NAG Library Chapter Introduction

f12 – Large Scale Eigenproblems

Contents

1 Scope of the Chapter

This chapter provides functions for computing **some** eigenvalues and eigenvectors of large-scale (sparse) standard and generalized eigenvalue problems. It provides functions for:

– solution of symmetric eigenvalue problems;

– solution of nonsymmetric eigenvalue problems;

– solution of generalized symmetric-definite eigenvalue problems;

– solution of generalized nonsymmetric eigenvalue problems;

– partial singular value decomposition.

Functions are provided for both *real* and *complex* data.

The functions in this chapter have all been derived from the ARPACK software suite (see Lehoucq *et al.* (1998)), a collection of Fortran 77 subfunctions designed to solve large scale eigenvalue problems. The interfaces provided in this chapter have been chosen to combine ease of use with the flexibility of the original ARPACK software. The underlying iterative methods and algorithms remain essentially the same as those in ARPACK and are described fully in Lehoucq *et al.* (1998).

The algorithms used are based upon an algorithmic variant of the Arnoldi process called the Implicitly Restarted Arnoldi Method. For symmetric matrices, this reduces to a variant of the Lanczos process called the Implicitly Restarted Lanczos Method. These variants may be viewed as a synthesis of the Arnoldi/Lanczos process with the Implicitly Shifted QR technique that is suitable for large scale problems. For many standard problems, a matrix factorization is not required. Only the action of the matrix on a vector is needed.

2 Background to the Problems

This section is only a brief introduction to the solution of large-scale eigenvalue problems. For a more detailed discussion see, for example, Saad (1992) or Lehoucq (1995) in addition to Lehoucq *et al.* (1998). The basic factorization techniques and definitions of terms used for the different problem types are given in Section 2 in the f08 Chapter Introduction.

2.1 Sparse Matrices and their Storage

A matrix A may be described as **sparse** if the number of zero elements is so large that it is worthwhile using algorithms which avoid computations involving zero elements.

If A is sparse, and the chosen algorithm requires the matrix coefficients to be stored, a significant saving in storage can often be made by storing only the nonzero elements. A number of different formats may be used to represent sparse matrices economically. These differ according to the amount of storage required, the amount of indirect addressing required for fundamental operations such as matrix-vector products, and their suitability for vector and/or parallel architectures. For a survey of some of these storage formats see Barrett *et al.* (1994).

Most of the functions in this chapter have been designed to be independent of the matrix storage format. This allows you to choose your own preferred format, or to avoid storing the matrix altogether. Other functions are **general purpose**, which are easier to use, but are based on fixed storage formats. One such format is currently provided. This is the banded coordinate storage format as used in Chapters f07 and f08 (LAPACK) for storing general banded matrices.

2.2 Symmetric Eigenvalue Problems

The *symmetric eigenvalue problem* is to find the *eigenvalues*, λ, and corresponding *eigenvectors*, $z \neq 0$, such that

$$Az = \lambda z, \quad A = A^{\mathrm{T}}, \qquad \text{where } A \text{ is real.}$$

For the *Hermitian eigenvalue problem* we have

$$Az = \lambda z, \quad A = A^{\mathrm{H}}, \qquad \text{where } A \text{ is complex.}$$

For both problems the eigenvalues λ are real.

The basic task of the symmetric eigenproblem functions is to compute some of the values of λ and, optionally, corresponding vectors z for a given matrix A. For example, we may wish to obtain the first ten eigenvalues of largest magnitude, of a large sparse matrix A.

2.3 Generalized Symmetric-definite Eigenvalue Problems

This section is concerned with the solution of the generalized eigenvalue problems $Az = \lambda Bz$, $ABz = \lambda z$, and $BAz = \lambda z$, where A and B are real symmetric or complex Hermitian and B is positive definite. Each of these problems can be reduced to a standard symmetric eigenvalue problem, using a Cholesky factorization of B as either $B = LL^{\mathrm{T}}$ or $B = U^{\mathrm{T}}U$ (LL^{H} or $U^{\mathrm{H}}U$ in the Hermitian case).

With $B = LL^{\mathrm{T}}$, we have

$$Az = \lambda Bz \Rightarrow \left(L^{-1}AL^{-\mathrm{T}}\right)\left(L^{\mathrm{T}}z\right) = \lambda\left(L^{\mathrm{T}}z\right).$$

Hence the eigenvalues of $Az = \lambda Bz$ are those of $Cy = \lambda y$, where C is the symmetric matrix $C = L^{-1}AL^{-\mathrm{T}}$ and $y = L^{\mathrm{T}}z$. In the complex, case C is Hermitian with $C = L^{-1}AL^{-\mathrm{H}}$ and $y = L^{\mathrm{H}}z$.

The basic task of the generalized symmetric eigenproblem functions is to compute some of the values of λ and, optionally, corresponding vectors z for a given matrix A. For example, we may wish to obtain the first ten eigenvalues of largest magnitude, of a large sparse matrix pair A and B.

2.4 Nonsymmetric Eigenvalue Problems

The *nonsymmetric eigenvalue problem* is to find the *eigenvalues*, λ, and corresponding *eigenvectors*, $v \neq 0$, such that

$$Av = \lambda v.$$

More precisely, a vector v as just defined is called a *right eigenvector* of A, and a vector $u \neq 0$ satisfying

$$u^{\mathrm{T}}A = \lambda u^{\mathrm{T}} \quad \left(u^{\mathrm{H}}A = \lambda u^{\mathrm{H}} \quad \text{when } u \text{ is complex}\right)$$

is called a *left eigenvector* of A.

A real matrix A may have complex eigenvalues, occurring as complex conjugate pairs.

This problem can be solved via the *Schur factorization* of A, defined in the real case as

$$A = ZTZ^{\mathrm{T}},$$

where Z is an orthogonal matrix and T is an upper quasi-triangular matrix with 1 by 1 and 2 by 2 diagonal blocks, the 2 by 2 blocks corresponding to complex conjugate pairs of eigenvalues of A. In the complex case, the Schur factorization is

$$A = ZTZ^{\mathrm{H}},$$

where Z is unitary and T is a complex upper triangular matrix.

