Suppose without loss of generality that x_i is converging to e_1, so we can write $x_i = e_1 + d_i$, where $\|d_i\|_2 \equiv \epsilon \ll 1$. To prove cubic convergence, we need to show that $x_{i+1} = e_1 + d_{i+1}$ with $\|d_{i+1}\|_2 = O(\epsilon^3)$.

We first note that

$$1 = x_i^T x_i = (e_1 + d_i)^T (e_1 + d_i) = e_1^T e_1 + 2e_1^T d_i + d_i^T d_i = 1 + 2d_{i1} + \epsilon^2$$

so that $d_{i1} = -\epsilon^2/2$. Therefore

$$\rho_i = x_i^T \Lambda x_i = (e_1 + d_i)^T \Lambda (e_1 + d_i) = e_1^T \Lambda e_1 + 2e_1^T \Lambda d_i + d_i^T \Lambda d_i = \alpha_1 - \eta,$$

where $\eta \equiv -2e_1^T \Lambda d_i - d_i^T \Lambda d_i = \alpha_1 \epsilon^2 - d_i^T \Lambda d_i$. We see that

$$|\eta| \le |\alpha_1|\epsilon^2 + \|\Lambda\|_2 \|d_i\|_2^2 \le 2\|\Lambda\|_2 \epsilon^2, \tag{5.12}$$

so $\rho_i = \alpha_1 - \eta = \alpha_1 + O(\epsilon^2)$ is a very good approximation to the eigenvalue α_1.

Now we can write

$$\begin{aligned}
y_{i+1} &= (\Lambda - \rho_i I)^{-1} x_i \\
&= \left[\frac{x_{i1}}{\alpha_1 - \rho_i}, \frac{x_{i2}}{\alpha_2 - \rho_i}, \ldots, \frac{x_{in}}{\alpha_n - \rho_i} \right]^T \\
&\qquad \text{because } (\Lambda - \rho_i I)^{-1} = \operatorname{diag}\left(\frac{1}{\alpha_j - \rho_i} \right) \\
&= \left[\frac{1 + d_{i1}}{\alpha_1 - \rho_i}, \frac{d_{i2}}{\alpha_2 - \rho_i}, \ldots, \frac{d_{in}}{\alpha_n - \rho_i} \right]^T \\
&\qquad \text{because } x_i = e_1 + d_i \\
&= \left[\frac{1 - \epsilon^2/2}{\eta}, \frac{d_{i2}}{\alpha_2 - \alpha_1 + \eta}, \ldots, \frac{d_{in}}{\alpha_n - \alpha_1 + \eta} \right]^T \\
&\qquad \text{because } \rho_i = \alpha_1 - \eta \text{ and } d_{i1} = -\epsilon^2/2 \\
&= \frac{1 - \epsilon^2/2}{\eta} \cdot \left[1, \frac{d_{i2}\eta}{(1 - \epsilon^2/2)(\alpha_2 - \alpha_1 + \eta)}, \ldots, \right. \\
&\qquad \left. \frac{d_{in}\eta}{(1 - \epsilon^2/2)(\alpha_n - \alpha_1 + \eta)} \right]^T \\
&\equiv \frac{1 - \epsilon^2/2}{\eta} \cdot (e_1 + \hat{d}_{i+1}).
\end{aligned}$$

To bound $\|\hat{d}_{i+1}\|_2$, we note that we can bound each denominator using $|\alpha_j - \alpha_1 + \eta| \ge \operatorname{gap}(1, \Lambda) - |\eta|$, so using (5.12) as well we get

$$\|\hat{d}_{i+1}\|_2 \le \frac{\|d_i\|_2 |\eta|}{(1 - \epsilon^2/2)(\operatorname{gap}(1, \Lambda) - |\eta|)} \le \frac{2\|\Lambda\|_2 \epsilon^3}{(1 - \epsilon^2/2)(\operatorname{gap}(1, \Lambda) - 2\|\Lambda\|\epsilon^2)}$$

or $\|\hat{d}_{i+1}\|_2 = O(\epsilon^3)$. Finally, since $x_{i+1} = e_1 + d_{i+1} = (e_1 + \hat{d}_{i+1})/\|e_1 + \hat{d}_{i+1}\|_2$, we see $\|d_{i+1}\|_2 = O(\epsilon^3)$ as well. □

