{X}_k^{\mathrm{T}} \mathbf{A} \mathbf{X}_k \tag{2.4}$$

is formed and reduced by a unitary matrix $\mathbf{Z}_k$ to the real Schur form

$$\mathbf{T}_k = \mathbf{Z}_k^{\mathrm{T}} \mathbf{B}_k \mathbf{Z}_k, \tag{2.5}$$

where $\mathbf{T}_k$ is a block triangular matrix, in which each diagonal block $(\mathbf{T}_k)_{ii}$ is either of order 1 or is a 2×2 matrix having complex conjugate eigenvalues, with the eigenvalues ordered along the diagonal blocks in descending order of their moduli. The matrix $\mathbf{X}_k$ is then replaced by $\hat{\mathbf{X}}_k = \mathbf{X}_k \mathbf{Z}_k$. For an arbitrary matrix $\mathbf{X}_0$, if $|\lambda_{i-1}| > |\lambda_i| > |\lambda_{i+1}|$, the i th column of $\hat{\mathbf{X}}_k$ will in general converge to the i th basis vector of the invariant subspace of $\mathbf{A}$ corresponding to $\lambda_1, \lambda_2, ..., \lambda_m$ linearly with convergence ratio $|\lambda_{m+1}/\lambda_i|$. Thus convergence is faster and the first columns of $\hat{\mathbf{X}}_k$ tend to converge more quickly than the later columns. If the r eigenvalues of $\mathbf{A}$ of largest moduli are required, it is usual to iterate with m ($m > r$) trial vectors, the additional vectors being called guard vectors. The slowest rate of convergence will be for the r th basis vector, which has a convergence ratio $|\lambda_{m+1}/\lambda_r|$.

The purpose of the orthogonalisation of $\mathbf{A}\mathbf{X}_k$ (2.2) is to maintain linear independence among the columns of $\mathbf{A}\mathbf{X}_k$. To reduce overheads, it should not be done until there is reason to believe that some columns have become linearly dependent. Thus, in practice, $\mathbf{A}\mathbf{X}_k$ is replaced by $\mathbf{A}^l \mathbf{X}_k$ for some $l \geq 1$ ($l = l(k)$), and each iteration of the subspace iteration algorithm then consists of four main steps:

1. Compute $\mathbf{A}^{l(k)}\mathbf{X}_k$.
2. Orthonormalise $\mathbf{A}^{l(k)}\mathbf{X}_k$.
3. Perform an SRR step.
4. Test for convergence.

From the computed eigenvalues of $\mathbf{T}_k$, the corresponding eigenvectors of $\mathbf{T}_k$ can be determined by a simple back-substitution process (see Peters and Wilkinson 1970). The eigenvectors of $\mathbf{T}_k$ can be used to obtain approximate eigenvectors of $\mathbf{A}$ corresponding to the converged eigenvalues. To see this, let $\mathbf{w}_i$ denote the eigenvector of $\mathbf{T}_k$ corresponding to λ_i. Then

$$\mathbf{T}_k\mathbf{w}_i = \lambda_i\mathbf{w}_i. \tag{2.6}$$

From (2.4) and (2.5),

$$\mathbf{T}_k = \hat{\mathbf{X}}_k^{\mathrm{T}}\mathbf{A}\hat{\mathbf{X}}_k, \tag{2.7}$$

and hence

$$\hat{\mathbf{X}}_k^{\mathrm{T}}(\mathbf{A}\hat{\mathbf{X}}_k\mathbf{w}_i - \lambda_i\hat{\mathbf{X}}_k\mathbf{w}_i) = 0. \tag{2.8}$$

It follows that if $\mathbf{y}_i = \hat{\mathbf{X}}_k\mathbf{w}_i$, then as k increases, $(\mathbf{y}_i, \lambda_i)$ converges to the i th eigenpair of $\mathbf{A}$.

As already noted, in practice the simple subspace iteration algorithm uses $\mathbf{A}^l$ in place of $\mathbf{A}$. One possible way of improving the convergence rate achieved by the subspace iteration algorithm is to replace $\mathbf{A}^l$ by an iteration polynomial $p_l(\mathbf{A})$. The use of Chebychev polynomials to accelerate the convergence of subspace iteration was suggested by Rutishauser (1969) for symmetric problems. For the unsymmetric problem, Saad (1984) discussed how the technique can be extended to find the right-most (or left-most) eigenvalues of a real unsymmetric matrix and to accelerate the convergence of the simple subspace iteration algorithm when the eigenvalues of largest moduli are also the right-most (or left-most) eigenvalues. We use many of the ideas proposed by Saad. Suppose we want to find the r right-most eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_r$ of $\mathbf{A}$. Let $E(d, c, a)$ denote an ellipse with centre d, foci $d-c$ and $d+c$, major semi-axis a, and which is symmetric with respect to the real axis (since $\mathbf{A}$ is real the spectrum of $\mathbf{A}$ is symmetric with respect to the real axis) so that d is real and a and c are either real or purely imaginary. Suppose $E(d, c, a)$ contains the set S of unwanted eigenvalues $\lambda_{r+1}, \lambda_{r+2}, \ldots, \lambda_m$. The Chebychev accelerated iteration algorithm then chooses the iteration polynomial $p_l(\lambda)$ to be the polynomial given by

$$p_l(\lambda) = \frac{T_l[(\lambda - d)/c]}{T_l[(\lambda_r - d)/c]}, \tag{2.9}$$

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 denominator $T_l[(\lambda_r - d)/c]$ in (2.9) is a scaling factor and λ_r is termed the reference eigenvalue (Ho 1990 and Ho, Chatelin, and Bennani 1990). In practice, since λ_r is not known, it is replaced by some approximation γ, called the reference point; this is discussed in Section 2.7. Associated with each eigenvalue $\lambda_j \, \varepsilon S$ is a convergence factor

$$R_j(d, c) = \left| \frac{(\lambda_j - d) + ((\lambda_j - d)^2 - c^2)^{\frac{1}{2}}}{(\lambda_r - d) + ((\lambda_r - d)^2 - c^2)^{\frac{1}{2}}} \right|. \tag{2.10}$$

The choice of d, c, a which give an ellipse $E(d, c, a)$ enclosing all $\lambda_j \varepsilon S$ and which minimises $\max_{\lambda_j \varepsilon S} R_j(d, c)$ defines the **optimal ellipse**.

Forming $p_l(\mathbf{A})$ explicitly may be avoided by computing the columns of the matrix $p_l(\mathbf{A})\mathbf{X}_k$ using the three-term recurrence relation for Chebychev polynomials

$$T_{q+1}(\lambda) = 2\lambda T_q(\lambda) - T_{q-1}(\lambda), \quad q = 1,2,... \tag{2.11}$$

with $T_0(\lambda) = 1$, $T_1(\lambda) = \lambda$. To see how (2.11) is used, for an arbitrary vector $\mathbf{z}_0$, let $\mathbf{z}_q = p_q(\mathbf{A})\mathbf{z}_0$. Defining $\sigma_{q+1} = \rho_q / \rho_{q+1}$ with $\rho_q = T_q[(\lambda_r - d)/c]$, it follows from (2.11) that the vector $\mathbf{z}_q$ may be computed for $q = 1,2,...$ using the recurrence

$$\mathbf{z}_{q+1} = 2\frac{\sigma_{q+1}}{c}(\mathbf{A} - d\,\mathbf{I})\mathbf{z}_q - \sigma_{q+1}\sigma_q \mathbf{z}_{q-1} \tag{2.12}$$

where

$$\sigma_{q+1} = \frac{1}{2/\sigma_1 - \sigma_q}, \tag{2.13}$$

with $\sigma_1 = c/(\lambda_r - d)$ and $\mathbf{z}_1 = \frac{\sigma_1}{c}(\mathbf{A} - d\,\mathbf{I})\mathbf{z}_0$. We note that if c is purely imaginary, provided the reference point γ used to approximate λ_r is real, the above recursion can be carried out in real arithmetic since in this case the scalars σ_i are purely imaginary and hence σ_{q+1}/c and $\sigma_{q+1}\sigma_q$ are real.

