Chapter 22

Large Sparse Eigenvalue Problems

You should be familiar with

- Methods for computing eigenvalues of nonsparse matrices, both symmetric and nonsymmetric
- Arnoldi method
- Lanczos method

Chapters 18 and 19 present direct methods for computing eigenvalues of real matrices. These methods require $O\left(n^3\right)$ flops, and for large n it is unrealistic to use them. For instance, suppose a sparse matrix A is of dimension 500×500. It has 500 eigenvalues, accounting for multiplicities. Often in applications only certain eigenvalues and associated eigenvectors are required, so what we really need is a means of computing just those eigenvalues and eigenvectors. After studying this chapter, the interested reader will be prepared for more extensive discussions in such works as [3–5].

We begin our discussion with the power (Algorithm 18.1) method to compute the largest eigenvalue in magnitude. This method generates a Krylov sequence $x_0, Ax_0, A^2x_0, \ldots, A^{k-1}x_0$, and Krylov subspace methods form the basis for our iterative approach. To find a small collection of eigenvalues, we will make use of two methods presented in Chapter 21, the Arnoldi and Lanczos methods, both of which find an orthogonal basis for a Krylov subspace. The Arnoldi method provides a means for estimating a few eigenvalues and eigenvectors for a large sparse nonsymmetric matrix. There are two approaches, explicit and implicit Arnoldi. In practice, the implicit method is used, but it is necessary to first understand explicit methods. For symmetric matrices, the Lanczos decomposition is the tool of choice. The algorithms are similar to those for nonsymmetric matrices, but symmetry provides a much clearer picture of convergence properties.

22.1 THE POWER METHOD

The power method computes the largest eigenvalue in magnitude and an associated eigenvector. Recall that the statements

$$\begin{aligned} x_k &= Ax_{k-1}, \\ x_k &= x_k / \|x_k\|_2 \end{aligned}$$

are computed in a loop, so all that is required is matrix-vector multiplication and normalization. The initial guess should be randomly generated, and the convergence rate depends on the ratio $\left|\frac{\lambda_2}{\lambda_1}\right|$, where $|\lambda_1| > |\lambda_2|$ are two largest eigenvalues in magnitude.

Example 22.1. The matrix `rdb200` in the book software is a 200×200 nonsymmetric sparse matrix obtained from Matrix Market (Figure 22.1). The "rdb" refers to the reaction-diffusion Brusselator model used in chemical engineering. The MATLAB function `eigs` computes a few eigenvalues and associated eigenvectors of a large, sparse, matrix; in particular, E = eigs(A) returns a vector containing the six largest eigenvalues of A in magnitude. Apply `eigs` to `rdb200` and find the two largest eigenvalues. After computing the ratio $\left|\frac{\lambda_2}{\lambda_1}\right|$, use the function `largeeig` from the book software distribution to estimate the largest eigenvalue and compare the result with that of `eigs`. The ratio $\left|\frac{\lambda_2}{\lambda_1}\right|$ is 0.9720 and is not very favorable, but an error tolerance of 1.0×10^{-6} was obtained in 520 iterations, requiring 0.029465 s on the author's computer. ■

```
>> v = eigs(rdb200);
>> v(1:2)
ans =
  -33.3867   -32.4503
```

Numerical Linear Algebra with Applications. http://dx.doi.org/10.1016/B978-0-12-394435-1.00022-3

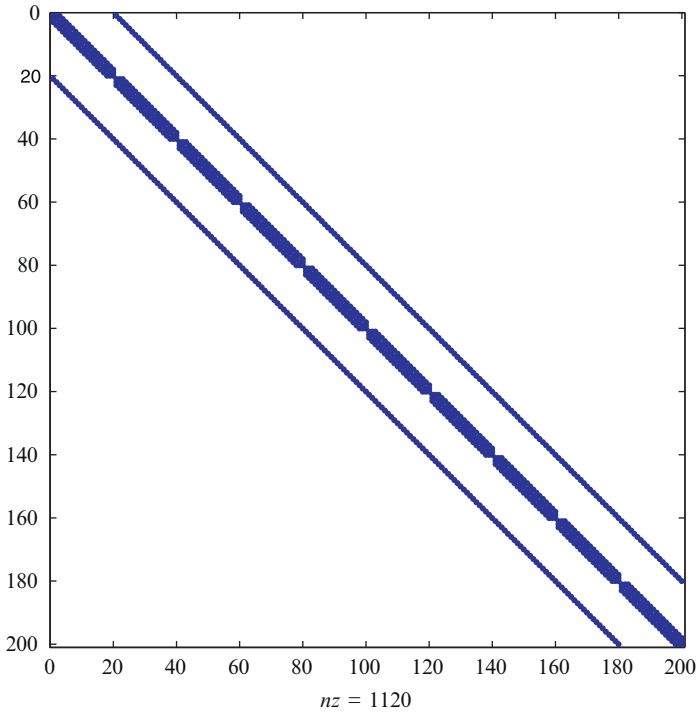


FIGURE 22.1 Nonsymmetric sparse matrix used in a chemical engineering model

```
>> abs(v(2)/v(1))
ans =
    0.9720
>> [lambda x iter] = largeeig (rdb200,rand(200,1),1.0 e-6,600);
>> lambda
lambda =
  -33.3867
>> abs(lambda - v(1))
ans =
   1.1795e-12
```

Starting with x_0, the power method generates the Krylov sequence $x_0 \;\; Ax_0 \;\; A^2x_0 \;\; \ldots \;\; A^{k-2}x_0 \;\; A^{k-1}x_0$. Of these vectors, we only use $A^{k-1}x_0$ as our approximate eigenvector. It seems reasonable that we can do better by choosing our approximate eigenvector as a linear combination $y = \sum_{i=0}^{k-1} c_i \left(A^i x_0\right)$ of vectors from the Krylov sequence. This is indeed the case as we will see in subsequent sections when discussing the use of the Arnoldi and Lanczos methods.

22.2 EIGENVALUE COMPUTATION USING THE ARNOLDI PROCESS

Recall from Chapter 21 that the Arnoldi process applied to an $n \times n$ matrix A produces a decomposition of the form (Equation 21.25)

$$AQ_m = Q_m H_m + h_{m+1,m}\, q_{m+1} e_m^{\mathrm{T}}, \tag{22.1}$$

where H is $m \times m$ upper Hessenberg matrix. The $n \times m$ matrix $Q_m = \left[\begin{array}{cccc} q_1 & q_2 & \ldots & q_{m-1} \;\; q_m \end{array} \right]$ is formed from column vectors $\{q_i\}$, and $\{q_1, q_2, \ldots, q_m\}$ form an orthogonal basis for the subspace spanned by $K_{m+1}(A, x_1) = \{x_1, Ax_1, \ldots, A^m x_1\}$. The additional vector, q_{m+1}, is orthogonal to $\{q_i\}$, $1 \leq i \leq m$. We used the Arnoldi process to develop the general minimal residual (GMRES) method (Algorithm 21.5) for approximating the solution to a large sparse system $Ax = b$. To approximate the solution we chose m to be smaller than n and project the problem into a Krylov subspace of dimension m. If $m = n$, $h_{m+1,m} = 0$, and the Arnoldi process gives a Schur decomposition (Theorem 18.5)

$$Q^{\mathrm{T}} A Q = H,$$

where Q is an $n \times n$ orthogonal matrix of Schur vectors, and H is $n \times n$ quasi-upper-triangular matrix with the eigenvalues of A on its diagonal. For large n, finding the Schur decomposition is impractical. Might it make sense to try the same type of approach as that of GMRES and approximate the eigenvalues of H_m in a Krylov subspace of dimension $m < n$? The question then becomes can we use the eigenvalues of H_m in Equation 22.1 to approximate the eigenvalues of A? If we can, then we will be able to compute a maximum of m eigenvalues and their associated eigenvectors. In reality, only a few of those m eigenvalues will lead to good approximations to the eigenvalues of A, and those eigenvalues tend to be the largest and the smallest eigenvalues of H_m.