The columns of Z are called the *Schur vectors*. For each k ($1 \leq k \leq n$), the first k columns of Z form an orthonormal basis for the *invariant subspace* corresponding to the first k eigenvalues on the diagonal of T. Because this basis is orthonormal, it is preferable in many applications to compute Schur vectors rather than eigenvectors. It is possible to order the Schur factorization so that any desired set of k eigenvalues occupy the k leading positions on the diagonal of T.

The two basic tasks of the nonsymmetric eigenvalue functions are to compute, for a given matrix A, some values of λ and, if desired, their associated right eigenvectors v, and the Schur factorization.

2.5 Generalized Nonsymmetric Eigenvalue Problem

The *generalized nonsymmetric eigenvalue problem* is to find the eigenvalues, λ, and corresponding *eigenvectors*, $v \neq 0$, such that

$$Av = \lambda Bv, \quad ABv = \lambda v, \quad \text{and} \quad BAv = \lambda v.$$

More precisely, a vector v as just defined is called a *right eigenvector* of the matrix pair (A, B), and a vector $u \neq 0$ satisfying

$$u^{\mathrm{T}}A = \lambda u^{\mathrm{T}}B \quad \left(u^{\mathrm{H}}A = \lambda u^{\mathrm{H}}B \text{ when } u \text{ is complex}\right)$$

is called a *left eigenvector* of the matrix pair (A, B).

2.6 The Singular Value Decomposition

The *singular value decomposition* (SVD) of an m by n matrix A is given by

$$A = U\Sigma V^{\mathrm{T}}, \quad \left(A = U\Sigma V^{\mathrm{H}} \text{in the complex case}\right)$$

where U and V are orthogonal (unitary) and Σ is an m by n diagonal matrix with real diagonal elements, σ_i, such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0.$$

The σ_i are the *singular values* of A and the first $\min(m, n)$ columns of U and V are the *left* and *right singular vectors* of A. The singular values and singular vectors satisfy

$$Av_i = \sigma_i u_i \quad \text{and} \quad A^{\mathrm{T}}u_i = \sigma_i v_i \quad \left(\text{or } A^{\mathrm{H}}u_i = \sigma_i v_i\right) \quad \text{so that} \quad A^{\mathrm{T}}Au_i = \sigma_i^2 u_i \quad (A^{\mathrm{H}}Au_i = \sigma_i^2 u_i)$$

where u_i and v_i are the ith columns of U and V respectively.

Thus selected singular values and the corresponding right singular vectors may be computed by finding eigenvalues and eigenvectors for the symmetric matrix $A^{\mathrm{T}}A$ (or the Hermitian matrix $A^{\mathrm{H}}A$ if A is complex).

An alternative approach is to use the relationship

$$\begin{pmatrix} 0 & A \\ A^{\mathrm{T}} & 0 \end{pmatrix} \begin{pmatrix} U \\ V \end{pmatrix} = \begin{pmatrix} U \\ V \end{pmatrix} \Sigma$$

and thus compute selected singular values and vectors via the symmetric matrix

$$C = \begin{pmatrix} 0 & A \\ A^{\mathrm{T}} & 0 \end{pmatrix} \quad \left(C = \begin{pmatrix} 0 & A \\ A^{\mathrm{H}} & 0 \end{pmatrix} \text{ if } A \text{ is complex}\right).$$

In many applications, one is interested in computing a few (say k) of the largest singular values and corresponding vectors. If U_k, V_k denote the leading k columns of U and V respectively, and if Σ_k denotes the leading principal submatrix of Σ, then

$$A_k \equiv U_k \Sigma_k {V^{\mathrm{T}}}_k \quad (\text{or } U_k \Sigma_k {V^{\mathrm{H}}}_k)$$

is the best rank-k approximation to A in both the 2-norm and the Frobenius norm. Often a very small k will suffice to approximate important features of the original A or to approximately solve least squares problems involving A.

2.7 Iterative Methods

Iterative methods for the solution of the standard eigenproblem

$$Ax = \lambda x \tag{1}$$

approach the solution through a sequence of approximations until some user-specified termination criterion is met or until some predefined maximum number of iterations has been reached. The number of iterations required for convergence is not generally known in advance, as it depends on the accuracy required, and on the matrix A, its sparsity pattern, conditioning and eigenvalue spectrum.

3 Recommendations on Choice and Use of Available Functions

3.1 Types of Function Available

The functions available in this chapter divide essentially into three suites of basic reverse communication functions and some general purpose functions for banded systems.

Basic functions are grouped in suites of five, and implement the underlying iterative method. Each suite comprises a setup function, an options setting function, a solver function, a function to return additional monitoring information and a post-processing function. The solver function is independent of the matrix storage format (indeed the matrix need not be stored at all) and the type of preconditioner. It uses **reverse communication**, i.e., it returns repeatedly to the calling program with the argument **irevcm** set to specified values which require the calling program to carry out a specific task (either to compute a matrix-vector product or to solve the preconditioning equation), to signal the completion of the computation or to allow the calling program to monitor the solution. Reverse communication has the following advantages:

(i) Maximum flexibility in the representation and storage of sparse matrices. All matrix operations are performed outside the solver function, thereby avoiding the need for a complicated interface with enough flexibility to cope with all types of storage schemes and sparsity patterns. This also applies to preconditioners.

(ii) Enhanced user interaction: you can closely monitor the solution and tidy or immediate termination can be requested. This is useful, for example, when alternative termination criteria are to be employed or in case of failure of the external functions used to perform matrix operations.

At present there are suites of basic functions for real symmetric and nonsymmetric systems, and for complex systems.

General purpose functions call basic functions in order to provide easy-to-use functions for particular sparse matrix storage formats. They are much less flexible than the basic functions, but do not use reverse communication, and may be suitable in many cases.

The structure of this chapter has been designed to cater for as many types of application as possible. If a general purpose function exists which is suitable for a given application you are recommended to use it. If you then decide you need some additional flexibility it is easy to achieve this by using basic and utility functions which reproduce the algorithm used in the general purpose function, but allow more access to algorithmic control arguments and monitoring.