It can be shown (Saad 1984) that if $E(d, c, a)$ is the optimal ellipse and $p_l(\mathbf{A})$ defined by (2.9) is used in the subspace iteration algorithm in place of $\mathbf{A}^l$, convergence is to the invariant subspace corresponding to the eigenvalues of $\mathbf{A}$ outside the ellipse and the convergence rate for the i th basis vector is η_i^l where

$$\eta_i = \frac{a + (a^2 - c^2)^{\frac{1}{2}}}{a_i + (a_i^2 - c^2)^{\frac{1}{2}}}, \quad 1 \le i \le m, \tag{2.14}$$

where $E(d, c, a_i)$ is the ellipse with centre d, foci $d - c$ and $d + c$, and major semi-axis a_i which passes through λ_i. If the m+1 right-most (or left-most) eigenvalues are also the m+1 eigenvalues of largest modulus, the convergence rate η_i^l can be much better than the value $|\lambda_{m+1}/\lambda_i|^l$ achieved by the simple subspace iteration algorithm. The effect of this faster rate of convergence is illustrated in Section 3 and is exploited in EB12 by employing Chebychev polynomials when the r right-most (or left-most) eigenvalues are sought and when the r eigenvalues of largest modulus are also the r right-most (or left-most) eigenvalues. The use of Chebychev polynomials is one way in which EB12 is a more general and flexible code than the codes LOPSI and SRRIT, which are only able to compute the eigenvalues of largest modulus.

Implicit in the above brief discussion of subspace iteration, there are many practical questions such as how to choose the subspace dimension m for a given value of r, how to start the iteration process, how to choose the degree l of the iteration polynomial $p_l(\mathbf{A})$, how to orthonormalise a set of vectors, how to construct the optimal ellipse $E(d, c, a)$, and how to test for convergence of an eigenvalue. We shall discuss these and other questions and how we deal with them in EB12 in some detail in later sections. However, omitting these problems for the present, the algorithm employed by EB12 has the following general structure.

1. *Start:* Choose the subspace dimension m and an $n \times m$ matrix $\mathbf{X}$ with orthonormal columns. Set $l = 1$, $p_l(\lambda) = 1$.

2. *Iteration:* Compute $\mathbf{X} \Leftarrow p_l(\mathbf{A})\mathbf{X}$.

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

4. *Convergence test:* If the first r columns of $\mathbf{X}$ is a satisfactory set of basis vectors spanning the invariant subspace corresponding to the r sought-after eigenvalues of $\mathbf{A}$ then **stop**, else determine the degree l of the iteration polynomial $p_l(\lambda)$ for the next iteration. If the eigenvalues of largest modulus are required and they are also the right-most (or left-most) eigenvalues, or if the right-most (or left-most) eigenvalues are required, find the ellipse $E(d,c,a)$ update the reference point γ, and set p_l according to equation (2.9) with λ_r replaced by γ. Otherwise, set $p_l(\lambda) = \lambda^l$.
   Go to 2.

In the following subsections we discuss how this algorithm is implemented in EB12. In Section 2.1 we describe the overall design of EB12 and, in particular, the use of reverse communication. In Section 2.2 the dimension m of the iteration subspace, which is a parameter which must be set by the user, is considered. The initial matrix $\mathbf{X}$ chosen by EB12 is discussed in Section 2.3. In Section 2.4 the convergence criterion is given. The determination of the degree l of the iteration polynomial is discussed in Section 2.5. In Section 2.6 the locking strategy employed by EB12 is outlined. In Section 2.7 the calculation of the ellipse is considered, and in Section 2.8 we discuss the computation of eigenvectors once the sought-after eigenvalues have been determined.

2.1 Overall control and design

The code EB12 is written in FORTRAN 77 and has two entries:

(a) EB12A (EB12AD in the double precision version) calculates the selected eigenvalues of $\mathbf{A}$.

(b) EB12B (EB12BD in the double precision version) uses the basis vectors calculated by EB12A to calculate the eigenvectors corresponding to the computed eigenvalues. Optionally, the scaled eigenvector residuals (see equation (2.36)) are computed.

EB12 does not require the user to supply the matrix $\mathbf{A}$ explicitly. Each time EB12 needs a set of vectors 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 available and only the product of $\mathbf{A}$ with vectors is known. Within the code EB12, only dense linear algebra operations are performed and for efficiency these exploit the BLAS (Basic Linear Algebra Subprograms) kernels. This includes the use of both level 2 BLAS routines (Dongarra, Du Croz, Hammarling, and Hanson 1988) and the level 3 BLAS routine _GEMM (Dongarra, Du Croz, Duff, and Hammarling 1990) to perform matrix-matrix multiplications of the form $\mathbf{B} = \mathbf{X}^{\mathrm{T}}\mathbf{Y}$, where $\mathbf{Y} = \mathbf{A}\mathbf{X}$ has been computed by the user. In addition to using BLAS routines, during the Schur-Rayleigh-Ritz step (see (2.4), (2.5)), EB12A employs the EISPACK routines ORTHES and ORTRAN to reduce $\mathbf{B} = \mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{X}$ to Hessenberg form $\mathbf{H}$ (Wilkinson and Reinsch

1971), which is then reduced to the real Schur form $\mathbf{T} = \mathbf{V}^{\mathrm{T}}\mathbf{H}\mathbf{V}$ using a modified version of the routine HQR3 given by Stewart (1976b). We have modified the routine HQR3 so that the diagonal blocks of **T** are ordered with the eigenvalues appearing in descending order of their moduli if the simple subspace iteration algorithm is being used, and in descending (or ascending) order of their real parts if Chebychev acceleration is being employed. This ordering is convenient so that we can "pick-off" the eigenvalues in turn as they converge (see Section 2.4).

2.2 The number of trial vectors

The user must supply EB12 with the number *r* of required eigenvalues and the dimension *m* of the iteration subspace to be used. The value of the parameter *m* is important. It influences the effectiveness of EB12 since the amount of storage required by the code and the number of matrix-vector multiplications at each iteration depends upon *m*, which implies that, for a specified *r*, *m* should not be chosen unnecessarily large. But if *m* is too small, the number of iterations required for convergence may be high. The number of trial vectors *m* must therefore be chosen with some care. The value of *m* must exceed *r*, the number of sought-after eigenvalues, to provide some guard vectors. For the simple subspace iteration algorithm, *m* must be at least *r*+1. If Chebychev polynomials are employed, in order to be able to construct the ellipse at each iteration and to allow for complex conjugate pairs of eigenvalues, EB12 normally requires *m* to be at least *r*+2 , but if the *r*+1 right-most (or left-most) eigenvalues are known to be real, *m* may equal *r*+1 (see Section 2.7 for more details). In practice, it is advisable to take *m* larger than this minimum value (see the discussion following equation (2.5)). In typical runs, we have taken *m* to be about 2*r*, but the best value for *m* for a given *r* is problem-dependent. At any stage of the computation, the user is able to increase (or decrease) the value of *m* and restart EB12. The results of employing different values of *m* for a given *r* for our test problems are presented in Section 3.