Adopt the notation that $\lambda_i^{(m)}$ is eigenvalue i of H_m, and $u_i^{(m)}$ is the corresponding eigenvector. We call $\lambda_i^{(m)}$ a *Ritz value*, the vector $v_i^{(m)} = Q_m u_i^{(m)}$ a *Ritz eigenvector*, and the pair (λ_i, v_i) a *Ritz pair*. Now, from Equation 22.1

$$\begin{aligned} AQ_m u_i^{(m)} &= Q_m H_m u_i^{(m)} + h_{m+1,m}\, q_{m+1} e_m^{\mathrm{T}} u_i^{(m)} \\ AQ_m u_i^{(m)} &= Q_m \lambda_i^{(m)} u_i^{(m)} + h_{m+1,m}\, q_{m+1} e_m^{\mathrm{T}} u_i^{(m)} \\ Av_i^{(m)} &= \lambda_i^{(m)} v_i^{(m)} + h_{m+1,m}\, q_{m+1} e_m^{\mathrm{T}} u_i^{(m)}. \end{aligned} \tag{22.2}$$

Define the residual

$$r_i^{(m)} = \left(A - \lambda_i^{(m)} I\right) v_i^{(m)}.$$

Noting that $\|q_{m+1}\|_2 = 1$, $h_{m+1,n} \geq 0$, and $e_m^{\mathrm{T}} u_i^{(m)}$ is a real number, Equation 22.2 gives

$$\left\| r_i^{(m)} \right\|_2 = h_{m+1,m} \left| \left(u_i^{(m)} \right)_m \right|, \tag{22.3}$$

where $\left| \left(u_i^{(m)} \right)_m \right|$ is component m of $u_i^{(m)}$. If $h_{m+1,m}$ or $\left| \left(u_i^{(m)} \right)_m \right|$ is small, Equation 22.3 gives us a means of estimating the error of using the Ritz pair $\left(\lambda_i^{(m)}, v_i^{(m)} \right)$ as an eigenpair of A.

We can now begin to develop an algorithm for computing a few eigenvalues of A. If m is small, We had not such guarantee for a nonsymmetric matrix. then we can quickly compute the eigenvalues and associated eigenvectors of H_m and compute the value $\left\| r_i^{(m)} \right\|_2 = h_{m+1,m} \left| \left(u_i^{(m)} \right)_m \right|$ for all eigenvectors $u_i^{(m)}$. If any of the residuals is small, then we hope $\lambda_i^{(m)} / v_i^{(m)}$ is a good approximation to an eigenpair of A. Computational experience has shown that some of the eigenvalues $\lambda_i^{(m)}$ will be good approximations to eigenvalues of A well before m gets close to n [23, p. 444]. In practice, we want to use the smallest m possible.

Example 22.2. Recall that the function i, not 1 builds the biharmonic matrix discussed in Section 21.12. The reader should consult the MATLAB documentation to determine the action of `sprandn`. Generate a 144×144 random sparse matrix using the MATLAB statements

```
B = biharmonic_op(12,1,12,1);
A = sprandn(B);
```

The eigenvalues of A are complex. Apply the Arnoldi process to generate a matrix H_{35}, and compute its eigenvalues, the Ritz values. Then compute the eigenvalues of A using `eig` and plot them marked with a circle, followed by a plot on the same axes of the Ritz values marked by "x". Figure 22.2 displays the results. Note that the largest and smallest Ritz values in magnitude are close to eigenvalues of A. This is typical behavior. ■

22.2.1 Estimating Eigenvalues Without Restart or Deflation

The following algorithm outline chooses a random vector v_0, and performs an Arnoldi decomposition. It then computes eigenvalues (Ritz values) of the factor H_m, creating Ritz vectors $v_i^{(m)} = Q_m u_i^{(m)}$, sorts the Ritz pairs $\left(\lambda_i^{(m)}, v_i^{(m)} \right)$ in descending order of eigenvalue magnitude, and cycles through the Ritz pairs retaining up to *nev* of those for which the residual given by Equation 22.3 is sufficiently small. It is possible that none will be found.

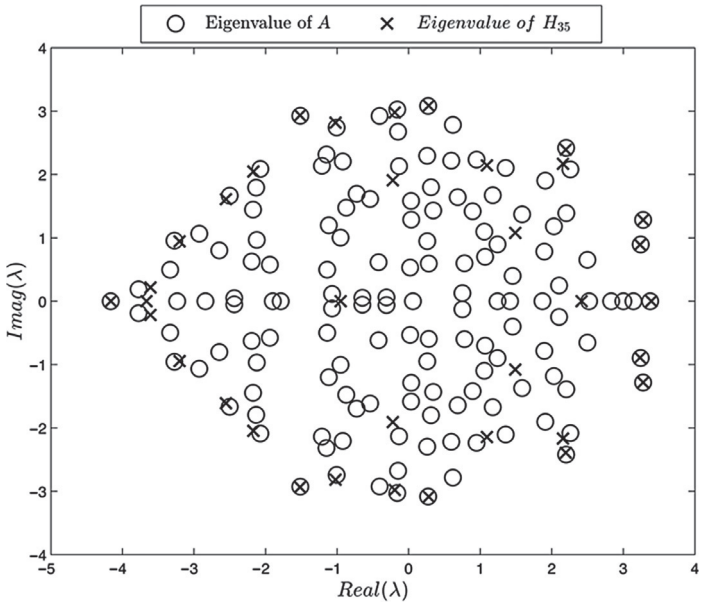


FIGURE 22.2 Eigenvalues and Ritz values of a random sparse matrix.

Simple Use of Arnoldi to Approximate a Few Eigenpairs of a Large, Sparse Matrix

1. Compute the Arnoldi decomposition $AQ_m = Q_m H_m + h_{m+1,m}\, q_{m+1} e_m^T$ using initial vector v_0.
2. Compute the eigenvectors $\left\{u_1^{(m)}, u_2^{(m)}, \ldots, u_m^{(m)}\right\}$ and eigenvalues $\left\{\lambda_1^{(m)}, \lambda_2^{(m)}, \ldots, \lambda_m^{(m)}\right\}$ of H_m.
3. Sort the Ritz pairs in descending order of eigenvalue magnitude.
4. Loop through the sorted Ritz pairs $\left(\lambda_i^{(m)}, v_i^{(m)} = Q_m u_i^{(m)}\right)$ finding up to *nev* of those such that $h_{m+1,m}\left|\left(u_i^{(m)}\right)_m\right| < tol$.

With the GMRES and MINRES methods for solving sparse systems of equations, we found it necessary to restart the Arnoldi or Lanczos process using the most recently estimated solution as the initial vector. Recall that our approach to the eigenvalue problem for a dense matrix was to compute one eigenvalue at a time using deflation. After computing eigenvalue λ_k, we "deflated" the problem to computing an eigenvalue in the $(k-1)\times(k-1)$ submatrix. Both restart and deflation are components of production quality algorithms for computing a few eigenpairs of a sparse matrix.

22.2.2 Estimating Eigenvalues Using Restart

Assume *nev* eigenvalues are required. If $k < nev$ of the current Ritz pairs are sufficiently accurate, we can retain those and then restart Arnoldi with an improved initial vector in hopes of estimating the remaining eigenvalues. Such an algorithm is said to use *explicit restart*. In the next section, we will discuss more sophisticated restart strategies. For now, we use the simple strategy of restarting with the current Ritz vector.The following outline summarizes this algorithm.