3.2 Iterative Methods for Real Nonsymmetric and Complex Eigenvalue Problems

The suite of basic functions nag_real_sparse_eigensystem_init (f12aac), nag_real_sparse_eigensystem_iter (f12abc), nag_real_sparse_eigensystem_sol (f12acc), nag_real_sparse_eigensystem_option (f12adc) and nag_real_sparse_eigensystem_monit (f12aec) implements the iterative solution of real nonsymmetric eigenvalue problems, finding estimates for a specified spectrum of eigenvalues. These eigenvalue estimates are often referred to as Ritz values and the error bounds obtained are referred to as the Ritz estimates. These functions allow a choice of termination criteria and many other options for specifying the problem type, allow monitoring of the solution process, and can return Ritz estimates of the calculated Ritz values of the problem A.

For complex matrices there is an equivalent suite of functions. nag_complex_sparse_eigensystem_init (f12anc), nag_complex_sparse_eigensystem_iter (f12apc), nag_complex_sparse_eigensystem_sol (f12aqc), nag_complex_sparse_eigensystem_option (f12arc) and nag_complex_sparse_eigensystem_monit (f12asc) are the basic functions which implement corresponding methods used for real nonsymmetric systems. Note that these functions are to be used for both Hermitian and non-Hermitian problems. Occasionally, when using these functions on a complex Hermitian problem, eigenvalues will be returned with small but nonzero imaginary part due to unavoidable round-off errors. These should be ignored unless they are significant with respect to the eigenvalues of largest magnitude that have been computed.

There are general purpose functions for the case where the matrices are known to be banded. In these cases an initialization function is called first to set up default options, and the problem is solved by a single call to a solver function. The matrices are supplied, in LAPACK banded-storage format, as

arguments to the solver function. For real general matrices these functions are nag_real_banded_sparse_eigensystem_init (f12afc) and nag_real_banded_sparse_eigensystem_sol (f12agc); and for complex matrices the pair is nag_complex_banded_eigensystem_init (f12atc) and nag_complex_banded_eigensystem_solve (f12auc). With each pair non-default options can be set, following a call to the initialization function, using nag_real_sparse_eigensystem_option (f12adc) for real matrices and nag_complex_sparse_eigensystem_option (f12arc) for complex matrices. For real matrices that can be supplied in the sparse matrix compressed column storage (CCS) format, the driver function nag_eigen_real_gen_sparse_arnoldi (f02ekc) is available. This function uses functions from Chapter f12 in conjunction with direct solver functions from Chapter f11.

There is little computational penalty in using the non-Hermitian complex functions for a Hermitian problem. The only additional cost is to compute eigenvalues of a Hessenberg rather than a tridiagonal matrix. The difference in computational cost should be negligible compared to the overall cost.

3.3 Iterative Methods for Real Symmetric Eigenvalue Problems

The suite of basic functions nag_real_symm_sparse_eigensystem_init (f12fac), nag_real_symm_sparse_eigensystem_iter (f12fbc), nag_real_symm_sparse_eigensystem_sol (f12fcc), nag_real_symm_sparse_eigensystem_option (f12fdc) and nag_real_symm_sparse_eigensystem_monit (f12fec) implement a Lanczos method for the iterative solution of the real symmetric eigenproblem.

There is a general purpose function pair for the case where the matrices are known to be banded. In this case an initialization function, nag_real_symm_banded_sparse_eigensystem_init (f12ffc), is called first to set up default options, and the problem is solved by a single call to a solver function, nag_real_symm_banded_sparse_eigensystem_sol (f12fgc). The matrices are supplied, in LAPACK banded-storage format, as arguments to nag_real_symm_banded_sparse_eigensystem_sol (f12fgc). Non-default options can be set, following a call to nag_real_symm_banded_sparse_eigensystem_init (f12ffc), using nag_real_symm_sparse_eigensystem_option (f12fdc).

3.4 Iterative Methods for Singular Value Decomposition

The partial singular value decomposition, A_k (as defined in Section 2.6), of an $(m \times n)$ matrix A can be computed efficiently using functions from this chapter. For real matrices, the suite of functions listed in Section 3.3 (for symmetric problems) can be used; for complex matrices, the corresponding suite of functions for complex problems can be used; however, there are no general purpose functions for complex problems.

The driver function nag_real_partial_svd (f02wgc) is available for computing the partial SVD of real matrices. The matrix is not supplied to nag_real_partial_svd (f02wgc); rather, a user-defined function argument provides the results of performing Matrix-vector products.

For both real and complex matrices, you should use the default options (see, for example, the options listed in Section 11 in nag_real_symm_sparse_eigensystem_option (f12fdc)) for problem type (**Standard**), computational mode (**Regular**) and spectrum (**Largest Magnitude**). The operation to be performed on request by the reverse communication function (e.g., nag_real_symm_sparse_eigensystem_iter (f12fbc)) is, for real matrices, to multiply the returned vector by the symmetric matrix $A^{\mathrm{T}}A$ if $m \geq n$, or by AA^{T} if $m < n$. For complex matrices, the corresponding Hermitian matrices are $A^{\mathrm{H}}A$ and AA^{H}.

The right ($m \geq n$) or left ($m < n$) singular vectors are returned by the post-processing function (e.g., nag_real_symm_sparse_eigensystem_sol (f12fcc)). The left (or right) singular vectors can be recovered from the returned singular vectors. Providing the largest singular vectors are not multiple or tightly clustered, there should be no problem in obtaining numerically orthogonal left singular vectors from the computed right singular vectors (or vice versa).

The second example in Section 10 in nag_real_symm_sparse_eigensystem_iter (f12fbc) illustrates how the partial singular value decomposition of a real matrix can be performed using the suite of functions for finding some eigenvalues of a real symmetric matrix. In this case $m \geq n$, however, the program is easily amended to perform the same task in the case $m < n$.