5.3.3. Divide-and-Conquer

This method is the fastest now available if you want all eigenvalues and eigenvectors of a tridiagonal matrix whose dimension is larger than about 25. (The exact threshold depends on the computer.) It is quite subtle to implement in a numerically stable way. Indeed, although this method was first introduced in 1981 [58], the "right" implementation was not discovered until 1992 [125, 129]). This routine is available as LAPACK routines `ssyevd` for dense matrices and `sstevd` for tridiagonal matrices. This routine uses divide-and-conquer for matrices of dimension larger than 25 and automatically switches to QR iteration for smaller matrices (or if eigenvalues only are desired).

We first discuss the overall structure of the algorithm, and leave numerical details for later. Let

$$\begin{aligned}
T &= \left[\begin{array}{ccccc|ccccc}
a_1 & b_1 & & & & & & & & \\
b_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & a_{m-1} & b_{m-1} & & & & & & \\
 & & b_{m-1} & a_m & b_m & & & & & \\
\hline
 & & & b_m & a_{m+1} & b_{m+1} & & & & \\
 & & & & b_{m+1} & \ddots & & & & \\
 & & & & & & \ddots & b_{n-1} & & \\
 & & & & & & b_{n-1} & a_n &
\end{array}\right] \\
&= \left[\begin{array}{ccccc|ccccc}
a_1 & b_1 & & & & & & & & \\
b_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & a_{m-1} & b_{m-1} & & & & & & \\
 & & b_{m-1} & a_m - b_m & & & & & & \\
\hline
 & & & & a_{m+1} - b_m & b_{m+1} & & & & \\
 & & & & b_{m+1} & \ddots & & & & \\
 & & & & & & \ddots & b_{n-1} & & \\
 & & & & & & b_{n-1} & a_n &
\end{array}\right] \\
&\quad + \left[\begin{array}{ccccc|ccccc}
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & b_m & b_m & & & & \\
\hline
 & & & & b_m & b_m & & & & \\
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & & & & &
\end{array}\right]
\end{aligned}$$

$$= \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + b_m \cdot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} [0, \ldots, 0, 1, 1, 0, \ldots, 0] \equiv \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + b_m v v^T.$$

Suppose that we have the eigendecompositions of T_1 and T_2: $T_i = Q_i \Lambda_i Q_i^T$. These will be computed recursively by this same algorithm. We relate the eigenvalues of T to those of T_1 and T_2 as follows.

$$\begin{aligned} T &= \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m v v^T \\ &= \begin{bmatrix} Q_1 \Lambda_1 Q_1^T & 0 \\ 0 & Q_2 \Lambda_2 Q_2^T \end{bmatrix} + b_m v v^T \\ &= \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \left(\begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} + b_m u u^T \right) \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix}, \end{aligned}$$

where

$$u = \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix} \quad v = \begin{bmatrix} \text{last column of } Q_1^T \\ \text{first column of } Q_2^T \end{bmatrix}$$

since $v = [0, \ldots, 0, 1, 1, 0, \ldots, 0]^T$. Therefore, the eigenvalues of T are the same as those of the similar matrix $D + \rho u u^T$ where $D = \left[\begin{smallmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{smallmatrix}\right]$ is diagonal, $\rho = b_m$ is a scalar, and u is a vector. Henceforth we will assume without loss of generality that the diagonal $d_1, \ldots, d_n$ of D is sorted: $d_n \leq \cdots \leq d_1$.