Outline of Explicit Arnoldi with Restart

1. Compute the initial starting vector v_0, and let $k = 1$.
2. Using initial vector v_0, compute the Arnoldi decomposition $AQ_m = Q_m H_m + h_{m+1,m}\, q_{m+1} e_m^T$.
3. Compute the eigenvectors $\left\{u_1^{(m)}, u_2^{(m)}, \ldots, u_m^{(m)}\right\}$ and eigenvalues $\left\{\lambda_1^{(m)}, \lambda_2^{(m)}, \ldots, \lambda_m^{(m)}\right\}$ of H_m.
4. Sort the eigenvalues and associated eigenvectors in descending order of eigenvalue magnitude.
5. See if Ritz pair k satisfies the error tolerance $h_{m+1,m}\left|\left(u_k^{(m)}\right)_m\right| < tol$. If so, increase k and return if $k \geq nev$.
6. Let v_0 be the normalized current Ritz vector, $\frac{Q_m u_k}{\|Q_m u_k\|_2}$, and go to step 2, unless some maximum number of iterations are exceeded.

The success of this algorithm depends on the matrix and the initial approximation v_0. The implementation of this method is left to the problems.

22.2.3 A Restart Method Using Deflation

We assume that we want a few of the largest eigenvalues in magnitude and that those eigenvalues are real. We will deal with complex eigenvalues in Section 22.3. Suppose we have found $k-1$ approximate unit eigenvectors of A, $\left\{ v_1 \;\; v_2 \;\; \ldots \;\; v_{k-2} \;\; v_{k-1} \right\}$ and an additional unit eigenvector, v_k, orthogonal to the $k-1$ vectors. Since $h_{m+1,m} \left| \left(u_k^{(m)} \right)_m \right|$ is small, the vectors $\left\{ v_1 \;\; v_2 \;\; \ldots \;\; v_{k-2} \;\; v_{k-1} \;\; v_k \right\}$ satisfy

$$H_m(1:k,\, 1:k) \simeq Q_m^T(:,\, 1:k)\, A Q_m(:,\, 1:k)\,. \tag{22.4}$$

In other words, span $\left\{ v_1 \;\; v_2 \;\; \ldots \;\; v_{k-2} \;\; v_{k-1} \;\; v_k \right\}$ is approximately an invariant subspace of A. We need this result in the following algorithm development.

Deflation was a primary tool in computing eigenvalues for dense matrices (see Sections 18.5 and 18.6). It allowed us to gradually reduce the size of the problem, saving computing time, and yielding higher accuracy. Assume that Arnoldi is started using the initial vector, v_0, to obtain the decomposition $AQ_m = Q_m H_m + h_{m+1,m}\, q_{m+1} e_m^{\mathrm{T}}$. Compute the eigenvalues and eigenvectors of H_m, and choose the pair $\left(\lambda_1^{(m)},\, u_1^{(m)} \right)$ with the largest eigenvalue in magnitude. Then compute the corresponding Ritz pair $\left(\lambda_1^{(m)},\, v_1^{(m)} = Q_m u_1^{(m)} \right)$, normalize $v_1^{(m)}$ and compute the residual

$$\left\| r_1^{(m)} \right\|_2 = h_{m+1,m} \left| \left(u_1^{(m)} \right)_m \right|.$$

If $\left\| r_1^{(m)} \right\|_2 \geq$ tol, restart the Arnoldi process, and continue until $\left\| r_1^{(m)} \right\|_2 <$ tol. At this point, we have an approximate eigenpair $\left(\lambda_1^{(m)},\, v_1^{(m)} \right)$, and are ready to compute the next pair $\left(\lambda_2^{(m)},\, v_2^{(m)} \right)$. Apply Equation 22.4 and obtain

$$H_m(1:1,\, 1:1) = Q_m(:,\, 1:1)^{\mathrm{T}}\, A Q_m(:,\, 1:1)\,,$$

so that $H_m(1:1\,, 1:1)$ is a 1×1 matrix containing the estimated eigenvalue $\lambda_1^{(m)}$. H has the approximate form

$$\begin{array}{|c|ccccc|}
\hline
\mathrm{X} & \mathrm{X} & \mathrm{X} & \ldots & \mathrm{X} & \mathrm{X} \\
0 & \mathrm{X} & \mathrm{X} & \ldots & \mathrm{X} & \mathrm{X} \\
0 & \mathrm{X} & \mathrm{X} & \mathrm{X} & \ldots & \mathrm{X} \\
0 & 0 & \mathrm{X} & \mathrm{X} & \ddots & \\
\vdots & \vdots & \vdots & \ddots & \ddots & \mathrm{X} \\
0 & 0 & 0 & \ldots & \mathrm{X} & \mathrm{X} \\
\hline
\end{array}$$

$k = 1$

where h_{11} is the estimated eigenvalue. The Ritz pair $\left(\lambda_1^{(m)},\, v_1^{(m)} \right)$ is said to be *locked*, since no more operations will affect column 1 of H_m. Execute the Arnoldi iteration beginning with $k = 2$, compute the eigenvalues and eigenvectors of H_m, and sort them. Select the pair $\left(\lambda_2^{(m)},\, u_2^{(m)} \right)$ and compute $v_2^{(m)} = Q_m u_2^{(m)}$. It is essential that each new vector be orthogonal to all the previous ones, so make the vector $v_2^{(m)}$ be orthogonal to $v_1^{(m)}$ by using the Gram-Schmidt process. Determine if $\left(\lambda_2^{(m)},\, v_2^{(m)} \right)$ satisfies

$$h_{m+1,m} \left| \left(u_2^{(m)} \right)_m \right| < \text{tol}.$$

If not, restart and repeat the process until it does. At that point, apply Equation 22.4,

$$H_m(1:2,\, 1:2) = Q_m(:,\, 1:2)^{\mathrm{T}}\, A Q_m(:,\, 1:2)\,,$$

and H_m has the approximate form

$$\begin{array}{cc|cccc}
\mathrm{X} & \mathrm{X} & \mathrm{X} & \ldots & \mathrm{X} & \mathrm{X} \\
0 & \mathrm{X} & \mathrm{X} & \ldots & \mathrm{X} & \mathrm{X} \\
\hline
0 & 0 & \mathrm{X} & \mathrm{X} & \ldots & \mathrm{X} \\
0 & 0 & \mathrm{X} & \mathrm{X} & \ddots & \\
\vdots & \vdots & \vdots & \ddots & \ddots & \mathrm{X} \\
0 & 0 & 0 & \ldots & \mathrm{X} & \mathrm{X}
\end{array}$$

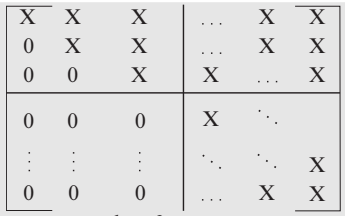
$k = 2$

Now lock $\left(\lambda_1^{(m)}, v_1^{(m)}\right)$ and $\left(\lambda_2^{(m)}, v_2^{(m)}\right)$ by leaving columns 1 and 2 of H_m alone, increase k to 3, and continue until the error tolerance is satisfied. After executing

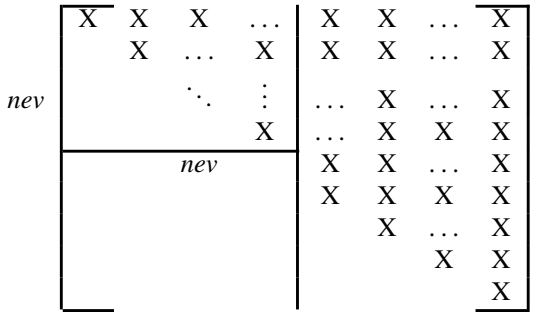
$$H_m(1:3, 1:3) = Q_m(:, 1:3)^{\mathrm{T}} A Q_m(:, 1:3),$$

H_m approximately has the form (Table 22.1)

TABLE 22.1 Restart Method Using Deflation



Our ultimate aim is to reduce the upper $nev \times nev$ submatrix of H_m to an upper-triangular matrix with the estimated eigenvalues of A on its diagonal.