Similarly, functions in this chapter may be used to estimate the 2-norm condition number,

$$K_2(A) = \frac{\sigma_1}{\sigma_n}.$$

This can be achieved by setting the option **Both Ends** to get the largest and smallest few singular values, then taking the ratio of largest to smallest computed singular values as your estimate.

3.5 Alternative Methods

Other functions for the solution of sparse linear eigenproblems can currently be found in Chapters f02 and f08. In particular, the following functions allow the direct solution of real symmetric systems:

Band	nag_dsbevd (f08hcc), nag_dsbtrd (f08hec) and nag_dsteqr (f08jec)

4 General Use of Functions

This section will describe the complete structure of the reverse communication interfaces contained in this chapter. Numerous computational modes are available, including several shift-invert strategies designed to accelerate convergence. Two of the more sophisticated modes will be described in detail. The remaining ones are quite similar in principle, but require slightly different tasks to be performed with the reverse communication interface.

This chapter is structured as follows. The naming conventions used in this chapter, and the data types available are described in Section 4.1, spectral transformations are discussed in Section 4.2. Spectral transformations are usually extremely effective but there are a number of problem dependent issues that determine which one to use. In Section 4.3 we describe the reverse communication interface needed to exercise the various shift-invert options. Each shift-invert option is specified as a computational mode and all of these are summarised in the remaining sections. There is a subsection for each problem type and hence these sections are quite similar and repetitive. Once the basic idea is understood, it is probably best to turn directly to the subsection that describes the problem setting that is most interesting to you.

Perhaps the easiest way to rapidly become acquainted with the modes in this chapter is to run each of the example programs which use the various modes. These may be used as templates and adapted to solve specific problems.

4.1 Naming Conventions

Functions for solving nonsymmetric (real and complex) eigenvalue problems, in their short names, have as first letter after the chapter name, the letter ‘a’, e.g., nag_real_sparse_eigensystem_iter (f12abc); equivalent functions for symmetric eigenvalue problems will have this letter replaced by the letter ‘f’ (and ‘_symm’ added to their long names), e.g., nag_real_symm_sparse_eigensystem_iter (f12fbc). For the letter following this, functions for real eigenvalue problems will have letters in the range ‘a to m’ (and have long names beginning ‘nag_real’) while those for complex eigenvalue problems will have letters correspondingly shifted into the range ‘n to z’ (and long names beginning ‘nag_complex’); so, for example, the complex equivalent of nag_real_sparse_eigensystem_option (f12adc) is nag_complex_sparse_eigensystem_option (f12arc), while the real symmetric equivalent is nag_real_symm_sparse_eigensystem_option (f12fdc).

A suite of five functions are named consecutively in their short names and differ only in the final word of their long names, e.g., nag_real_sparse_eigensystem_init (f12aac), nag_real_sparse_eigensystem_iter (f12abc), nag_real_sparse_eigensystem_sol (f12acc), nag_real_sparse_eigensystem_option (f12adc) and nag_real_sparse_eigensystem_monit (f12aec). Each general purpose function has its own initialization function, but uses the option setting function from the suite relevant to the problem type. Thus each general purpose function can be viewed as belonging to a suite of three functions, even though only two functions will be named consecutively. For example, nag_real_sparse_eigensystem_option (f12adc), nag_real_banded_sparse_eigensystem_init (f12afc) and nag_real_banded_sparse_eigensystem_sol (f12agc) represent the suite of functions for solving a banded real symmetric eigenvalue problem.

4.2 Shift and Invert Spectral Transformations

The most general problem that may be solved here is to compute a few selected eigenvalues and corresponding eigenvectors for

$$Ax = \lambda Bx, \qquad \text{where } A \text{ and } B \text{ are real or complex } n \times n \text{ matrices.} \tag{2}$$

The shift and invert spectral transformation is used to enhance convergence to a desired portion of the spectrum. If (x, λ) is an eigen-pair for (A, B) and $\sigma \neq \lambda$ then

$$(A - \sigma B)^{-1} Bx = \nu x, \qquad \text{where } \nu = \frac{1}{\lambda - \sigma}. \tag{3}$$

This transformation is effective for finding eigenvalues near σ since the n_ν eigenvalues of $C \equiv (A - \sigma B)^{-1} B$ that are largest in magnitude correspond to the n_ν eigenvalues λ_j of the original problem that are nearest to the shift σ in absolute value. These transformed eigenvalues of largest magnitude are precisely the eigenvalues that are easy to compute with a Krylov method. (See Barrett *et al.* (1994)). Once they are found, they may be transformed back to eigenvalues of the original problem. The direct relation is

$$\lambda_j = \sigma + \frac{1}{\nu_j}$$

and the eigenvector x_j associated with ν_j in the transformed problem is also an eigenvector of the original problem corresponding to λ_j. Usually the Arnoldi process will rapidly obtain good approximations to the eigenvalues of C of largest magnitude. However, to implement this transformation, you must provide the means to solve linear systems involving $A - \sigma B$ either with a matrix factorization or with an iterative method.

In general, C will be non-Hermitian even if A and B are both Hermitian. However, this is easily remedied. The assumption that B is Hermitian positive definite implies that the bilinear form

$$\langle x, y \rangle \equiv x^{\mathrm{H}} By$$

is an inner product. If B is positive semidefinite and singular, then a semi-inner product results. This is a weighted B-inner product and vectors x, y are called B-orthogonal if $\langle x, y \rangle = 0$. It is easy to show that if A is Hermitian (self-adjoint) then C is Hermitian self-adjoint with respect to this B-inner product (meaning $\langle Cx, y \rangle = \langle x, Cy \rangle$ for all vectors x, y). Therefore, symmetry will be preserved if we force the computed basis vectors to be orthogonal in this B-inner product. Implementing this B-orthogonality requires you to provide a matrix-vector product Bv on request along with each application of C. In the following sections we shall discuss some of the more familiar transformations to the standard eigenproblem. However, when B is positive (semi)definite, we recommend using the shift-invert spectral transformation with B-inner products if at all possible. This is a far more robust transformation when B is ill-conditioned or singular. With a little extra manipulation (provided automatically in the post-processing functions) the semi-inner product induced by B prevents corruption of the computed basis vectors by roundoff-error associated with the presence of infinite eigenvalues. These very ill-conditioned eigenvalues are generally associated with a singular or highly ill-conditioned B. A detailed discussion of this theory may be found in Chapter 4 of Lehoucq *et al.* (1998).