For some problems, if *r* eigenvalues are sought, it can be advantageous to run EB12 with *r* replaced by r_1, where r_1 exceeds *r*. The parameter *m* must then be chosen to satisfy $m \geq r_1+1$ for simple subspace iteration and $m \geq r_1+2$ for subspace iteration with Chebychev acceleration. The computation may be terminated once *r* eigenvalues have converged. This strategy may be useful if, for example, some of the unwanted eigenvalues of **A** have real parts which are almost equal to the real part of one or more of the wanted eigenvalues. This is discussed further in Section 2.7 and is illustrated in Example 2 of Section 3.

2.3 The starting matrix X

EB12A allows the user to supply an initial estimate of the *r* basis vectors which span the invariant subspace corresponding to the sought-after eigenvalues of **A**. If the user wishes to supply an estimate, on the first entry to EB12A the estimated values should be stored in the first *r* columns of **X**. The remaining *m*−*r* guard vectors are generated using the Harwell Subroutine Library random number generator, FA01AS, which generates random numbers in the range [−1,1]. The resulting set of *m* vectors is then orthonormalised using the modified Gram-Schmidt algorithm (see, for example, Golub and Van Loan 1989). The implementation of the modified Gram-Schmidt algorithm employed in EB12 uses level 2 BLAS kernels and was written by Van Loan (1989).

If the user does not wish to supply an estimate of the initial basis, a normalised random vector, $\mathbf{x}_1$, is

generated using FA01AS and control is passed to the user for the matrix-vector multiplication $\mathbf{Ax}_1$. In the next call to EB12A $\mathbf{Ax}_1$ is orthonormalised with respect to $\mathbf{x}_1$ using the modified Gram-Schmidt algorithm to give $\mathbf{x}_2$. The process is repeated until $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m$ have been computed. The resulting set of orthonormal vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m$ are taken to be the columns of the starting matrix **X**. This choice of starting matrix amounts to using one step of Arnoldi's method (see, for example, Saad 1980). In general we found that this starting matrix yielded better results than were obtained using a starting matrix with random orthonormalised columns (that is, fewer matrix-vector multiplications and fewer iterations were required for convergence). This was particularly true when Chebychev acceleration was employed since in this case the use of Arnoldi's method on the first step provided better initial ellipse parameters than were obtained from a random starting matrix. If the user has some prior knowledge of the spectrum of **A** and feels that one step of Arnoldi's method is unlikely to provide a good starting matrix, a random starting matrix may be employed by placing random vectors in the first *r* columns of **X**.

2.4 The convergence criterion

We test for convergence after an SRR step. The convergence criterion used in EB12A essentially amounts to demanding that the relation

$$\mathbf{AX} = \mathbf{XT} \tag{2.15}$$

is almost satisfied. In particular, the *i* th column of **X** is considered to have converged if the following inequality is satisfied

$$\|(\mathbf{AX} - \mathbf{XT})_i\|_2 \leq \text{EPS}(2) * \|(\mathbf{AX})_i\|_2 \ , \tag{2.16}$$

where EPS(2) is a convergence parameter. The user is asked to assign to the parameter EPS(1) a value in the range $(u, 1.0)$, where u is the machine precision. If the user supplies a value which is out of range, EB12A issues a warning and sets EPS(2) to the default value $\sqrt{u}$; otherwise, EPS(2) is initially set equal to EPS(1).

EB12A requires the columns of **X** to be accepted in the order $i = 1, 2, ..., r$ so that column j is only tested for convergence if the preceding columns $i = 1, 2, ..., j-1$ have all converged. If columns j and $j+1$ of **X** correspond to a complex conjugate pair of eigenvalues, then (2.16) must hold simultaneously for $i = j$ and $i = j+1$. If $j-1$ eigenvalues have converged, then until the j th eigenvalue is accepted, the code monitors the scaled residual R_j given by

$$R_j = \|(\mathbf{AX} - \mathbf{XT})_j\|_2 / \|(\mathbf{AX})_j\|_2 . \tag{2.17}$$

Let $\lambda_j(k)$ and $R_j(k)$ denote, respectively, the computed approximation to the j th eigenvalue and the corresponding scaled residual on the k th iteration. For all k sufficiently large, the scaled residuals should satisfy

$$R_j(k+1) \leq R_j(k). \tag{2.18}$$

If

$$R_j(k+1) > R_j(k) > \text{EPS}(2) \tag{2.19}$$

and

$$|\lambda_j(k+1) - \lambda_j(k)| < \mathrm{EPS}(2) * 10^2 * \max(|\lambda_j(k+1)|, |\lambda_j(k)|), \tag{2.20}$$

then EB12A accepts $\lambda_j(k+1)$, issues a warning that the convergence tolerance requested by the user was not achieved, and sets EPS(2) to the value for which (2.16) is satisfied for $i=j$.

EB12A also checks for slow convergence. Convergence of the j th eigenvalue is considered to be intolerably slow if, for some k,

$$R_j(k+1) \le \mathrm{EPS}(2) * 10^2, \tag{2.21a}$$

and

$$R_j(k+1) \le \max_{l=0,1,2,3} R_j(k-l), \tag{2.21b}$$

and

$$R_j(k+1) \ge 0.9 * \max_{l=0,1,2,3} R_j(k-l). \tag{2.21c}$$

In this case, EB12A again accepts $\lambda_j(k+1)$, issues a warning, and sets EPS(2) to the value for which (2.16) is satisfied for $i=j$. If EB12A does return with $\mathrm{EPS}(2) \ne \mathrm{EPS}(1)$ (or $\mathrm{EPS}(2) \ne \sqrt{u}$ if EPS(1) was supplied out of range), the requested accuracy can often be achieved by increasing m and recalling EB12A. .

2.5 The degree l of $p_l(\lambda)$

At each iteration, EB12A must determine the degree l of the iteration polynomial $p_l(\lambda)$ to be used on the next iteration. We want to ensure that l is chosen so that the columns of $p_l(\mathbf{A})\mathbf{X}$ remain linearly independent. Björck (1967) showed that the modified Gram-Schmidt algorithm applied to a matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m]$ produces a computed orthonormal matrix $\hat{\mathbf{X}}$ which satisfies

$$\hat{\mathbf{X}}^{\mathrm{T}}\hat{\mathbf{X}} = \mathbf{I} + \mathbf{E}, \qquad \|\mathbf{E}\|_2 \approx u\kappa_2(\mathbf{X}), \tag{2.22}$$

where $\kappa_2(\mathbf{X}) = \max_{\|\mathbf{y}\|_2=1} \|\mathbf{X}\mathbf{y}\|_2 / \min_{\|\mathbf{y}\|_2=1} \|\mathbf{X}\mathbf{y}\|_2$ is the condition number of $\mathbf{X}$ and u is the machine precision. Thus the modified Gram-Schmidt algorithm should only be used to compute orthonormal bases when the vectors to be orthonormalised are reasonably independent. Let $l(k)$ denote the degree of the iteration polynomial on the k th iteration. We use (2.22) to obtain an estimate of $\kappa_2(\mathbf{X})$ and then use this to try and ensure that $l(k+1)$ will not be chosen so large that some of the columns of $p_{l(k+1)}(\mathbf{A})\mathbf{X}$ become linearly dependent. In particular, we require $l(k+1) \le l_1$ where

$$l_1 = \begin{cases} l(k) \times (1 + |\log_{10}(\kappa_2(\mathbf{X}) \times 10^{-q})|), & \kappa_2(\mathbf{X}) \le 10^q \\ l(k) \times (1 + \log_{10}(\kappa_2(\mathbf{X}) \times 10^{-q}))^{-1}, & \kappa_2(\mathbf{X}) > 10^q, \end{cases} \tag{2.23}$$

with $q=3$. This bound, which we have not seen in the published literature, was chosen to ensure l_1 varies from $l(k)$ in a controlled manner. If $\kappa_2(\mathbf{X})$ is not large, so that the columns of $\mathbf{X}$ are reasonably linearly independent, l_1 will be greater than $l(k)$, but if $\kappa_2(\mathbf{X})$ is large, l_1 is smaller than $l(k)$. The value of $q=3$ was selected as a result of our numerical experiments. Provided the other restrictions on l discussed in the remainder of this section are imposed, our results were not found to be very sensitive to changing the value of q to 2 or 4, but larger values of q sometimes led to an unnecessary amount of work being done before the ellipse is updated, while smaller values sometimes caused an SRR step to be performed unnecessarily early.