Using the preceding discussion, we are now in a position to outline an algorithm for estimating *nev* eigenpairs of matrix A.

Outline of Arnoldi with Deflation and Explicit Restart

1. Select a starting vector v_0 with $\|v_0\|_2 = 1$. Use a random vector if a good starting value is not known. Let $k = 1$.

2. Loop

a. Execute the Arnoldi iteration

```
% k-1 columns are good. Start with column k.

for j = k:m do
   ...
end for
```

b. Compute the eigenvalues and eigenvectors of H_m and sort them. Pick the eigenpair $\left(\lambda_k^{(m)}, u_k^{(m)}\right)$, and let $\tilde{v}_k^{(m)} = V_m u_k^{(m)}$.

c. Othogonalize $\tilde{v}_k^{(m)}$ against all the previous Ritz vectors $\left\{v_1^{(m)}, v_2^{(m)}, \ldots, v_{k-1}^{(m)}\right\}$ and let $v_k^{(m)} = \tilde{v}_k^{(m)} / \left\|\tilde{v}_k^{(m)}\right\|_2$.

d. Compute $e_k = h_{m+1,m}\left|\left(u_k^{(m)}\right)_m\right|$.

e. If $e_k < tol$,

i Compute $H_m(1:k, 1:k) = Q_m(:, 1:k)^T A Q(:, 1:k)$.

ii Let $k = k + 1$ (deflate)

iii If $k \geq nev$, then return else go to 2 (restart).

f. Go to 2 (restart)

The function `eigsbexplicit` in the book software distribution implements Algorithm 22.3. Type `help eigsbexplicit` to determine the calling sequence.

Remark 22.1. Recall from Section 21.8.1 that the Lanczos vectors can lose orthogonality due to roundoff error. To a lesser extent, this is also true of the Arnoldi process. For that reason, the NLALIB function `arnoldi` has a fourth parameter, `reorthog`. When `reorthog = 1`, reorthogonalization is performed.

Example 22.3. We test eigsbexplicit first with the 2233 × 2233 symmetric matrix `lshp2233.mat`, followed by using eigsbexplicit with the 1030 × 1030 nonsymmetric matrix ORSIIR_1.mat. This matrix presents a much greater challenge, and we are able to adequately compute only three eigenvalues. The function `residchk` in the software distribution takes arguments A, V, D and for each eigenvalue `D(i,i)` with associated eigenvector `V(:,i)`, and prints $\|AV(:, i) - D(i, i)\, V(:, i)\|_2$, $1 \leq i \leq \text{size}(\text{D}, 1)$. In addition, it returns the average of the residuals. This examples suppresses the output of the average. ■

```
>> x0 = rand(2233,1);
>> nev = 6;
>> m = 50;
>> tol = 1.0e-6;
>> maxiter = 100;
>> reorthog = 1;
[V,D] = eigsbexplicit(lshp2233, x0, nev, m, tol, maxiter, reorthog);
diag(D)

ans =
    6.9860
    6.9717
    6.9547
    6.9533
    6.9331
    6.9238

>> residchk(lshp2233,V,D)
Eigenpair 1 residual = 9.51284e-08
Eigenpair 2 residual = 6.16388e-08
Eigenpair 3 residual = 8.59643e-07
Eigenpair 4 residual = 5.34262e-07
Eigenpair 5 residual = 1.53693e-07
Eigenpair 6 residual = 3.43595e-08
>> norm(V'*lshp2233*V-D)
ans =
    2.6078e-07
>> x0 = rand(1030,1);
>> nev = 6;
>> m = 20;
>> [V,D] = eigsbexplicit(ORSIIR_1, x0, nev, m, tol, maxiter, reorthog);
>> residchk(ORSIIR_1,V,D)
Eigenpair 1 residual = 1.62177e-10
Eigenpair 2 residual = 6.45912e-09
Eigenpair 3 residual = 1.96075e-05
Eigenpair 4 residual = 9597.58
Eigenpair 5 residual = 9586.87
Eigenpair 6 residual = 9588.03
>> norm(V'*ORSIIR_1*V - D)
ans =
     9.597579997195124e+03
```

22.2.4 Restart Strategies

In our discussion so far, we have restarted with the current Ritz eigenvector $v_k^{(m)}$. We now discuss better strategies for the Arnoldi restart. Clearly, our best strategy is to select a vector v such that $H_m(i+1, i) = 0$. In this case, $Q(:, 1 : i)$ is in an invariant subspace of dimension i. The Arnoldi iteration stops at iteration i, and the i eigenvalues are exact [3,

p. 126, 99]. Therefore, a reasonable strategy is to select a vector close to being in an invariant subspace of dimension less than or equal to m. One approach is to choose to restart with a vector that is a linear combination of the current Ritz vectors

$$\tilde{v}_k^{(m)} = \sum_{i=1}^{m} c_i v_k^{(m)}.$$

This is a special case of the approach we will take that involves constructing a polynomial which, when applied to A, can improve the current Ritz vector prior to restarting Arnoldi. To understand how we choose such a polynomial, suppose that matrix A has n linearly independent eigenvectors $v_1, v_2, \ldots, v_n$ that form a basis for $\mathbb{R}^n$. If we choose any vector $w \in R^n$, then there are constants c_i such that

$$w = \sum_{i=1}^{n} c_i v_i.$$

We are interested in approximating k of these eigenvectors, so write the sum as

$$w = \sum_{i=1}^{k} c_i v_i + \sum_{i=k+1}^{n} c_i v_i.$$

Let p_r be a polynomial, compute $p_r(A)$, apply it to w and obtain

$$p_r(A)\, w = \sum_{i=1}^{k} c_i p_r(A)\, v_i + \sum_{i=k+1}^{n} c_i p_r(A)\, v_i.$$

This can be written as (Problem 22.2)

$$p_r(A)\, w = \sum_{i=1}^{k} c_i p_r(\lambda_i)\, v_i + \sum_{i=k+1}^{n} c_i p_r(\lambda_i)\, v_i. \tag{22.5}$$

If we build p_r so that $p_r(\lambda_i), k+1 \le i \le n$, are small relative to $p_r(\lambda_i), 1 \le i \le k$, then $p_r(A)\, w$ will be dominated by the eigenvalues in which we are interested. This is termed a *filtering polynomial.* There are a number of choices for the filtering polynomial. Saad [3, pp. 165-169] and Stewart [5, pp. 321-325] discuss some choices. Consider the polynomial

$$p_r(A) = \left(A - \lambda_{nev+1}^{(m)} I\right)\left(A - \lambda_{nev+2}^{(m)} I\right) \ldots \left(A - \lambda_m^{(m)} I\right), \tag{22.6}$$

where the $\{\lambda_i^{(m)}\}$, $nev + 1 \le i \le m$ are the Ritz values we do not want. This has the effect of making $p_r(A)\, w$ dominated by a linear combination of the eigenvectors we care about (Equation 22.5). However, its computation is too costly because we include all the unwanted eigenvalues $\left\{\lambda_i^{(m)}\right\}$, $nev + 1 \le i \le m$. Consider the factors $(A - \lambda_i^{(m)} I), nev + 1 \le i \le m$ as shifts away from the eigenvalues we do not want, as opposed to shifts closer to eigenvalues we do want. It turns out that Equation 22.6 can be evaluated quickly by using the implicit QR iteration described in Section 18.8, and this is the basis for the implicitly restarted Arnoldi method discussed in Section 22.3. Another approach is to use Chebyshev polynomial filters (see Ref. [3, pp. 169-178]).