To find the eigenvalues of $D + \rho u u^T$, assume first that $D - \lambda I$ is nonsingular, and compute the characteristic polynomial as follows:

$$\det(D + \rho u u^T - \lambda I) = \det((D - \lambda I)(I + \rho (D - \lambda)^{-1} u u^T)). \tag{5.13}$$

Since $D - \lambda I$ is nonsingular, $\det(I + \rho (D - \lambda)^{-1} u u^T) = 0$ whenever λ is an eigenvalue. Note that $I + \rho (D - \lambda)^{-1} u u^T$ is the identity plus a rank-1 matrix; the determinant of such a matrix is easy to compute:

LEMMA 5.1. *If x and y are vectors,* $\det(I + x y^T) = 1 + y^T x$.

The proof is left to Question 5.14.

Therefore

$$\det(I + \rho (D - \lambda)^{-1} u u^T) = 1 + \rho u^T (D - \lambda)^{-1} u = 1 + \rho \sum_{i=1}^{n} \frac{u_i^2}{d_i - \lambda} \equiv f(\lambda), \tag{5.14}$$

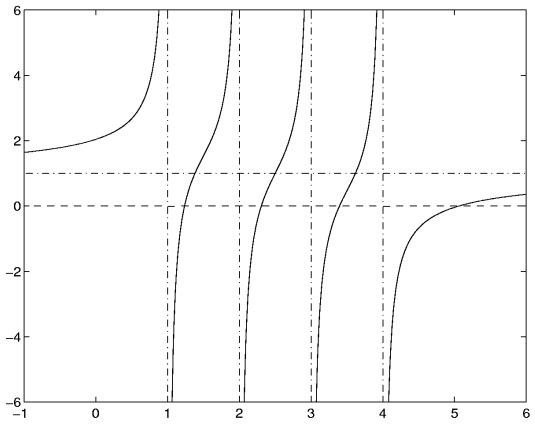

Fig. 5.2. *Graph of* $f(\lambda) = 1 + \frac{.5}{1-\lambda} + \frac{.5}{2-\lambda} + \frac{.5}{3-\lambda} + \frac{.5}{4-\lambda}$.

and the eigenvalues of T are the roots of the so-called *secular equation* $f(\lambda) = 0$. If all d_i are distinct and all $u_i = 0$ (the generic case), the function $f(\lambda)$ has the graph shown in Figure 5.2 (for $n = 4$ and $\rho > 0$).

As we can see, the line $y = 1$ is a horizontal asymptote, and the lines $\lambda = d_i$ are vertical asymptotes. Since $f'(\lambda) = \rho \sum_{i=1}^{n} \frac{u_i^2}{(d_i - \lambda)^2} > 0$, the function is strictly increasing except at $\lambda = d_i$. Thus the roots of $f(\lambda)$ are interlaced by the d_i, and there is one more root to the right of d_1 ($d_1 = 4$ in Figure 5.2). (If $\rho < 0$, then $f(\lambda)$ is decreasing and there is one more root to the left of d_n.) Since $f(\lambda)$ is monotonic and smooth on the intervals (d_i, d_{i+1}), it is possible to find a version of Newton's method that converges fast and monotonically to each root, given a starting point in (d_i, d_{i+1}). We discuss details later in this section. All we need to know here is that in practice Newton converges in a bounded number of steps per eigenvalue. Since evaluating $f(\lambda)$ and $f'(\lambda)$ costs $O(n)$ flops, finding one eigenvalue costs $O(n)$ flops, and so finding all n eigenvalues of $D + \rho u u^T$ costs $O(n^2)$ flops.

It is also easy to derive an expression for the eigenvectors of $D + u u^T$.