When Chebychev acceleration is used, it is necessary to restrict the degree of the iteration polynomial when the current ellipse is not a good ellipse, since otherwise (2.23) may lead to a large number of matrix-vector multiplications being performed before there is an opportunity to update the ellipse. The restriction EB12A imposes on the degree of the iteration polynomial for both the simple subspace and Chebychev accelerated algorithms is taken from Stewart and Jennings (1981) and is $l(k+1) \le l_2$ with

$$l_2 = 0.5 \times (1 + \log_{10}(u^{-1})/\log_{10}(ratio)), \tag{2.24}$$

where *ratio* is the ratio of the convergence rates of the slowest and fastest converging eigenvalue. If the ellipse is poor, *ratio* is large and l_2 will be small but as the algorithm converges, l_2 increases and our numerical experiments found that the degree of the iteration polynomial is then governed by (2.23) rather than (2.24).

Near convergence, if (2.23) and (2.24) are used to determine $l(k)$ and its value is large, the iteration polynomial may yield approximate eigenvalues which are more accurate than required by the convergence criterion. In this case, unnecessary matrix-vector multiplications may be performed. To avoid this we monitor the scaled residual R_r for the r th eigenvalue, which is given by equation (2.17) with $j = r$. For values of R_r close to EPS(2) we want to restrict l. As a result of our numerical experiments, in EB12A we set $l(k+1) \le l_3$ where

$$l_3 = t \times (1 + |\log_{10}(R_r/\text{EPS}(2))|). \tag{2.25}$$

with $t = 40$. Our numerical results were not very sensitive to the choice $t = 40$; similar results were obtained for $t = 30$ and $t = 50$, but larger values of t did not prevent unnecessary multiplications and, if t was too small, more iterations were needed for convergence. For some of our test problems, if only a small number of eigenvalues were required, the savings resulting from limiting l near convergence were significant. For example, for Example 2 of Section 3 with $n = 400$, if the two left-most eigenvalues were required and $m = 5$ was chosen, the restriction (2.25) gave a saving of more than 50 per cent in the number of matrix-vector multiplications required.

When the simple subspace algorithm is used, if $|\lambda_1| > 1$ the entries of the matrix $\mathbf{A}^l\mathbf{X}$ will grow as l increases. Let $\mathbf{x}_1$ ($\|\mathbf{x}_1\|_2 = 1$) be the first column of $\mathbf{X}$. Then

$$\mathbf{A}^l\mathbf{x}_1 = \lambda_1^l\mathbf{x}_1 + \mathbf{z} \tag{2.26}$$

for some $\mathbf{z}$. To prevent overflow, for the simple subspace iteration algorithm we require $l(k+1) \le l_4$ where l_4 satisfies

$$|\lambda_1^{l_4}| < M * 10^{-2}, \tag{2.27}$$

where M is the overflow limit. This gives a bound on l_4 of

$$l_4 < \frac{\log_{10} M - 2}{\log_{10}(|\lambda_1|)}. \tag{2.28}$$

If i eigenvalues have already converged, the restriction (2.28) becomes

$$l_4 < \frac{\log_{10} M - 2}{\log_{10}(|\lambda_{i+1}|)}. \tag{2.29}$$

Since λ_{i+1} is not known, the current estimate of λ_{i+1} is used in (2.29). For the Chebychev accelerated

algorithm it is not necessary to impose the restriction (2.28) (or (2.29)) since the iteration polynomial (2.9) is scaled so that the matrix entries do not overflow.

We remark that Stewart and Jennings (1981) impose a maximum value LMAX on the degree l of the iteration polynomial $p_l(\lambda)$ used in their code LOPSI (see also Saad 1984). In particular, they suggest using LMAX = 20. We have considered imposing a restriction $l \le$ LMAX in EB12A but have found that it usually led to poorer results. We report on this further in Section 3.

2.6 The use of locking

Computation time may be reduced when several eigenvalues are desired by using a "locking" technique. The idea behind locking techniques, which are sometimes also termed implicit deflation techniques (see, for example, Stewart 1976a and Saad 1989), is to exploit the fact that the initial columns of $\mathbf{X}$ tend to converge before the later ones. Once the basis vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_i$ corresponding to $\lambda_1, \lambda_2, ..., \lambda_i$ $(1 \le i < r)$ have satisfied the convergence criterion (2.16), they are "locked" and no further computations are carried out with these vectors. On the next and subsequent iterations, an iteration subspace of dimension $m-i$ is used to find the next eigenvalue. To consider in more detail the locking technique employed in EB12A, suppose i basis vectors have converged and let $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2)$, where $\mathbf{X}_1$ is the $n \times i$ matrix containing the vectors which have converged. On subsequent iterations, EB12A forms $(\mathbf{X}_1, p_l(\mathbf{A})\mathbf{X}_2)$. This can lead to significant savings in the number of matrix-vector multiplications if several eigenvalues are required. These savings are illustrated in Section 3. At the orthogonalisation step, further savings are made since the columns of $\mathbf{X}_1$ are already orthonormal. In the Schur-Rayleigh-Ritz step, the matrix $\mathbf{B} = \mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{X}$ has the form

$$\mathbf{B} = \begin{pmatrix} \mathbf{T}_{11} & \mathbf{X}_1^{\mathrm{T}}\mathbf{A}\mathbf{X}_2 \\ \mathbf{X}_2^{\mathrm{T}}\mathbf{A}\mathbf{X}_1 & \mathbf{X}_2^{\mathrm{T}}\mathbf{A}\mathbf{X}_2 \end{pmatrix}, \tag{2.30}$$

where $\mathbf{T}_{11} = \mathbf{X}_1^{\mathrm{T}}\mathbf{A}\mathbf{X}_1$ is an $i \times i$ block upper triangular matrix. Since the columns of $\mathbf{X}_1$ have converged

$$\mathbf{A}\mathbf{X}_1 = \mathbf{X}_1\mathbf{T}_{11} + \mathbf{E}_1 \tag{2.31}$$

for some matrix $\mathbf{E}_1$ with norm dependent upon the convergence parameter EPS(2) (as in (2.16)). Assuming $\mathbf{X}_2^{\mathrm{T}}\mathbf{X}_1$ is small (since the columns of $\mathbf{X}_2$ have been orthogonalised with respect to those of $\mathbf{X}_1$) and assuming $\mathbf{X}_2^{\mathrm{T}}\mathbf{E}_1$ is small, we can work with the partly triangularized system

$$\hat{\mathbf{B}} = \begin{pmatrix} \mathbf{T}_{11} & \mathbf{X}_1^{\mathrm{T}}\mathbf{A}\mathbf{X}_2 \\ 0 & \mathbf{B}_2 \end{pmatrix}, \quad \mathbf{B}_2 = \mathbf{X}_2^{\mathrm{T}}\mathbf{A}\mathbf{X}_2. \tag{2.32}$$

Thus it is only necessary to reduce the $(m-i) \times (m-i)$ matrix $\mathbf{B}_2$ to real Schur form.