Shift-invert spectral transformations are very effective and should even be used on standard problems, $B = I$, whenever possible. This is particularly true when interior eigenvalues are sought or when the desired eigenvalues are clustered. Roughly speaking, a set of eigenvalues is clustered if the maximum distance between any two eigenvalues in that set is much smaller than the minimum distance between these eigenvalues and any other eigenvalues of (A, B).

If you have a generalized problem $B \neq I$, then you must provide a way to solve linear systems with either A, B or a linear combination of the two matrices in order to use the reverse communication suites in this chapter. In this case, a sparse direct method should be used to factor the appropriate matrix whenever possible. The resulting factorization may be used repeatedly to solve the required linear systems once it has been obtained. If instead you decide to use an iterative method, the accuracy of the solutions must be commensurate with the convergence tolerance used for the Arnoldi iteration. A slightly more stringent tolerance is needed relative to the desired accuracy of the eigenvalue calculation.

The main drawback with using the shift-invert spectral transformation is that the coefficient matrix $A - \sigma B$ is typically indefinite in the Hermitian case and has zero-valued eigenvalues in the non-Hermitian case. These are often the most difficult situations for iterative methods and also for sparse direct methods.

The decision to use a spectral transformation on a standard eigenvalue problem $B = I$ or to use one of the simple modes is problem dependent. The simple modes have the advantage that you only need to supply a matrix vector product Av. However, this approach is usually only successful for problems where *extremal* non-clustered eigenvalues are sought. In non-Hermitian problems, extremal means eigenvalues near the boundary of the spectrum of A. For Hermitian problems, extremal means eigenvalues at the left- or right-hand end points of the spectrum of A. The notion of non-clustered (or well separated) is difficult to define without going into considerable detail. A simplistic notion of a well-separated eigenvalue λ_j for a Hermitian problem would be $\left\|\lambda_i - \lambda_j\right\| > \tau\|\lambda_n - \lambda_1\|$ for all $j \neq i$ with $\tau \gg \epsilon$, where λ_1 and λ_n are the smallest and largest algebraically. Unless a matrix vector product is quite difficult to code or extremely expensive computationally, it is probably worth trying to use the simple mode first if you are seeking extremal eigenvalues.

The remainder of this section discusses additional transformations that may be applied to convert a generalized eigenproblem to a standard eigenproblem. These are appropriate when B is well-conditioned (Hermitian or non-Hermitian).

4.2.1 *B* is Hermitian positive definite

If B is Hermitian positive definite and well-conditioned ($\|B\|\left\|B^{-1}\right\|$ is of modest size), then computing the Cholesky factorization $B = LL^{\mathrm{H}}$ and converting equation (2) to

$$\left(L^{-1}AL^{-\mathrm{H}}\right)y = \lambda \mathrm{y}, \qquad \text{where } L^{\mathrm{H}}x = y$$

provides a transformation to a standard eigenvalue problem. In this case, a request for a matrix vector product would be satisfied with the following three steps:

(i) Solve $L^{\mathrm{H}}z = v$ for z.

(ii) Matrix-vector multiply $z \leftarrow Az$.

(iii) Solve $Lw = z$ for w.

Upon convergence, a computed eigenvector y for $(L^{-1}AL^{-\mathrm{H}})$ is converted to an eigenvector x of the original problem by solving the triangular system $L^{\mathrm{H}}x = y$. This transformation is most appropriate when A is Hermitian, B is Hermitian positive definite and extremal eigenvalues are sought. This is because when A is Hermitian, so is $(L^{-1}AL^{-\mathrm{H}})$.

If A is Hermitian positive definite and the smallest eigenvalues are sought, then it would be best to reverse the roles of A and B in the above description and ask for the largest algebraic eigenvalues or those of largest magnitude. Upon convergence, a computed eigenvalue $\hat{\lambda}$ would then be converted to an eigenvalue of the original problem by the relation $\lambda \leftarrow \dfrac{1}{\hat{\lambda}}$.

4.2.2 *B* is not Hermitian positive semidefinite

If neither A nor B is Hermitian positive semidefinite, then a direct transformation to standard form is required. One simple way to obtain a direct transformation of equation (2) to a standard eigenvalue problem $Cx = \lambda x$ is to multiply on the left by B^{-1} which results in $C = B^{-1}A$. Of course, you should not perform this transformation explicitly since it will most likely convert a sparse problem into a dense one. If possible, you should obtain a direct factorization of B and when a matrix-vector product involving C is called for, it may be accomplished with the following two steps:

(i) Matrix-vector multiply $z \leftarrow Av$.

(ii) Solve $Bw = z$ for w.

Several problem-dependent issues may modify this strategy. If B is singular or if you are interested in eigenvalues near a point σ then you may choose to work with $C \equiv (A - \sigma B)^{-1}B$ but without using the

B-inner products discussed previously. In this case you will have to transform the converged eigenvalues of C to eigenvalues of the original problem.

4.3 Reverse Communication and Shift-invert Modes

The reverse communication interface function for real nonsymmetric problems is nag_real_sparse_eigensystem_iter (f12abc); for complex problems is nag_complex_sparse_eigensystem_iter (f12apc); and for real symmetric problems is nag_real_symm_sparse_eigensystem_iter (f12fbc). First the reverse communication loop structure will be described and then the details and nuances of the problem setup will be discussed. We use the symbol OP for the operator that is applied to vectors in the Arnoldi/Lanczos process and B will stand for the matrix to use in the weighted inner product described previously. For the shift-invert spectral transformation mode OP denotes $(A-\sigma B)^{-1}B$.

The basic idea is to set up a loop that repeatedly call one of nag_real_sparse_eigensystem_iter (f12abc), nag_complex_sparse_eigensystem_iter (f12apc) and nag_real_symm_sparse_eigensystem_iter (f12fbc). On each return, you must either apply OP or B to a specified vector or exit the loop depending upon the value returned in the reverse communication argument **irevcm**.