LEMMA 5.2. *If α is an eigenvalue of $D + \rho u u^T$, then $(D - \alpha I)^{-1} u$ is its eigenvector. Since $D - \alpha I$ is diagonal, this costs $O(n)$ flops to compute.*

Proof.

$$\begin{aligned} (D + \rho u u^T)[(D - \alpha I)^{-1} u] &= (D - \alpha I + \alpha I + \rho u u^T)(D - \alpha I)^{-1} u \\ &= u + \alpha (D - \alpha I)^{-1} u + u[\rho u^T (D - \alpha I)^{-1} u] \end{aligned}$$

$$
\begin{aligned}
&= u + \alpha(D-\alpha I)^{-1}u - u \\
&\quad \text{since } \rho u^T(D-\alpha I)^{-1}u + 1 = f(\alpha) = 1 \\
&= \alpha[(D-\alpha I)^{-1}u] \quad \text{as desired.} \quad \square
\end{aligned}
$$

Evaluating this formula for all n eigenvectors costs $O(n^2)$ flops. Unfortunately, this simple formula for the eigenvectors is not numerically stable, because two very close values of α_i can result in nonorthogonal computed eigenvectors u_i. Finding a stable alternative took over a decade from the original formulation of this algorithm. We discuss details later in this section.

The overall algorithm is recursive.

ALGORITHM 5.2. *Finding eigenvalues and eigenvectors of a symmetric tridiagonal matrix using divide-and-conquer:*

proc dc_eig (T, Q, Λ) ..... *from input* T *compute outputs* Q *and* Λ *where* $T = Q\Lambda Q^T$

if T *is* 1*-by-*1
  return $Q = 1, \Lambda = T$
else
  form $T = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m v v^T$
  call dc_eig (T_1, Q_1, Λ_1)
  call dc_eig (T_2, Q_2, Λ_2)
  form $D + \rho u u^T$ *from* $\Lambda_1, \Lambda_2, Q_1, Q_2$
  find eigenvalues Λ *and eigenvectors* Q' *of* $D + \rho u u^T$
  form $Q = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \cdot Q' =$ *eigenvectors of* T
  return Q *and* Λ
endif

We analyze the complexity of Algorithm 5.2 as follows. Let $t(n)$ be the number of flops to run dc_eig on an n-by-n matrix. Then

$$
\begin{aligned}
t(n) &= 2t(n/2) && \text{for the 2 recursive calls to dc\_eig}(T_i, Q_i, \Lambda_i) \\
&\quad + O(n^2) && \text{to find the eigenvalues of } D + \rho u u^T \\
&\quad + O(n^2) && \text{to find the eigenvectors of } D + \rho u u^T \\
&\quad + c \cdot n^3 && \text{to multiply } Q = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \cdot Q'.
\end{aligned}
$$

If we treat Q_1, Q_2, and Q' as dense matrices and use the standard matrix multiplication algorithm, the constant in the last line is $c = 1$. Thus we see that the major cost in the algorithm is the matrix multiplication in the last line. Ignoring the $O(n^2)$ terms, we get $t(n) = 2t(n/2) + cn^3$. This geometric sum can be evaluated, yielding $t(n) \approx c\frac{4}{3}n^3$ (see Question 5.15). In practice, c is usually much less than 1, because a phenomenon called *deflation* makes Q' quite sparse.

After discussing deflation in the next section, we discuss details of solving the secular equation, and computing the eigenvectors stably. Finally, we discuss how to accelerate the method by exploiting FMM techniques used in electrostatic particle simulation [122]. These sections may be skipped on a first reading.

Deflation

So far in our presentation we have assumed that the d_i are distinct, and the u_i nonzero. When this is not the case, the secular equation $f(\lambda) = 0$ will have $k < n$ vertical asymptotes, and so $k < n$ roots. But it turns out that the remaining $n - k$ eigenvalues are available very cheaply: If $d_i = d_{i+1}$, or if $u_i = 0$, one can easily show that d_i is also an eigenvalue of $D + \rho u u^T$ (see Question 5.16). This process is called *deflation*. In practice we use a threshold and deflate d_i either if it is close enough to d_{i+1} or if u_i is small enough.

In practice, deflation happens quite frequently: In experiments with random dense matrices with uniformly distributed eigenvalues, over 15% of the eigenvalues of the largest $D + \rho u u^T$ deflated, and in experiments with random dense matrices with eigenvalues approaching 0 geometrically, over 85% deflated! It is essential to take advantage of this behavior to make the algorithm fast [58, 208].