Locking may cause some of the computed eigenvalues to appear out of order. To avoid this and to overcome the errors which are introduced by treating $\mathbf{X}_2^{\mathrm{T}}\mathbf{A}\mathbf{X}_1$ as zero, once EB12A has tentatively accepted the first r columns of $\mathbf{X}$ as basis vectors for the invariant subspace corresponding to the desired eigenvalues, the locking device is switched off. EB12A then takes the computed $n \times m$ matrix $\mathbf{X}$ and restarts the iterative process. The convergence criterion (2.16) must be satisfied simultaneously for $j = 1, 2, ..., r$. In all of our numerical experiments only one iteration with the unlocked system was necessary.

This locking technique differs from that described by Stewart and Jennings (1981) since their

technique finds all the eigenvalues of the $m \times m$ matrix **B** in equation (2.30) at each iteration and, if i basis vectors have already converged, only the vector $\mathbf{x}_{i+1}$ (and $\mathbf{x}_{i+2}$ for a complex conjugate pair) is tested for convergence. Since the matrix **B** changes with each iteration, the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_i$ of **B** and the corresponding basis vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_i$ which passed the convergence test on iteration k could fail the test on subsequent iterations. There is therefore a danger that eigenvalues and basis vectors which do not satisfy the convergence criterion will be returned. Moreover, when sorting the eigenvalues, Stewart and Jennings have to prevent vectors that are not locked from changing positions with locked vectors. They do this by computing the eigenvalues in an unordered sequence (using the EISPACK routine HQR2) and then artificially increasing the modulus of each eigenvalue corresponding to a locked vector immediately prior to sorting. After sorting they restore the values of the eigenvalue estimates.

2.7 Constructing the ellipse

When Chebychev acceleration is used, we want to find the optimal ellipse $E(d, c, a)$ enclosing the unwanted eigenvalues λ_j, $j \geq r+1$. Manteuffel's technique (1975, 1977, 1978) for doing this in the case of the solution of linear systems has been adapted by Saad (1984) to the unsymmetric eigenvalue problem. Manteuffel's algorithm does not allow a complex reference eigenvalue λ_r in equation (2.9) so Saad replaces λ_r by a real reference point γ and, in particular, on the k th iteration Saad takes γ to be the point on the real line which has the same convergence ratio as λ_r with respect to the ellipse found on the (k–1) st iteration. The best ellipse for the k th iteration is then determined to minimise the maximum convergence factor $R_j(d, c)$ given by equation (2.10) with λ_r replaced by γ. When λ_r is real, $\gamma = \lambda_r$. Since in general $\gamma \neq \lambda_r$, the ellipse found using Saad's method is only an approximation to the optimal ellipse. Nevertheless, our experience is that this approximation generally works well and so EB12A uses Saad's choice of γ.

EB12A uses the eigenvalue estimates $\lambda_j^{(k)}$, $1 \leq j \leq m$, computed on iteration k to construct a sequence of ellipses $E^k(d, c, a)$, $k = 1,2,...$ using the following procedure. Here it is assumed that the r right-most eigenvalues are required.

for k := 1 **step** 1 **until** s **do**
begin
If $k = 1$, set $p_1(\lambda) = 1$; otherwise let γ be the point on the real line with the same convergence ratio as $\lambda_r^{(k-1)}$ with respect to $E^{k-1}(d, c, a)$ and define $p_1(\lambda)$ using (2.9) with λ_r replaced by γ.
Compute the eigenvalue estimates $\lambda_j^{(k)}$, $1 \leq j \leq m$, using steps 2 and 3 of the basic iteration algorithm of Section 2.
if the convergence criterion (2.16) is satisfied for $i = 1, 2, ..., r$ **go to** *exit*
else define the barrier $b = Re(\lambda_r^{(k)})$ and the set of unwanted eigenvalues $S^k = \{\lambda_j^{(k)} : Re(\lambda_j^{(k)}) < b\}$.
Construct the positive convex hull K^k containing S^k and the points (x, y) on the previous hull K^{k-1} for which $x < b$.
Find the best ellipse $E^k(d, c, a)$ using the algorithms of Manteuffel (1975, 1977) and Saad (1990).
end
end
exit:

We observe that if $\lambda_r^{(k)}$ is real (respectively, complex), S^k will usually be nonempty provided $m \geq r+1$ (respectively, $m \geq r+2$). Thus in general the number of trial vectors m must satisfy $m \geq r+2$.

Manteuffel (1975, 1977) describes and implements an algorithm for finding the ellipse $E^k(d, c, a)$ using the positive convex hull K^k (see also Ashby 1985). In EB12A, we follow the procedure described by Manteuffel but we have modified his code.

We remark that it is necessary to use the previous hull K^{k-1} when constructing K^k. Suppose the right-most eigenvalues are sought. Typically, during the first few iterations, the ellipse $E^k(d, c, a)$ will not contain the actual left-most eigenvalues of **A**. Since $p_I(\lambda)$ is small for eigenvalues inside the ellipse compared with those lying outside the ellipse, convergence will be towards the left-most (unwanted) eigenvalues as well as to the right-most eigenvalues. At some stage, the computed left-most eigenvalues will lie within a hull which contains the actual left-most eigenvalues of **A** and, provided all subsequent hulls contain these computed left-most eigenvalues, convergence will be to the sought-after right-most eigenvalues.

Some of the ellipses $E^k(d, c, a)$ may contain wanted eigenvalues, which will slow convergence down. This is likely to be a problem if there is a cluster of eigenvalues near λ_r. In this case, it can be advantageous to set *r* to be larger than the actual number of required eigenvalues (see Section 2.2). The effect of choosing a larger value of *r* is to move the barrier *b* to the left (or right). If more than one eigenvalue (or more than one pair of complex conjugate eigenvalues) is required, once they have all converged, we recall EB12A with *r* set to the actual number of required eigenvalues to overcome the errors introduced by locking. This is illustrated in Example 2 of Section 3.

2.8 Computing the eigenvectors

Once EB12A has successfully computed the required eigenvalues of **A**, the user may call EB12B to compute the corresponding eigenvectors. EB12B computes the eigenvectors $\mathbf{w}_i$ of the block triangular matrix **T** using back-substitution and then takes the approximate eigenvectors of **A** to be $\mathbf{y}_i = \mathbf{X}\mathbf{w}_i$ (see (2.8)). The computed eigenvectors $\mathbf{y}_i$ are normalised. If the *i* th eigenvalue is complex with positive imaginary part, on exit from EB12B the *i* th and (*i*+1) th columns of a matrix **Y** will hold the real and imaginary parts of the *i* th eigenvector, respectively. Since the (*i*+1) th eigenvector is the complex conjugate of the *i* th eigenvector, working with complex arrays is avoided. When computing the eigenvectors of **T** using back-substitution, we found it was necessary to set all the entries in the lower triangular part of **T** (except those in the 2×2 diagonal blocks corresponding to complex eigenvalues) to zero. If we did not do this, the small off-diagonal entries in the matrix **T** computed by EB12A could cause large errors in the computed eigenvectors of **A**.