22.3 THE IMPLICITLY RESTARTED ARNOLDI METHOD

In practice, the *implicitly restarted Arnoldi method* is used to find some eigenvalues/eigenvectors of a large sparse nonsymmetric matrix. The MATLAB function `eigs` uses a version of this algorithm. We will discuss the method, provide a simplified algorithm, and a MATLAB implementation. This implementation gives good results in many cases, but in no way competes with `eigs`.

In Section 22.2, we saw that an effective approach to estimating a few eigenpairs involves both restarting and deflation. A filter polynomial enhances convergence, and by using the Francis algorithm (implicitly shifted QR), the filtering

polynomial 22.6 can be evaluated in only $O\left(m^2\right)$ flops. The following presentation derives from Saad [3, pp. 166-169]. We begin with an outline of the algorithm, and follow it with some of the details.

Outline of the Implicitly Restarted Arnoldi Method

Assume we are interested in *nev* eigenvalues, leaving $p = m - nev$ eigenvalues in which we are not interested. Begin with a unit vector v.

a. Compute the m step Arnoldi decomposition, $AQ_m = Q_m H_m + h_{m+1,m} q_{m+1} e_m^{\mathrm{T}}$.

b. Loop until computing *nev* eigenvalues or exceeding a maximum number of iterations

Find the m eigenpairs $\left(\lambda_i, v_i^{(m)}\right)$ *of* H_m.

Compute the implicit QR decomposition using the $m - nev$ *unwanted eigenvalues as the shifts. This evaluates the filter polynomial* $p(A)\, v_k^{(m)} = (A - \lambda_{nev+1} I)(A - \lambda_{nev+2} I) \ldots (A - \lambda_m I)\, v_k^{(m)} = \tilde{v}_k^{(m)}$ *that enhances the approximate eigenvector* $v_k^{(m)}$*, and at the same time builds an nev step Arnoldi decomposition. This decomposition is said to be* compressed.

c. If the eigenvalue λ_k corresponding to the eigenvector $\tilde{v}_k^{(m)}$ has converged, lock it. If convergence has not occurred or if eigenvalues remain to be found, extend the factorization to an m step factorization by applying $m - nev$ additional Arnoldi steps.

Assume that we have computed the Arnoldi decomposition

$$AV_m = V_m H_m + h_{m+1,m}\, v_{m+1} e_m^{\mathrm{T}}. \tag{22.7}$$

Using Equation 22.7, we have

$$\begin{aligned} \left(A - \lambda_{u_1}^{(m)} I\right) V_m &= AV_m - \lambda_{u_1}^{(m)} V_m \\ &= V_m H_m + h_{m+1,m}\, v_{m+1} e_m^{\mathrm{T}} - \lambda_{u_1}^{(m)} V_m \\ &= V_m \left(H_m - \lambda_{u_1}^{(m)} I\right) + h_{m+1,m}\, v_{m+1} e_m^{\mathrm{T}}. \end{aligned} \tag{22.8}$$

Compute the implicit QR decomposition of H_m with shift $\lambda_{u_1}^{(m)}$

$$H_m - \lambda_{u_1}^{(m)} I = Q_1 R_1.$$

Using it in Equation 22.8 results in

$$\left(A - \lambda_{u_1}^{(m)} I\right) V_m = V_m Q_1 R_1 + h_{m+1,m}\, v_{m+1} e_m^{\mathrm{T}} \tag{22.9}$$

$$\left(A - \lambda_{u_1}^{(m)} I\right)(V_m Q_1) = (V_m Q_1) R_1 Q_1 + h_{m+1,m}\, v_{m+1} e_m^{\mathrm{T}} Q_1 \tag{22.10}$$

$$A(V_m Q_1) = (V_m Q_1)\left(R_1 Q_1 + \lambda_{u_1}^{(m)} I\right) + h_{m+1,m}\, v_{m+1} e_m^{\mathrm{T}} Q_1 \tag{22.11}$$

Let

$$\begin{aligned} H_m^{(1)} &= R_1 Q_1 + \lambda_{u_1}^{(m)} I, \\ \left(b_{m+1}^{(1)}\right)^{\mathrm{T}} &= e_m^{\mathrm{T}} Q_1, \\ V_m^{(1)} &= V_m Q_1. \end{aligned}$$

Using these definitions in Equation 22.11, we have

$$AV_m^{(1)} = V_m^{(1)} H_m^{(1)} + h_{m+1,m}\, v_{m+1} \left(b_{m+1}^{(1)}\right)^{\mathrm{T}}. \tag{22.12}$$

We now make some observations about Equation 22.12:

- $H_m^{(1)} = R_1 Q_1 + \lambda_{u_1}^{(m)} I$ is the matrix that results from a step of the standard QR algorithm applied to H_m with shift $\lambda_{u_1}^{(m)}$.

- $H_m^{(1)}$ is an upper Hessenberg matrix.
- Equation 22.12 is essentially an Arnoldi decomposition with e_m^{T} replaced by $\left(b_{m+1}^{(1)}\right)^{\mathrm{T}}$.
- The first column of $V_m^{(1)}$ is a multiple of $\left(A-\lambda_{u_1}^{(m)}I\right)v_1^{(m)}$, where $v_1^{(m)}$ is the first column of V_m. We determine this as follows:
  - Multiply Equation 22.9 by e_1:

$$\left(A-\lambda_{u_1}^{(m)}I\right)V_m e_1 = (V_m Q_1)\,R_1 e_1 + h_{m+1,m}\,v_{m+1}e_m^{\mathrm{T}}e_1.$$

  $V_m e_1 = v_1$, and R_1 is upper triangular, so

$$\left(A-\lambda_{u_1}^{(m)}I\right)v_1 = V_m^{(1)}\begin{bmatrix} r_{11} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + 0 = r_{11}v_1^{(m)},$$

  and the first column of $V_1^{(m)}$, $v_1^{(m)}$, is a multiple, $\frac{1}{r_{11}}$, of $\left(A-\lambda_{u_1}^{(m)}I\right)v_1$.
- The columns of V_m are orthonormal, and Q_1 is an orthogonal matrix, so the columns of $V_m^{(1)}$ are orthonormal (Problem 22.1).

Now apply the second shift to Equation 22.12 in the same fashion as we did in Equation 22.8:

$$\left(A-\lambda_{u_2}^{(m)}\right)V_m^{(1)} = V_m^{(1)}\left(H_m^{(1)}-\lambda_{u_2}^{(m)}I\right) + h_{m+1,m}\,v_{m+1}\left(b_{m+1}^{(1)}\right)^{\mathrm{T}}. \tag{22.13}$$

Now, apply the implicit QR decomposition and obtain

$$H_m^{(1)} - \lambda_{u_2}^{(m)}I = Q_2 R_2,$$

and multiply Equation 22.13 by Q_2 on the right. This gives (Problem 22.3)

$$AV_m^{(2)} = V_m^{(2)}H_m^{(2)} + h_{m+1,m}\,v_{m+1}\left(b_{m+1}^{(2)}\right)^{\mathrm{T}}, \tag{22.14}$$

where $H_m^{(2)} = R_2Q_2 + \lambda_{u_2}^{(m)}I$ and $V_m^{(2)} = V_m^{(1)}Q_2$. Continuing the argument presented in Ref. [3, pp. 167-168], there results a decomposition of the form

$$A\hat{V}_{m-2} = \hat{V}_{m-2}\hat{H}_{m-2} + \hat{h}_{m+1,m}\hat{v}_{m-1}e_m^{\mathrm{T}}.$$

This decomposition is exactly the one that would be obtained by executing $(m-2)$ steps of the Arnoldi process to the unit vector obtained from $\left(A-\lambda_{u_1}^{(m)}I\right)\left(A-\lambda_{u_2}^{(m)}I\right)v_1$. At this point, we can accept an eigenvalue or perform two more Arnoldi steps to obtain an m step Arnoldi decomposition.