4.3.1 Shift and invert on a generalized eigenproblem

The example program in Section 10 in nag_real_sparse_eigensystem_monit (f12aec) illustrates the reverse communication loop for nag_real_sparse_eigensystem_iter (f12abc) in shift-invert mode for a generalized nonsymmetric eigenvalue problem. This loop structure will be identical for the symmetric problem calling nag_real_symm_sparse_eigensystem_iter (f12fbc). The loop structure is also identical for the complex arithmetic function nag_complex_sparse_eigensystem_iter (f12apc).

In the example, the matrix B is assumed to be symmetric and positive semidefinite. In the loop structure, you will have to supply a function to obtain a matrix factorization of $(A-\sigma B)$ that may repeatedly be used to solve linear systems. Moreover, a function needs to be provided to perform the matrix-vector product $z=Bv$ and a function is required to solve linear systems of the form $(A-\sigma B)w=z$ as needed using the previously computed factorization.

When convergence has taken place (indicated by **irevcm** $=5$ and **fail** $=0$), the reverse communication loop will be exited. Then, post-processing using the relevant function from nag_real_sparse_eigensystem_sol (f12acc), nag_complex_sparse_eigensystem_sol (f12aqc) and nag_real_symm_sparse_eigensystem_sol (f12fcc) must be done to recover the eigenvalues and corresponding eigenvectors of the original problem. When operating in shift-invert mode, the eigenvalue selection option is normally set to **Largest Magnitude**. The post-processing function is then used to convert the converged eigenvalues of OP to eigenvalues of the original problem (2). Also, when B is singular or ill-conditioned, the post-processing function takes steps to purify the eigenvectors and rid them of numerical corruption from eigenvectors corresponding to near-infinite eigenvalues. These procedures are performed automatically when operating in any one of the computational modes described above and later in this section.

You may wish to construct alternative computational modes using spectral transformations that are not addressed by any of the modes specified in this chapter. The reverse communication interface will easily accommodate these modifications. However, it will most likely be necessary to construct explicit transformations of the eigenvalues of OP to eigenvalues of the original problem in these situations.

4.3.2 Using the computational modes

The problem set up is similar for all of the available computational modes. In the previous section, a detailed description of the reverse communication loop for a specific mode (Shift-invert for a Generalized Problem) was given. To use this or any of the other modes listed below, you are strongly urged to modify one of the example programs.

The first thing to decide is whether the problem will require a spectral transformation. If the problem is generalized, $B\neq I$, then a spectral transformation will be required (see Section 4.2). Such a transformation will most likely be needed for a standard problem if the desired eigenvalues are in the interior of the spectrum or if they are clustered at the desired part of the spectrum. Once this decision has been made and OP has been specified, an efficient means to implement the action of the operator OP on

a vector must be devised. The expense of applying OP to a vector will of course have direct impact on performance.

Shift-invert spectral transformations may be implemented with or without the use of a weighted B-inner product. The relation between the eigenvalues of OP and the eigenvalues of the original problem must also be understood in order to make the appropriate eigenvalue selection option (e.g., **Largest Magnitude**) in order to recover eigenvalues of interest for the original problem. You must specify the number of eigenvalues to compute, which eigenvalues are of interest, the number of basis vectors to use, and whether or not the problem is standard or generalized. These items are controlled by setting options via the option setting function.

Setting the number of eigenvalues **nev** and the number of basis vectors **ncv** (in the setup function) for optimal performance is very much problem dependent. If possible, it is best to avoid setting **nev** in a way that will split clusters of eigenvalues. As a rule of thumb **ncv** $\geq 2 \times$ **nev** is reasonable. There are trade-offs due to the cost of the user-supplied matrix-vector products and the cost of the implicit restart mechanism. If the user-supplied matrix-vector product is relatively cheap, then a smaller value of **ncv** may lead to more user matrix-vector products and implicit Arnoldi iterations but an overall decrease in computation time. Convergence behaviour can be quite different depending on which of the spectrum options (e.g., **Largest Magnitude**) is chosen. The Arnoldi process tends to converge most rapidly to extreme points of the spectrum. Implicit restarting can be effective in focusing on and isolating a selected set of eigenvalues near these extremes. In principle, implicit restarting could isolate eigenvalues in the interior, but in practice this is difficult and usually unsuccessful. If you are interested in eigenvalues near a point that is in the interior of the spectrum, a shift-invert strategy is usually required for reasonable convergence.

The integer argument **irevcm** is the reverse communication flag that will specify a requested action on return from one of the solver functions nag_real_sparse_eigensystem_iter (f12abc), nag_complex_sparse_eigensystem_iter (f12apc) and nag_real_symm_sparse_eigensystem_iter (f12fbc). The options **Standard** and **Generalized** specify if this is a standard or generalized eigenvalue problem. The dimension of the problem is specified on the call to the initialization function only; this value, together with the number of eigenvalues and the dimension of the basis vectors is passed through the communication array. There are a number of spectrum options which specify the eigenvalues to be computed; these options differ depending on whether a Hermitian or non-Hermitian eigenvalue problem is to be solved. For example, the **Both Ends** is specific to Hermitian (symmetric) problems while the **Largest Imaginary** is specific to non-Hermitian eigenvalue problems (see Section 11.1 in nag_real_sparse_eigensystem_option (f12adc)). The specification of problem type will be described separately but the reverse communication interface and loop structure is the same for each type of the basic modes **Regular**, **Regular Inverse**, **Shifted Inverse** (also **Shifted Inverse Real** and **Shifted Inverse Imaginary** for real nonsymmetric problems), and for the problem type: **Standard** or **Generalized**. There are some additional specialised modes for symmetric problems, **Buckling** and **Cayley**, and for real nonsymmetric problems with complex shifts applied in real arithmetic. You are encouraged to examine the documented example programs for these modes.