Suppose $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2)$ where the *r* columns of $\mathbf{X}_1$ have converged and $\mathbf{T}_{11} = \mathbf{X}_1^{\mathrm{T}}\mathbf{A}\mathbf{X}_1$. The residuals for the computed eigenvectors of **A** will be small if the residuals for the corresponding eigenvectors of $\mathbf{T}_{11}$ are small. To demonstrate this, let $\mathbf{w}$ ($\|\mathbf{w}\|_2 = 1$) denote the computed eigenvector of $\mathbf{T}_{11}$ corresponding to the computed eigenvalue λ and let $\mathbf{r}_1$ be the residual vector

$$\mathbf{r}_1 = \mathbf{T}_{11}\mathbf{w} - \lambda\mathbf{w}. \tag{2.33}$$

The corresponding approximate eigenvector of **A** is given by $\mathbf{y} = \mathbf{X}_1\mathbf{w}$, and from (2.31) and (2.33) we have

$$\mathbf{E}_1\mathbf{w} = \mathbf{A}\mathbf{X}_1\mathbf{w} - \mathbf{X}_1\mathbf{T}_{11}\mathbf{w} = \mathbf{A}\mathbf{y} - \lambda\mathbf{y} - \mathbf{X}_1\mathbf{r}_1. \tag{2.34}$$

Hence, since $\mathbf{X}_1$ has orthonormal columns,

$$\|\mathbf{A}\mathbf{y} - \lambda\mathbf{y}\|_2 \le \|\mathbf{E}_1\|_2 + \|\mathbf{r}_1\|_2. \tag{2.35}$$

The inequality (2.35) shows that provided $\|\mathbf{E}_1\|_2$ and $\|\mathbf{r}_1\|_2$ are small, the residual for the computed eigenvector of **A** will be small. If the scaled eigenvector residuals

$$\frac{\|(\mathbf{A}\mathbf{y}_i - \lambda_i\mathbf{y}_i)\|_2}{\|(\mathbf{A}\mathbf{y}_i)\|_2}, \quad 1 \le i \le r, \tag{2.36}$$

are required, the user must compute **AY**, where **Y** has columns $\mathbf{y}_1$, $\mathbf{y}_2$,..., $\mathbf{y}_m$ and recall EB12B.

2.9 Use of EB12 to obtain other parts of the spectrum

The code EB12 can be used to find parts of the spectrum other than that corresponding to the eigenvalues of largest modulus or the right-most (or left-most) eigenvalues of **A**. If, for example, we wish to compute a group of interior eigenvalues, say those closest to *p* (that is, those for which $|\lambda - p|$ is smallest), we can use the simple subspace iteration algorithm option in EB12 and replace **A** by $(\mathbf{A} - p\,\mathbf{I})^{-1}$. In this case, on each return to the user, a matrix-matrix multiplication $\mathbf{U} = (\mathbf{A} - p\,\mathbf{I})^{-1}\mathbf{W}$, with **W** of order $n \times (m–i)$ (*i* is the number of locked vectors), must be performed. This is equivalent to solving the system of equations

$$(\mathbf{A} - p\,\mathbf{I})\,\mathbf{U} = \mathbf{W}. \tag{2.37}$$

If **A** is large the solution of the system (2.37) may itself be quite time-consuming but note that, if a direct method of solution is used, the decomposition of $\mathbf{A} - p\,\mathbf{I}$ into triangular factors needs only to be done once for a value of the shift *p*. Having performed the decomposition, on each return to the user it is only necessary to perform relatively cheap forward and backward substitutions. In practice, *p* may be an approximation to a required eigenvalue of **A**, in which case it may be advantageous to update the value of the shift as the computation proceeds and consequently several factorizations may be required. However, if the shift *p* is suitably chosen, the matrix $\mathbf{B} = (\mathbf{A} - p\,\mathbf{I})^{-1}$ will have a spectrum with much better separation properties than the original matrix **A** and the subspace iteration algorithm applied to **B** should require far fewer iterations for convergence than when it is applied to **A**. Thus, the rationale behind using a so-called shift-and-invert strategy is that the additional cost of the factorizations is amply repaid by the reduction in the number of iterations required by using **B** in place of **A**. Note that if the shift *p* is complex, the use of complex arithmetic in the subspace iteration algorithm may be avoided by replacing the complex operator $(\mathbf{A} - p\,\mathbf{I})^{-1}$ by the real operator $Re\,[(\mathbf{A} - p\,\mathbf{I})^{-1}]$ (see Saad 1989). The user must form the **LU** decomposition of the matrix $(\mathbf{A} - p\,\mathbf{I})^{-1}$ and every time $\mathbf{U} = Re\,[(\mathbf{A} - p\,\mathbf{I})^{-1}]\mathbf{W}$ is required, must perform forward and backward solves in the usual way and take the real part of the resulting matrix to yield the matrix **U** which is returned to EB12.

The code EB12 can also be used for the generalised eigenvalue problem $\mathbf{K}\mathbf{x} = \lambda\mathbf{M}\mathbf{x}$. In this case, on each return to the user it is necessary to solve a system of the form

$$\mathbf{M}\mathbf{U} = \mathbf{K}\mathbf{W}. \tag{2.38}$$

If a direct method of solution is used, the factorization of **M** needs only be done once for the entire calculation. To gain faster convergence, the shifted and inverted operator $(\mathbf{K} - p\,\mathbf{M})^{-1}\mathbf{M}$ may be used.

3 Numerical experiments

The code EB12 has been tested on a number of problems. In this section, we describe the results of using EB12 to calculate selected eigenpairs for three representative test examples. In each example the convergence parameter EPS(1) (see Section 2.4) was set to 10^{-5}. The numerical experiments were performed on a SUN SPARCstation using double precision (i.e. $u \approx 2.220446 * 10^{-16}$). The number of iterations required is defined to be the number of times the iteration polynomial $p_l(\mathbf{A})\mathbf{X}$ is computed. Throughout this section, r, m, and $l_{\max}$ denote, respectively, the number of eigenpairs sought, the number of trial vectors used, and the highest degree of the iteration polynomial used by EB12A. All CPU timings are in seconds.

Example 1. The first problem is taken from Stewart and Jennings (1981), who use this problem to illustrate the effectiveness of their subspace iteration code LOPSI. The matrix is a stochastic matrix obtained during the application of Markov modelling techniques to the analysis and evaluation of computer systems. The matrix is of order 163 and has 1207 nonzero entries. Table 3.1 compares the convergence characteristics for different numbers of trial vectors for the simple subspace iteration algorithm and for the Chebychev accelerated algorithm. For this example, the eigenvalues of largest modulus are also the right-most eigenvalues, so the two algorithms may be compared directly.

Table 3.1. A comparison of the simple subspace iteration algorithm and the Chebychev accelerated algorithm for Example 1 (right-most eigenvalues).

r	m	Matrix-vector products		Iterations		$l_{\max}$		CPU time	
		Simple	Chebychev	Simple	Chebychev	Simple	Chebychev	Simple	Chebychev
5	7	2348	1037	10	14	121	55	47.2	28.9
5	8	1810	991	7	6	114	85	39.3	25.7
5	10	2128	958	7	6	101	60	48.9	28.6
10	15	4089	2034	11	9	94	75	105.6	70.4
10	20	3129	2304	9	8	69	39	120.1	99.8

The eigenvector corresponding to the dominant eigenvalue $\lambda_1 = 1$ is known to have all its elements of equal value. In all the tests using this problem, we used the option offered by EB12A for supplying an initial estimate of the r basis vectors corresponding to the sought-after eigenvalues to specify the first basis vector with length 1 and all its elements of equal value; the initial estimate of each of the other basis vectors for this problem was taken to be a random vector. The first basis vector passed the convergence test on the initial iteration and was then locked. For this problem, the locking facility for the remaining basis vectors did not come into effect until late in the computation so that most of the savings due to the locking techniques employed by EB12A come from the first vector. If locking is switched off, with $r = 10$ and $m = 20$, the simple subspace and Chebychev accelerated algorithms required 3540 and 2820 matrix-vector products, respectively.