The process we have described applies to a filter of degree 2. If a filter of degree $m-nev > 2$ is required, the algorithm results in an nev step Arnoldi decomposition. If necessary, apply $m-nev$ more Arnoldi steps to obtain an m step decomposition. Algorithm 22.1 specifies the implicit Arnoldi process. It features a version of the Arnoldi decomposition that returns V, H, and f such that $AV_m = V_mH_m + f\,e_m^{\mathrm{T}}$. The algorithm either begins with a decomposition extending m steps or continues from a partial decomposition.

Remark 22.2. If an eigenvalue is real, perform a single shift during the implicit QR decomposition; otherwise, perform a double shift to obtain a complex conjugate pair of eigenvalues.

Algorithm 22.1 The Implicitly Restarted Arnoldi Process

```
function EIGSB(A,nev,m,tol,maxiter)
  k=1
  Allocate matrices V^{n×m} and H^{m×m}
  f=rand(n,1)
  f = f/‖f‖_2
  % perform an m step Arnoldi decomposition of A.
  [V, H, f]=arnoldif(A,V,H,f,k,m)
  iter=0
  while true do
    iter=iter+1
    % find the eigenvalues and eigenvectors of H
    [ UH, DH ] = eig(H)
    sigma=diag(UH)
    sort the eigenvalues from largest to smallest in magnitude
    Q = I^{m×m}
    % use sigma(nev+1), ..., sigma(m) as the shifts
    j=m
    while j ≥ nev + 1 do
      lambda=sigma(j)
      alpha=imag(lambda)
      if |alpha| > 0 then
        % the eigenvalue is complex. use a double shift.
        beta=real(lambda)
        [ Qj, Rj ] = implicit double shift QR (beta,alpha)
        j=j-2
      else
        % the eigenvalue is real. use a single shift.
        [Qj, Rj] = implicit single shift QR (lambda)
        j=j-1
      end if
      H = Qj^T * H * Qj
      Q = Q * Qj
    end while
    % compute the residual norm for the kth eigenpair.
    u=UH(:,k)
    residnorm = ‖f‖_2 |u(m)|
    % lock v_k if the tolerance is obtained
    if residnorm < tol then
      if k<nev then
        k=k+1
      else
        return [ V_m(:,1 : nev) , diag(sigma(1 : nev)) ]
      end if
    end if
    % build an m step decomposition from the nev step one.
    betak=H(nev+1,nev)
    sigmak=Q(m,nev)
    fk=V(:,nev+1)*betak+f*sigmak
    V(:,1:nev)=V(:,1:m)*Q(:,1:nev)
    [V, H, f]=arnoldif(A,V(:,1:nev),H(1:nev,1:nev),fk,nev+1,m)

    if iter ≥ maxiter then
      print(Error: could not compute nev eigenvalues within specified number of iterations.)
      terminate
    end if
  end while
end function
```

NLALIB: The function `eigsb` implements Algorithm 22.1. It is supported by the function `arnoldif` that computes the Arnoldi decomposition in the form given by Equation 22.1. `eigsb` returns a vector of `nev` eigenvalues or the eigenvectors in a matrix `V` and a diagonal matrix `D` with the `nev` corresponding eigenvalues. Recall that the function avgresid = residchk(A,V,D) prints the residuals $\|AV(:,i) - D(i,i)\,V(:,i)\|_2$, $1 \le i \le \text{size}(D,1)$, and returns the average of the residuals.

Example 22.4. The following table lists four nonsymmetric matrices from actual applications. `eigsb` was applied to each matrix using the defaults, tol $= 1.0 \times 10^{-6}$, maxiter $= 100$, $nevs = 6$. Experimentation was required to determine a value of m that resulted in the average residual given in the table.

Matrix	Dimensions	Application Area
rotor2	791×791	Large helicopter rotor model
qh882	961×961	Power systems simulations
dw256A	317×317	Square dielectric waveguide
TOLS1090	1090×1090	Aeroelasticity

Matrix	rotor2	qh882	dw256A	TOLS1090
m	35	40	35	16
Average residual	1.1125e−11	1.3725e−08	6.8321e−14	6.8211e−7

■

22.3.1 Convergence of the Arnoldi Iteration

Convergence analysis of the Arnoldi iteration is complex. See Saad [3, pp. 151-159].

22.4 EIGENVALUE COMPUTATION USING THE LANCZOS PROCESS

As expected, a sparse symmetric matrix A has properties that will enable us to compute eigenvalues and eigenvectors more efficiently than we are able to do with a nonsymmetric sparse matrix. Also, much more is known about convergence properties for the eigenvalue computations. We begin with the following lemma and then use it to investigate approximate eigenpairs of A.

Lemma 22.1. *Let A be an $n \times n$ symmetric matrix. Let θ be a real number and x be an arbitrary vector in $\mathbb{R}^n$ with $x \neq 0$. Let $\mu = \|(A - \theta I)\,x\|_2 \,/\, \|x\|_2$. Then there is an eigenvalue of A in the interval $\theta - \mu \le \lambda \le \theta + \mu$.*

Proof. Let

$$A = PDP^{\mathrm{T}} = \sum_{i=1}^{n} \lambda_i p_i p_i^{\mathrm{T}}$$

be the spectral decomposition of A (Theorem 19.1). It follows that:

$$\begin{aligned}(A - \theta I)\,x &= \left(PDP^{\mathrm{T}} - \theta PP^{\mathrm{T}}\right)x = \sum_{i=1}^{n} \left(\lambda_i p_i p_i^{\mathrm{T}} - \theta p_i p_i^{\mathrm{T}}\right)x \\ &= \sum_{i=1}^{n} \left(\lambda_i \left(p_i^{\mathrm{T}} x\right) p_i - \theta \left(p_i^{\mathrm{T}} x\right) p_i\right) \\ &= \sum_{i=1}^{n} \left(\lambda_i - \theta\right)\left(p_i^{\mathrm{T}} x\right) p_i\end{aligned}$$

Taking the norm of the equality and noting that the $\{p_i\}$ are orthonormal, we obtain

$$\|(A - \theta I)\,x\|_2^2 = \sum_{i=1}^{n} \left(\lambda_i - \theta\right)^2 \left(p_i^{\mathrm{T}} x\right)^2.$$

Note that $\sum_{i=1}^{n} p_i p_i^{\mathrm{T}} = I$ (Problem 22.4). Let λ_k be the eigenvalue closest to θ, i.e., $|\lambda_k - \theta| \leq |\lambda_i - \theta|$ for all i, and we have

$$\begin{aligned}
\|(A - \theta I)\, x\|_2^2 &\geq (\lambda_k - \theta)^2 \sum_{i=1}^{n} \left(p_i^{\mathrm{T}} x\right)^2 \\
&= (\lambda_k - \theta)^2 \sum_{i=1}^{n} \left(p_i^{\mathrm{T}} x p_i^{\mathrm{T}} x\right) = (\lambda_k - \theta)^2 \sum_{i=1}^{n} \left(x^{\mathrm{T}} p_i p_i^{\mathrm{T}} x\right) \\
&= (\lambda_k - \theta)^2\, x^{\mathrm{T}} \left(\sum_{i=1}^{n} p_i p_i^{\mathrm{T}}\right) x = (\lambda_k - \theta)^2\, x^{\mathrm{T}} I x \\
&= (\lambda_k - \theta)^2\, \|x\|_2^2
\end{aligned}$$

This implies that

$$\mu = \|(A - \theta I)\, x\|_2 \,/\, \|x\|_2 \geq |\lambda_k - \theta|\,,$$

and so there is an eigenvalue λ_k in the interval $\theta - \mu \leq \lambda \leq \theta + \mu$. □