The payoff in deflation is not in making the solution of the secular equation faster; this costs only $O(n^2)$ anyway. The payoff is in making the matrix multiplication in the last step of the algorithm fast. For if $u_i = 0$, then the corresponding eigenvector is e_i, the ith column of the identity matrix (see Question 5.16). This means that the ith column of Q' is e_i, so no work is needed to compute the ith column of Q in the two multiplications by Q_1 and Q_2. There is a similar simplification when $d_i = d_{i+1}$. When many eigenvalues deflate, much of the work in the matrix multiplication can be eliminated. This is borne out in the numerical experiments presented in section 5.3.6.

Solving the Secular Equation

When some u_i is small but too large to deflate, a problem arises when trying to use Newton's method to solve the secular equation. Recall that the principle of Newton's method for updating an approximate solution λ_j of $f(\lambda) = 0$ is

1. to approximate the function $f(\lambda)$ near $\lambda = \lambda_j$ with a linear function $l(\lambda)$, whose graph is a straight line tangent to the graph of $f(\lambda)$ at $\lambda = \lambda_j$,
2. to let λ_{j+1} be the zero of this linear approximation: $l(\lambda_{j+1}) = 0$.

The graph in Figure 5.2 offers no apparent difficulties to Newton's method, because the function $f(\lambda)$ appears to be reasonably well approximated by

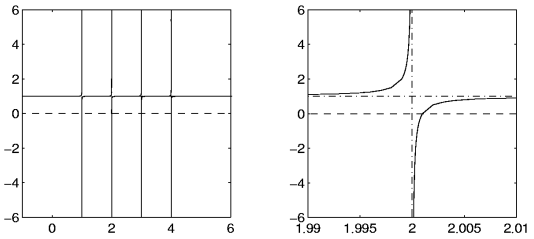

Fig. 5.3. *Graph of* $f(\lambda) = 1 + \frac{10^{-3}}{1-\lambda} + \frac{10^{-3}}{2-\lambda} + \frac{10^{-3}}{3-\lambda} + \frac{10^{-3}}{4-\lambda}$.

straight lines near each zero. But now consider the graph in Figure 5.3, which differs from Figure 5.2 only by changing u_i^2 from .5 to .001, which is not nearly small enough to deflate. The graph of $f(\lambda)$ in the left-hand figure is visually indistinguishable from its vertical and horizontal asymptotes, so in the right-hand figure we blow it up around one of the vertical asymptotes, $\lambda = 2$. We see that the graph of $f(\lambda)$ "turns the corner" very rapidly and is nearly horizontal for most values of λ. Thus, if we started Newton's method from almost any λ_0, the linear approximation $l(\lambda)$ would also be nearly horizontal with a slightly positive slope, so λ_1 would be an enormous negative number, a useless approximation to the true zero.

Newton's method can be modified to deal with this situation as follows. Since $f(\lambda)$ is not well approximated by a straight line $l(x)$, we approximate it by another simple function $h(x)$. There is nothing special about straight lines; any approximation $h(\lambda)$ that is both easy to compute and has zeros that are easy to compute can be used in place of $l(x)$ in Newton's method. Since $f(\lambda)$ has poles at d_i and d_{i+1} and these poles dominate the behavior of $f(\lambda)$ near them, it is natural when seeking the root in (d_i, d_{i+1}) to choose $h(\lambda)$ to have these poles as well, i.e.,

$$h(\lambda) = \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda} + c_3.$$

There are several ways to choose the constants c_1, c_2, and c_3 so that $h(\lambda)$ approximates $f(\lambda)$; we present a slightly simplified version of the one used in the LAPACK routine `slaed4` [170, 44]. Assuming for a moment that we have chosen c_1, c_2, and c_3, we can easily solve $h(\lambda) = 0$ for λ by solving the equivalent quadratic equation

$$c_1(d_{i+1} - \lambda) + c_2(d_i - \lambda) + c_3(d_i - \lambda)(d_{i+1} - \lambda) = 0.$$