For this example, we observe that, in each case, the value of $l_{\max}$ for the simple subspace iteration algorithm exceeded that for the Chebychev accelerated algorithm and that, with $r = 5$, the simple

subspace iteration algorithm took fewer iterations to converge than the Chebychev accelerated algorithm. However, the number of matrix-vector multiplications and the computation times for the Chebychev accelerated algorithm were considerably less than for the simple subspace iteration algorithm. Because of the way Stewart and Jennings (1981) present their results for this problem, it is difficult to make a direct comparison between the results obtained by LOPSI and EB12. However, compared with LOPSI, EB12 appears to require significantly fewer iterations and, if Chebychev acceleration is used, the number of matrix-vector multiplications used by EB12 is also considerably smaller.

Example 2. The second test example is taken from Garratt, Moore, and Spence (1991) and is concerned with the detection of Hopf bifurcation points in the parameter dependent nonlinear system

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, \nu), \quad \mathbf{f} : \mathbf{R}^n \times \mathbf{R} \to \mathbf{R}^n, \quad \mathbf{x} \in \mathbf{R}^n, \quad \nu \in \mathbf{R}. \tag{3.1}$$

The set $\Gamma := \{(\mathbf{x}, \nu) \in \mathbf{R}^{n+1} : \mathbf{f}(\mathbf{x}, \nu) = 0\}$ represents the steady-state solutions of (3.1) and it is often important to determine the (linearised) stability of a branch of Γ. If λ_1 denotes the right-most eigenvalue of the Jacobian matrix $\mathbf{A} = \mathbf{f}_{\mathbf{x}}(\mathbf{x}, \nu)$, then a steady-state solution is stable (unstable) if $Re(\lambda_1)$ is negative (positive). It is also desirable to be able to detect a Hopf bifurcation point, that is a point of Γ where $Re(\lambda_1)$ changes sign as ν varies. Example 2 arises from a pair of equations of the form (3.1) which model a tubular reactor (see equations (1)–(5) of Heinemann and Poore 1981), and which are discretised using simple central differences. We have taken n to be 200 and 400 and the parameter ν (the Damkohler number) to be 0.23. With this value for ν the steady-state solution is stable (see Garratt *et al.*1989). The matrix $\mathbf{A}$ is banded and has 796 and 1596 nonzero entries when $n = 200$ and 400, respectively. For $n = 200$, the right-most eigenvalue is (approximately) $-0.83285 \pm 1.5926i$ and the left-most eigenvalue is $-7850.5 \pm 1.2049i$. For $n = 400$, the right-most eigenvalue is $-0.04345 \pm 1.5806i$ and the left-most eigenvalue is $-31689.56 \pm 1.05784i$. Although it is the right-most eigenvalue that is of practical importance, we shall also use EB12 to calculate the left-most eigenvalue since the left-most eigenvalue is also the eigenvalue of largest modulus, which allows us to compare the simple subspace algorithm with the Chebychev accelerated algorithm. The results are given in Table 3.2.

In Table 3.2 we see that, when $n = 200$ and the simple subspace iteration algorithm is used, for each value of m, $l_{\max} = 78$. In this example $|\lambda_1| \approx 7850$ and the degree of the iteration polynomial is limited by (2.28), which prevents overflow. Similarly, when $n = 400$, (2.28) limits the degree of the iteration polynomial used by the simple subspace iteration algorithm to 68. There is no corresponding limit for the Chebychev accelerated algorithm, which again gives much better results than the simple subspace iteration algorithm. In particular, when $n = 400$, $r = 2$, and $m = 5$, the simple subspace iteration algorithm requires more than 21 times as many matrix-vector multiplications, 15 times as many iterations, and 18 times as much CPU time as the Chebychev accelerated algorithm.

In Table 3.3 we present some results for Example 2 for the Chebychev accelerated algorithm used to find the right-most eigenvalues. We observe that for $n = 200$ and $m = 8$, fewer matrix-vector products and less CPU time are required for convergence when $r = 4$ than when $r = 2$. In fact, with $m = 8$, the right-most pair of complex conjugate eigenvalues is found after 2541 matrix-vector products and 76.6 seconds CPU time. This is an example which illustrates that it can be advantageous to set r to be larger than the number of eigenvalues actually required. The results we obtained for Example 2 using EB12 compare favourably with those obtained by Garratt (1991) using Chebychev acceleration techniques.

Table 3.2. A comparison of the simple subspace iteration algorithm and the Chebychev accelerated algorithm for Example 2 (left-most eigenvalues).

n	r	m	Matrix-vector products		Iterations		l_{max}		CPU time	
			Simple	Chebychev	Simple	Chebychev	Simple	Chebychev	Simple	Chebychev
200	2	4	9763	1179	38	7	78	111	236.0	32.8
200	2	5	9674	969	31	6	78	99	238.2	27.4
200	2	8	11023	1623	24	7	78	64	284.2	49.3
400	2	5	43759	2069	139	9	68	96	2158.0	115.4

Table 3.3. Convergence results for the Chebychev accelerated subspace iteration algorithm for Example 2 (right-most eigenvalues).

n	r	m	Matrix-vector products	Iterations	l_{max}	CPU time
200	2	4	3575	10	224	96.7
200	2	6	3401	10	148	92.4
200	2	8	4127	10	139	115.9
200	4	6	4075	13	167	111.3
200	4	8	2881	11	125	87.4
400	2	4	13439	27	239	699.5
400	2	5	7489	15	230	398.3
400	2	8	8887	18	237	483.1

Example 3. The third test problem is another example of Markov chain modelling and is used by Saad (1984). This example models a random walk on a $(j+1)\times(j+1)$ triangular grid. Following Saad, we take $j=30$, so that the order of the matrix is $n=496$. For this problem, matrix-vector products **Ax** can be performed by a simple subroutine without explicitly forming the matrix **A**. Since EB12 returns control to the user for all matrix-vector products, it is particularly convenient for solving problems of this kind. In this problem, the eigenvalues of largest modulus are 1 and -1 and the right-most eigenvalue is 1.

In Table 3.4 we give convergence results for obtaining the dominant pair of eigenvalues $\lambda=\pm 1$ using the simple subspace iteration algorithm for various values of the parameter m. In Table 3.5 we present results for the Chebychev accelerated subspace iteration algorithm, but in this case only the right-most eigenvalue $\lambda=1$ is obtained. The numbers in parentheses in the second column of Table 3.5 are the figures reported by Saad (1984). The difference in the number of matrix-vector products used by EB12 and by Saad is mainly attributable to the fact that Saad uses different criteria for choosing the degree l of the iteration polynomial and, following Stewart and Jennings (1981), Saad imposes a maximum degree on the iteration polynomial. For this example, EB12 performs consistently better than the code used by Saad.

Table 3.4. Convergence results for the simple subspace iteration algorithm for Example 3 ($r = 2$).

m	Matrix-vector products	Iterations	l_{max}	CPU time
3	3407	19	116	161.0
4	1819	9	144	88.0
6	1721	6	129	88.3
8	1819	8	94	103.1
10	1739	7	54	108.1

Table 3.5. Convergence results for the Chebychev accelerated subspace iteration algorithm for Example 3 (right-most eigenvalue).

m	Matrix-vector products	Iterations	l_{max}	CPU time
3	371 (–)	5	48	21.6
4	419 (–)	5	42	25.1
6	527 (645)	5	39	35.1
8	567 (903)	6	23	45.9
10	669 (909)	6	25	60.4

From Tables 3.1-3.5 we see that, in general, the best results are obtained by choosing a value of m which is larger than the minimum allowed value. We also observe that, in most of the examples, increasing the number of trial vectors reduces (or keeps constant) the number of iterations required for convergence but the total number of matrix-vector multiplications needed may increase. Even if the number of matrix-vector multiplications needed decreases as m increases, the CPU time may increase since the order of the matrix which must be reduced to real Schur form is m. However, when the order of the matrix n is large, most of the CPU time is in the matrix-vector multiplication stage, and the total CPU time taken is dependent upon the efficiency with which these multiplications can be carried out. In particular, the time taken depends upon whether the user is able to exploit the structure of the matrix and vectorisation or parallelism.