The **Tolerance** specifies the accuracy requested. If you wish to supply shifts for implicit restarting then the **Supplied Shifts** must be selected, otherwise the default **Exact Shifts** strategy will be used. The **Supplied Shifts** should only be used when you have a great deal of knowledge about the spectrum and about the implicit restarted Arnoldi method and its underlying theory. The **Iteration Limit** should be set to the maximum number of implicit restarts allowed. The cost of an implicit restart step (major iteration) is in the order of $4n($**ncv** $-$ **nev**$)$ floating-point operations for the dense matrix operations and **ncv** $-$ **nev** matrix-vector products $w \leftarrow$ Av with the matrix A.

The choice of computational mode through the option setting function is very important. The legitimate computational mode options available differ with each problem type and are listed below for each of them.

4.3.3 Computational modes for real symmetric problems

The reverse communication interface function for symmetric eigenvalue problems is nag_real_symm_sparse_eigensystem_iter (f12fbc). The option for selecting the region of the spectrum of interest can be one of those listed in Table 1.

Largest Magnitude	The eigenvalues of greatest magnitude
Largest Algebraic	The eigenvalues of largest algebraic value (rightmost)
Smallest Magnitude	The eigenvalues of least magnitude.
Smallest Algebraic	The eigenvalues of smallest algebraic value (leftmost)
Both Ends	The eigenvalues from both ends of the algebraic spectrum

Table 1
Eigenvalue spectrum options for symmetric eigenproblems

Table 2 lists the spectral transformation options for symmetric eigenvalue problems together with the specification of OP and B for each mode and the problem type option setting.

Problem Type	**Mode**	**Problem**	**OP**	***B***
Standard	**Regular**	$Ax = \lambda \mathrm{x}$	A	I
Standard	**Shifted Inverse**	$Ax = \lambda \mathrm{x}$	$(A - \sigma I)^{-1}$	I
Generalized	**Regular Inverse**	$Ax = \lambda Bx$	$B^{-1}Ax$	B
Generalized	**Shifted Inverse**	$Ax = \lambda Bx$	$(A - \sigma B)^{-1}B$	B
Generalized	**Buckling**	$\mathrm{Kx} = \lambda K_G x$	$(K - \sigma K_G)^{-1}K$	K
Generalized	**Cayley**	$Ax = \lambda Bx$	$(A - \sigma B)^{-1}(A + \sigma B)$	B

Table 2
Problem types, computational modes and spectral transformations for symmetric eigenproblems

4.3.4 Computational modes for non-Hermitian problems

When A is a general non-Hermitian matrix and B is Hermitian and positive semidefinite, then the selection of the eigenvalues is controlled by the choice of one of the options in Table 3.

Largest Magnitude	The eigenvalues of greatest magnitude
Smallest Magnitude	The eigenvalues of least magnitude
Largest Real	The eigenvalues with largest real part
Smallest Real	The eigenvalues with smallest real part
Largest Imaginary	The eigenvalues with largest imaginary part
Smallest Imaginary	The eigenvalues with smallest imaginary part

Table 3
Eigenvalue spectrum options for real nonsymmetric and complex eigenproblems

Table 4 lists the spectral transformation options for real nonsymmetric eigenvalue problems together with the specification of OP and B for each mode and the problem type option setting. The equivalent listing for complex non-Hermitian eigenvalue problems is given in Table 5.

Problem Type	**Mode**	**Problem**	**OP**	***B***
Standard	**Regular**	$Ax = \lambda$x	A	I
Standard	**Shifted Inverse Real**	$Ax = \lambda$x	$(A - \sigma I)^{-1}$	I
Generalized	**Regular Inverse**	$Ax = \lambda Bx$	$B^{-1}Ax$	B
Generalized	**Shifted Inverse Real** with real σ	$Ax = \lambda Bx$	$(A - \sigma B)^{-1}B$	B
Generalized	**Shifted Inverse Real** with complex σ	$Ax = \lambda Bx$	$\text{real}\left\{(A - \sigma B)^{-1}B\right\}$	B
Generalized	**Shifted Inverse Imaginary** with complex σ	$Ax = \lambda Bx$	$\text{imag}\left\{(A - \sigma B)^{-1}B\right\}$	B

Table 4
Problem types, computational modes and spectral transformations for real nonsymmetric eigenproblems

Note that there are two shifted inverse modes with complex shifts in Table 4. Since σ is complex, these both require the factorization of the matrix $A - \sigma B$ in complex arithmetic even though, in the case of real nonsymmetric problems, both A and B are real. The only advantage of using this option for real nonsymmetric problems instead of using the equivalent suite for complex problems is that all of the internal operations in the Arnoldi process are executed in real arithmetic. This results in a factor of two saving in storage and a factor of four saving in computational cost. There is additional post-processing that is somewhat more complicated than the other modes in order to get the eigenvalues and eigenvectors of the original problem. These modes are only recommended if storage is extremely critical.

Problem Type	**Mode**	**Problem**	**OP**	***B***
Standard	**Regular**	$Ax = \lambda$x	A	I
Standard	**Shifted Inverse**	$Ax = \lambda$x	$(A - \sigma I)^{-1}$	I
Generalized	**Regular Inverse**	$Ax = \lambda Bx$	$B^{-1}Ax$	B
Generalized	**Shifted Inverse**	$Ax = \lambda Bx$	$(A - \sigma B)^{-1}B$	B

Table 5
Problem types, computational modes and spectral transformations for complex non-Hermitian eigenproblems

4.3.5 Post processing

On the final successful return from a reverse communication function, the corresponding post-processing function must be called to get eigenvalues of the original problem and the corresponding eigenvectors if desired. In the case of **Shifted Inverse** modes for **Generalized** problems, there are some subtleties to recovering eigenvectors when B is ill-conditioned. This process is called eigenvector purification. It prevents eigenvectors from being corrupted with noise due to the presence of eigenvectors corresponding to near infinite eigenvalues. These operations are completely transparent to you. There is negligible additional cost to obtain eigenvectors. An orthonormal (Arnoldi/Lanczos) basis is always computed. The approximate eigenvalues of the original problem are returned in ascending algebraic order. The option relevant to this function is **Vectors** which may be set to values that determine whether only eigenvalues are desired or whether corresponding eigenvectors and/or Schur vectors are required. The value of the shift σ used in spectral transformations must be passed to the post-processing function through the appropriately named argument(s). The eigenvectors returned are normalized to have unit length with

respect to the semi-inner product that was used. Thus, if $B = I$ then they will have unit length in the standard-norm. In general, a computed eigenvector x will satisfy $x^{\mathrm{H}}Bx = 1$.