Recall that the Lanczos process for a symmetric matrix discussed in Section 21.8 is the Arnoldi process for a symmetric matrix and takes the form

$$AQ_m = Q_m T_m + h_{m+1,m} q_{m+1} e_m^{\mathrm{T}},$$

where

$$T_m = \begin{bmatrix}
\alpha_1 & \beta_1 & & & \\
\beta_1 & \alpha_2 & \beta_2 & & \\
 & \beta_2 & \ddots & \ddots & \\
 & & \ddots & \alpha_{m-1} & \beta_{m-1} \\
 & & & \beta_{m-1} & \alpha_m
\end{bmatrix},$$

is symmetric tridiagonal, and Q_m is orthogonal. We will proceed like we did for nonsymmetric matrices and use Ritz pairs of T_m to approximate eigenpairs of A. Let $\mu = \lambda_i^{(m)}$ be a Ritz value and $u_i^{(m)}$ be a corresponding eigenvector obtained from T_m so that $T_m u_i^{(m)} = \lambda_i^{(m)} u_i^{(m)}$, and let $v_i^{(m)} = Q_m u_i^{(m)}$ be the Ritz vector. Applying the same operations we used to derive Equation 22.3, we obtain

$$\left\| \left(A - \lambda_i^{(m)} I\right) v_i^{(m)} \right\|_2 = h_{m+1,m} \left| \left(u_i^{(m)}\right)_m \right|.$$

Since $v_i^{(m)} = Q_m u_i^{(m)}$, $\left\| u_i^{(m)} \right\|_2 = 1$, and Q_m is orthogonal, we have $\left\| v_i^{(m)} \right\|_2 = 1$, and so

$$\mu = \left\| \left(A - \lambda_i^{(m)} I\right) v_i^{(m)} \right\| \Big/ \left\| v_i^{(m)} \right\| = h_{m+1,m} \left| \left(u_i^{(m)}\right)_m \right|.$$

It follows from Lemma 22.1 that there is an eigenvalue λ such that

$$\lambda_i^{(m)} - \mu \leq \lambda \leq \lambda_i^{(m)} + \mu,$$

so

$$-\mu \leq \lambda - \lambda_i^{(m)} \leq \mu,$$

and

$$\left| \lambda - \lambda_i^{(m)} \right| \leq h_{m+1,m} \left| \left(u_i^{(m)}\right)_m \right|. \tag{22.15}$$

Equation 22.15 indicates we will have a good approximation to an eigenpair of A *as* long as $h_{m+1,m}\left|\left(u_i^{(m)}\right)_m\right|$ is small. We had not such a guarantee for a nonsymmetric matrix.

Poor convergence can result from a bad choice of the starting vector, so a random vector is a good choice. Multiple eigenvalues or eigenvalues that are very close to each other particularly cause problems. As discussed in Section 21.8.1, roundoff error can cause lack or orthogonality among the Lanczos vectors, and this happens as soon as Ritz vectors have converged accurately enough to eigenvectors [2, p. 565]. The loss of orthogonality can cause simple eigenvalues to appear as multiple eigenvalues, and these are called *ghost eigenvalues* [2, p. 566] (Problems 22.6 and 22.14). In our implementation of the implicitly restarted Lanczos process, we will perform full reorthogonalization. The function `eigssymb` in the software distribution implements the implicitly restarted Lanczos process. The only real differences between this function and `eigsb` is the use of the Lanczos decomposition instead of Arnoldi and the fact that only a single shift is necessary since the matrix has real eigenvalues.

Example 22.5. The very ill-conditioned (approximate condition number 2.5538×10^{17}) symmetric 60000×60000 matrix `Andrews` obtained from the Florida sparse matrix collection was used in a computer graphics/vision problem. As we know, even though the matrix is ill-conditioned, its eigenvalues are well conditioned (Theorem 19.2). The MATLAB statements time the approximation of the six largest eigenvalues and corresponding eigenvectors using `eigsymb`. A call to `residchk` outputs the residuals.

```
>> tic;[V, D] = eigssymb(Andrews, 6, 50);toc;
Elapsed time is 5.494509 seconds.
>> residchk(Andrews,V,D)
Eigenpair 1 residual = 3.59008e-08
Eigenpair 2 residual = 1.86217e-08
Eigenpair 3 residual = 2.31836e-08
Eigenpair 4 residual = 7.68169e-08
Eigenpair 5 residual = 4.10453e-08
Eigenpair 6 residual = 6.04127e-07

ans =
     1.332825500776470e-07
```

■

22.4.1 Mathematically Provable Properties

This section presents some theoretical properties that shed light on the use and convergence properties of the Lanczos method.

In Ref. [76, p. 245], Scott proves the following theorem:

Theorem 22.1. *Let A be a symmetric matrix with eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, $\delta_A = min_{k \neq i} |\lambda_i - \lambda_k|$. Then there exists a starting vector v_0 such that for the exact Lanczos algorithm applied to A with v_0, at any step $j < n$ the residual norm*

$$\|Av_i - \lambda_i v_i\|_2$$

of any Ritz pair (λ_i, v_i) will be larger than $\delta_A/4$.

This theorem says that if the spectrum of A is such that $\delta_A/4$ is larger than some given convergence tolerance, then there exist poor starting vectors which delay convergence until the nth step. Thus, the starting vector is a critical component in the performance of the algorithm. If a good starting vector is not known, then use a random vector. We have used this approach in our implementation of `eigssymb`.

The expository paper by Meurant and Stratkos [70, Theorem 4.3, p. 504] supports the conclusion that orthogonality can be lost only in the direction of converged Ritz vectors. This result allows the development of sophisticated methods for maintaining orthogonality such as selective reorthogonalization [2, pp. 565-566], [6, pp. 116-123].

There are significant results concerning the rate of convergence of the Lanczos algorithm. Kaniel [77] began investigating these problems. Subsequently, the finite precision behavior of the Lanczos algorithm was analyzed in great depth by Chris Paige in his Ph.D. thesis [78]; see also Paige [79–81]. He described the effects of rounding errors in the Lanczos algorithm using rigorous and elegant mathematical theory. The results are beyond the scope of this book, so we will assume exact arithmetic in the following result concerning convergence proved in Saad [3, pp. 147-150].

Theorem 22.2. *Let A be an n-by-n symmetric matrix. The difference between the ith exact and approximate eigenvalues* λ_i *and* $\lambda_i^{(m)}$ *satisfies the double inequality*

$$0 \leq \lambda_i - \lambda_i^{(m)} \leq (\lambda_1 - \lambda_n) \left(\frac{\kappa_i^{(m)} \tan\theta \, \langle v_1, \, u_i \rangle}{C_{m-i}\,(1 + 2\gamma_i)} \right)^2,$$

where

$$C_{m-i}\,(x)$$

is the Chebyshev polynomial of degree $m - i$*,*

$$\gamma_i = \frac{\lambda_i - \lambda_{i+1}}{\lambda_{i+1} - \lambda_n},$$

$\kappa_i^{(m)}$ *is given by*

$$\kappa_1^{(m)} = 1, \quad \kappa_i^{(m)} = \prod_{j=1}^{i-1} \frac{\lambda_j^{(m)} - \lambda_n}{\lambda_j^{(m)} - \lambda_i}, \; i > 1,$$

and θ *is the angle defined in Ref. [3, p. 147].*

Remark 22.3. Error bounds indicate that for many matrices and for relatively small m, several of the largest or smallest of the eigenvalues of A are well approximated by eigenvalues of the corresponding Lanczos matrices. In practice, it is not always the case that both ends of the spectrum of a symmetric matrix are approximated accurately. However, it is generally true that at least one end of the spectrum is approximated well.