We remarked in Section 2.5 that Stewart and Jennings (1981) impose a maximum value LMAX on the degree l of the iteration polynomial $p_l(\lambda)$ used in their code LOPSI. For all our test examples we found that the maximum degree used by EB12 exceeded the value $\mathrm{LMAX} = 20$ suggested by Stewart and Jennings. With the restriction $l \leq \mathrm{LMAX} = 20$, the Chebychev accelerated algorithm applied to Example 2 with $n = 200$, $r = 2$, and $m = 4$ required 7043 matrix-vector products and 86 iterations for convergence. This compares unfavourably with the corresponding result of 3575 matrix-vector products and 10 iterations given in Table 3.3. Further numerical experiments for this and other test examples using a range of values for LMAX led us to conclude that, in general, the results are worse when a restriction LMAX is placed on l.

In Section 2.9 we discussed the use of EB12 to compute eigenvalues of the matrix **A** other than

those which are right-most, left-most, or are of largest modulus. We may, for example, have an approximation p to an eigenvalue of $\mathbf{A}$ and want to obtain a more accurate approximation. In this case we would replace $\mathbf{A}$ by $(\mathbf{A}-p\,\mathbf{I})^{-1}$ and employ the simple subspace iteration algorithm, solving equation (2.37) on each return to the user. We have performed some numerical experiments to do this for the matrix in Example 1. In these experiments we used the Harwell Subroutine Library routine MA28AD to factor the matrix $(\mathbf{A}-p\,\mathbf{I})$ once, and on each return the factors created by MA28AD were used by MA28CD to solve (2.37). Full details of the MA28 package may be found in Duff (1977). We observed that for this example EB12 converged very quickly. Typically if an approximation to a (real) eigenvalue inside the spectrum was known to two decimal places, setting $r=1$ and $m=6$, convergence with EPS(2) equal to 10^{-5} (see equation (2.16)) was achieved in only one or two iterations. The computed eigenvalues agreed with those given in Table I of Stewart and Jennings (1981).

4 Concluding remarks

The purpose of this paper was to discuss the design and development of the code EB12 for computing selected eigenvalues and the corresponding eigenvectors of a real unsymmetric matrix $\mathbf{A}$. Existing codes LOPSI (Stewart and Jennings 1981) and SRRIT (Stewart 1978) use subspace iteration techniques to compute the eigenvalues of largest moduli. EB12 uses a subspace iteration algorithm, combined with Chebychev acceleration if either the right-most (or left-most) eigenvalues are required or it is known that the eigenvalues of largest moduli are also the right-most (or left-most) eigenvalues. EB12A works in terms of the Schur vectors of $\mathbf{A}$ and a second optional entry, EB12B, is used to obtain the eigenvectors once the Schur vectors have converged.

An important design feature of the code EB12 is that control is returned to the user each time a matrix-vector product $\mathbf{Ax}$ needs to be formed. This use of reverse communication makes the code suitable for large sparse problems and gives flexibility over the way in which the matrix is stored and matrix-vector products are performed. Another feature of the code is that it employs a new locking technique which is designed to reduce the number of matrix-vector multiplications required for convergence when more than one eigenvalue is required. The use of locking has been found to be efficient in practice.

The user of EB12 must choose the dimension m of the iteration subspace. This is an important parameter which effects the efficiency of the algorithm. We have provided the user with some guidance regarding the choice of m and, in addition, if a poor choice is made, we have designed EB12 so that the computation can be restarted at any stage with a different value of m while taking advantage of the basis vectors which have already been computed.

The usefulness of the code EB12 has been illustrated on a number of representative practical problems. The numerical results show that subspace iteration combined with Chebychev acceleration is significantly superior to simple subspace iteration when the right-most (or left-most) eigenvalues are also those of largest modulus. The results also show that the efficiency of our subspace iteration algorithm (with or without Chebychev acceleration) is very dependent upon how the algorithm chooses l, the degree of the iteration polynomial. In particular, we found that imposing a maximum value on l, as suggested by other authors (for example, Stewart and Jennings 1981 and Saad 1984), could lead to a considerable degradation of the results. We have introduced new criteria for choosing l

which are designed to prevent the columns of the iteration matrix from becoming dependent, to limit the number of unnecessary matrix-vector multiplications, and, for the simple subspace iteration algorithm, to prevent overflow. In addition, we have used new stopping criteria designed to terminate the computation if the residuals increase close to convergence or if convergence becomes intolerably slow. In either case, EB12A issues a warning message and the user then has the option of restarting the computation with an increased number of trial vectors.

5 Availability of the code

EB12 is written in standard FORTRAN 77. The code will be included in Release 11 of the Harwell Subroutine Library. Anyone interested in using the code should contact the authors for licence details. The specification sheet (write-up) for EB12 is also available from the authors.

6 Acknowledgments

The authors would like to acknowledge work done on the early development of EB12 by C. F. Van Loan and S. Considine. The authors would also like to thank T. J. Garratt, Y. Saad, J.K. Reid, M. Sadkane, and S. Godet-Thobie for helpful discussions. In addition, we are grateful to T. J. Garratt for supplying the data for the test example 2.

7 References

Ashby, S. F. (1985). Chebycode: A Fortran implementation of Manteuffel's adaptive Chebyshev algorithm. M.Sc. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Illinois.

Björck, A. (1967). Solving linear least squares problems by Gram Schmidt orthogonalisation. *BIT*, **7**, 1-21.

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

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

Duff, I. S. (1977) MA28 a set of Fortran subroutines for sparse unsymmetric matrices. Harwell Report AERE R.8730, HMSO, London.

Garratt, T. J. (1991) Private communication.

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

Godet-Thobie, S. (1991) Private communication.

Golub, G. H. and Van Loan, C. F. (1989). Matrix Computations. Second Edition. John Hopkins Univ. Press, London.

Heinemann, R. F. and Poore, A. B. (1981). Multiplicity, stability, and oscillatory dynamics of the tubular reactor. *Chemical Engineering Science* **36**, 1411-1419.

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

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

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

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

Manteuffel, T. A. (1978). Adaptive procedure for estimating parameters for the nonsymmetric Tchebyshev iteration. *Numerische Math.* **31**, 183-208.

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

Rutishauser, H (1969). Computational aspects of F. L. Bauer's simultaneous iteration method. *Numerische Math.* **13**, 4-13.

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

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

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

Saad, Y. (1990). Private communication.

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

Stewart, G. W. (1975). Methods of simultaneous iteration for calculating eigenvectors of matrices. In: Topics in Numerical Analysis II (ed. J. H. H. Miller), Academic Press, 169-185.

Stewart, G. W. (1976a). Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices. *Numerische Math.* **25**, 123-136.

Stewart, G. W. (1976b). ALGORITHM 506. HQR3 and EXCHNG: Fortran subroutines for calculating and ordering the eigenvalues of a real upper Hessenberg matrix. *ACM Trans. Math. Softw.* **2**, 275-280.

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

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

Van Loan, C. F. (1989). Private communication.

Wilkinson, J. H. and Reinsch, C. (1971) Handbook for Automatic Computation. Vol. II. Springer-Verlag.