4.3.6 Solution monitoring and printing

The option setting function for each suite allows the setting of three options that control solution printing and the monitoring of the iterative and post-processing stages. These three options are: **Advisory**, **Monitoring** and **Print Level**. By default, no solution monitoring or printing is performed. The **Advisory** option controls where solution details are printed; the **Monitoring** option controls where monitoring details are to be printed and is mainly used for debugging purposes; the **Print Level** option controls the amount of detail to be printed, see individual option setting function documents for specifications of each print level. The value passed to **Advisory** and **Monitoring** can be the same, but it is recommended that the two sets of information be kept separate. Note that the monitoring information can become very voluminous for the highest settings of **Print Level**.

To use the above options to print information to a file, the function nag_open_file (x04acc) must be called to open a file with a given name and return an associated Nag_FileID (see Section 3.2.1.1 in the Essential Introduction) for that file. The Nag_FileID (see Section 3.2.1.1 in the Essential Introduction) value can then be passed to the advisory or monitoring option setting string. On final exit from the post-processing function the file may be closed by a call to nag_close_file (x04adc).

The following example extract shows how the files ‘solut.dat’ and ‘monit.dat’ may be opened for the printing of solution and monitoring information respectively.

```
Nag_FileID solutid, monitid;
char option1[16], option2[16];
x04acc("solut.dat", 1, &solutid, &fail);
x04acc("monit.dat", 1, &monitid, &fail);
Vsprintf(option1, "advisory=%4ld", (Integer) solutid);
Vsprintf(option2, "monitoring=%4ld", (Integer) monitid);
.
.
.
f12adc(option1, icomm, comm, &fail);
f12adc(option2, icomm, comm, &fail);
f12adc("print level = 10", icomm, comm, &fail);
.
.
.
x04adc(solutid, &fail);
x04adc(monitid, &fail);
```

5 Functionality Index

Standard or generalized eigenvalue problems for complex matrices,
- banded matrices,
  - initialize problem and method ..... nag_complex_banded_eigensystem_init (f12atc)
  - selected eigenvalues, eigenvectors and/or Schur vectors ..... nag_complex_banded_eigensystem_solve (f12auc)
- general matrices,
  - initialize problem and method ..... nag_complex_sparse_eigensystem_init (f12anc)
  - option setting ..... nag_complex_sparse_eigensystem_option (f12arc)
  - reverse communication implicitly restarted Arnoldi method ..... nag_complex_sparse_eigensystem_iter (f12apc)
  - reverse communication monitoring ..... nag_complex_sparse_eigensystem_monit (f12asc)
  - selected eigenvalues, eigenvectors and/or Schur vectors of original problem ..... nag_complex_sparse_eigensystem_sol (f12aqc)

Standard or generalized eigenvalue problems for real nonsymmetric matrices,
- banded matrices,
  - initialize problem and method ..... nag_real_banded_sparse_eigensystem_init (f12afc)
  - selected eigenvalues, eigenvectors and/or Schur vectors ..... nag_real_banded_sparse_eigensystem_sol (f12agc)

general matrices,
initialize problem and method ..... nag_real_sparse_eigensystem_init (f12aac)
option setting ..... nag_real_sparse_eigensystem_option (f12adc)
reverse communication implicitly restarted Arnoldi method
..... nag_real_sparse_eigensystem_iter (f12abc)
reverse communication monitoring ..... nag_real_sparse_eigensystem_monit (f12aec)
selected eigenvalues, eigenvectors and/or Schur vectors of original problem
..... nag_real_sparse_eigensystem_sol (f12acc)

Standard or generalized eigenvalue problems for real symmetric matrices,
banded matrices,
initialize problem and method ..... nag_real_symm_banded_sparse_eigensystem_init (f12ffc)
selected eigenvalues, eigenvectors and/or Schur vectors
..... nag_real_symm_banded_sparse_eigensystem_sol (f12fgc)
general matrices,
initialize problem and method ..... nag_real_symm_sparse_eigensystem_init (f12fac)
option setting ..... nag_real_symm_sparse_eigensystem_option (f12fdc)
reverse communication implicitly restarted Arnoldi(Lanczos) method
..... nag_real_symm_sparse_eigensystem_iter (f12fbc)
reverse communication monitoring ..... nag_real_symm_sparse_eigensystem_monit (f12fec)
selected eigenvalues and eigenvectors and/or Schur vectors of original problem
..... nag_real_symm_sparse_eigensystem_sol (f12fcc)

6 Auxiliary Functions Associated with Library Function Arguments

None.

7 Functions Withdrawn or Scheduled for Withdrawal

None.

8 References

Barrett R, Berry M, Chan T F, Demmel J, Donato J, Dongarra J, Eijkhout V, Pozo R, Romine C and Van der Vorst H (1994) *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* SIAM, Philadelphia

Lehoucq R B (1995) Analysis and implementation of an implicitly restarted iteration *PhD Thesis* Rice University, Houston, Texas

Lehoucq R B (2001) Implicitly restarted Arnoldi methods and subspace iteration *SIAM Journal on Matrix Analysis and Applications* **23** 551–562

Lehoucq R B and Scott J A (1996) An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices *Preprint MCS-P547-1195* Argonne National Laboratory

Lehoucq R B and Sorensen D C (1996) Deflation techniques for an implicitly restarted Arnoldi iteration *SIAM Journal on Matrix Analysis and Applications* **17** 789–821

Lehoucq R B, Sorensen D C and Yang C (1998) *ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* SIAM, Philidelphia

Saad Y (1992) *Numerical Methods for Large Eigenvalue Problems* Manchester University Press, Manchester, UK