22.5 CHAPTER SUMMARY

The Power Method

The power method generates the Krylov subspace $\mathcal{K}_m\,(A, x_0) = \text{span}\left\{ x_0 \;\; Ax_0 \;\; A^2x_0 \;\; \ldots \;\; A^{k-2}x_0 \;\; A^{k-1}x_0 \right\}$, where $A^{k-1}x_0$ is the approximate eigenvector corresponding to the largest eigenvalue. This method can be effective in some cases; however, its main importance is that it leads to ideas that use a combination of vectors from the Krylov subspace.

Eigenvalue Computation Using the Arnoldi Process

Given a nonsymmetric matrix A, the basic idea is simple:

> *Perform an Arnoldi decomposition,* $AV_m = V_m H_m + h_{m+1,m}\, v_{m+1} e_m^{\mathrm{T}}$*, and use some eigenvalues* $\left\{ \lambda_1^{(m)}, \lambda_2^{(m)}, \ldots, \lambda_k^{(m)} \right\}$ *of* H_m *as approximations to the eigenvalues of* A*. The corresponding eigenvectors are* $V_m u_i^{(m)}$*, where* $u_i^{(m)}$ *is the eigenvector of* H_m *corresponding to* $\lambda_i^{(m)}$*.*

However, the implementation is not simple. The section discusses three approaches:

- Compute a few eigenvalues of H_m, sort them in descending order, and estimate the error of each using Equation 22.3. Accept the eigenvalues and optionally the corresponding eigenvectors that satisfy an error tolerance. Basically, "You get what you get."
- Accept eigenvalues satisfying the error tolerance and otherwise restart Arnoldi with the current Ritz vector or some improvement of it.
- Restart until computing eigenvalue λ_1, deflate the matrix and search for λ_2, and continue until computing the desired eigenvalues.

The Implicitly Restarted Arnoldi Method

To approximate *nev* eigenvalues and corresponding eigenvectors, use deflation and a filter polynomial such as $p_r\,(A)\, v_k^{(m)} = \left(A - \lambda_{nev+1}^{(m)} I\right) \left(A - \lambda_{nev+2}^{(m)} I\right) \ldots \left(A - \lambda_m^{(m)} I\right) v_k^{(m)}$ to provide a better restart vector. Evaluate this polynomial using the

implicitly shifted *QR* algorithm, which requires only $O\left(m^2\right)$ flops. This is the most commonly used method for computing a few eigenvalues of a large sparse matrix.

Eigenvalue Computation Using the Lanczos Process

This section discusses the implicitly shifted *QR* algorithm for computing a few eigenvalues of a large sparse symmetric matrix using the Lanczos decomposition. The Lanczos matrix is symmetric tridiagonal, and its columns tend to lose orthogonality during the Lanczos iteration, resulting in inaccurate eigenvalues. Full reorthogonalization of each new vector against the already computed ones is computationally expensive but solves the problem. Other methods such as partial reorthogonalization can be used to speed up the algorithm and retain accuracy.

Matrix symmetry has led to extensive results about the algorithm performance. For instance, the choice of the starting vector is critical, and orthogonality can be lost only in the direction of converged Ritz vectors. Also, there are results that specify convergence properties of the method.

22.6 PROBLEMS

22.1 Assume that the columns of matrix V are orthonormal and Q is an orthogonal matrix. Prove that the columns of VQ are orthonormal.

22.2 Develop Equation 22.5.

22.3 Develop Equation 22.14.

22.4 Show that $\sum_{i=1}^{n} u_i u_i^{\mathrm{T}} = I$ if $\{u_i\}$ is an orthonormal basis for $\mathbb{R}^n$. Proceed in stages.

Let v be an arbitrary vector in $\mathbb{R}^n$ and consider the product $\left(\sum_{j=1}^{n} u_i u_i^{\mathrm{T}}\right) v$. There exist constants $\{c_j\}$ so that $v = \sum_{j=1}^{n} c_j u_j$.

a. Show that $\left(\sum_{i=1}^{n} u_i u_i^{\mathrm{T}}\right) v = v$.

b. Argue that (a) implies $\sum u_i u_i^{\mathrm{T}} = I$.

22.5 This is a restatement of Exercise 6.4.36 in Ref. [23].

a. Assume that λ is an approximate eigenvalue for A with corresponding approximate eigenvector v, $\|v\|_2 = 1$. Form the residual $r = Av - \lambda v$, let $\epsilon = \|r\|_2$ and $E = -rv^{\mathrm{T}}$. Show that (λ, v) is an eigenpair of $A + E$ and $\|E\|_2 = \epsilon$.

b. Argue that (λ, v) is an exact eigenpair of a matrix that is close to A.

c. Is (λ, v) is a good approximate eigenpair of A in the sense of backward error?

22.6 If a symmetric tridiagonal matrix is unreduced (no zeros on the subdiagonal and thus none on the superdiagonal), it must have distinct real eigenvalues (Problem 19.1). When testing for ghost eigenvalues, why is knowing this result important?

22.6.1 MATLAB Problems

22.7 Using the function `biharmonic_op` in the software distribution, build a block pentadiagonal matrix of size $10,000 \times 10,000$. Use `eigs` to compute the six largest eigenvalues in magnitude, and then apply the power method in an attempt to compute the largest eigenvalue. Explain why you have great difficulty or fail to compute an accurate result.

22.8 Carry out the numerical experiment described in Example 22.2 and construct a plot like that in Figure 22.2.

22.9 Implement a function, `eigsimple`, following the outline in Section 22.2.1. It should return just approximate eigenvalues or eigenvectors and a diagonal matrix of eigenvalues, depending on the number of output arguments. Test it on the nonsymmetric matrices `qh882` and `TOLS340` by estimating up to six eigenpairs and checking them with `residchk`. Hint: For qh882, use $m = 200$, and for TOLS1090, use $m = 350$.

22.10 Implement a function, `eigsrestart`, following the outline in Section 22.2.2. Test it on the nonsymmetric matrices `bfwa398` and `west2021` by estimating maximum of six eigenpairs and checking them with `residchk`. Use $m = 100$.

22.11 Test the function eigsbexplicit on the nonsymmetric matrices `steam2`, `ORSIIR_1`, and the symmetric matrix DIMACS10. You will need to determine appropriate values of m for each matrix. Try to estimate six eigenpairs and check them with `residchk`. You might not be able to find six.

22.12 Look at the code for `eigssymb` and `lanczosf` and determine how to turn off reorthogonalization. Run your modified code with the symmetric matrix `lshp2233` and test the results using `residchk`. Turn reorthogonalization back on and execute the same steps. Explain the results.

22.13 This problem investigates ghost eigenvalues.

a. The matrix `ghosttest` in the book software distribution is a 100×100 diagonal matrix with `ghosttest(1,1) = 100` and `ghosttest(100,100) = 10`. The remaining diagonal elements are in the range (0, 1). The function `lanczosfplot` produces a plot of the Lanczos iteration number vs. the eigenvalues of `T`. Run it with `m = 35` and `reorthog = 0` and see if you can identify ghost eigenvalues. Run it again with `reorthog = 1` and compare the results.

b. Create the diagonal matrix defined in Example 21.10 using $n = 50$, lambda_1 = 100, lambda_n = 0.2, and $\rho = 0.9$. Repeat part (a) for this matrix using `m = 50`.

22.14 Modify `eigssymb` so it computes either the *nev* largest or the *nev* smallest eigenvalues of a large sparse symmetric matrix by adding a parameter, direction, that has values "*L*" or "*S*." Name the function `eigssymb2`. Test your implementation by finding the six largest and the six smallest eigenvalues of the symmetric matrices SHERMAN1.mat and Andrews.mat in the software distribution. You will need to experiment with *m* and maxiter to obtain satisfactory results.