Given the approximate zero λ_j, here is how we compute c_1, c_2, and c_3 so that for λ near λ_j

$$\frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda} + c_3 = h(\lambda) \approx f(\lambda) = 1 + \rho \sum_{k=1}^{n} \frac{u_k^2}{d_k - \lambda}.$$

Write

$$f(\lambda) = 1 + \sum_{k=1}^{i} \frac{u_k^2}{d_k - \lambda} + \sum_{k=i+1}^{n} \frac{u_k^2}{d_k - \lambda} \equiv 1 + \psi_1(\lambda) + \psi_2(\lambda).$$

For $\lambda \in (d_i, d_{i+1})$, $\psi_1(\lambda)$ is a sum of negative terms and $\psi_2(\lambda)$ is a sum of positive terms. Thus both $\psi_1(\lambda)$ and $\psi_2(\lambda)$ can be computed accurately, whereas adding them together would likely result in cancellation and loss of relative accuracy in the sum. We now choose c_1 and $\hat{c}_1$ so that

$$\begin{aligned} h_1(\lambda) &\equiv \hat{c}_1 + \frac{c_1}{d_i - \lambda} \quad \text{satisfies} \\ & \quad h_1(\lambda_j) = \psi_1(\lambda_j) \quad \text{and} \quad h_1'(\lambda_j) = \psi_1'(\lambda_j). \end{aligned} \tag{5.15}$$

This means that the graph of $h_1(\lambda)$ (a hyperbola) is tangent to the graph of $\psi_1(\lambda)$ at $\lambda = \lambda_j$. The two conditions in equation (5.15) are the usual conditions in Newton's method, except instead of using a straight line approximation, we use a hyperbola. It is easy to verify that $c_1 = \psi_1'(\lambda_j)(d_i - \lambda_j)^2$ and $\hat{c}_1 = \psi_1(\lambda_j) - \psi_1'(\lambda_j)(d_i - \lambda_j)$. (See Question 5.17.)

Similarly, we choose c_2 and $\hat{c}_2$ so that

$$\begin{aligned} h_2(\lambda) &\equiv \hat{c}_2 + \frac{c_2}{d_{i+1} - \lambda} \quad \text{satisfies} \\ & \quad h_2(\lambda_j) = \psi_2(\lambda_j) \quad \text{and} \quad h_2'(\lambda_j) = \psi_2'(\lambda_j). \end{aligned} \tag{5.16}$$

Finally, we set

$$\begin{aligned} h(\lambda) &= 1 + h_1(\lambda) + h_2(\lambda) \\ &= (1 + \hat{c}_1 + \hat{c}_2) + \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda} \\ &\equiv c_3 + \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda}. \end{aligned}$$

EXAMPLE 5.8. For example, in the example in Figure 5.3, if we start with $\lambda_0 = 2.5$, then

$$h(\lambda) = \frac{1.1111 \cdot 10^{-3}}{2 - \lambda} + \frac{1.1111 \cdot 10^{-3}}{3 - \lambda} + 1,$$

and its graph is visually indistinguishable from the graph of $f(\lambda)$ in the right-hand figure. Solving $h(\lambda_1) = 0$, we get $\lambda_1 = 2.0011$, which is accurate to 4 decimal digits. Continuing, λ_2 is accurate to 11 digits, and λ_3 is accurate to all 16 digits. $\diamond$

The algorithm used in LAPACK routine `slaed4` is a slight variation on the one described here (the one here is called *the Middle Way* in [170]). The LAPACK routine averages two to three iterations per eigenvalue to converge to full machine precision, and never took more than seven steps in extensive numerical tests.

Computing the Eigenvectors Stably

Once we have solved the secular equation to get the eigenvalues α_i of $D+\rho uu^T$, Lemma 5.2 provides a simple formula for the eigenvectors: $(D-\alpha_i I)^{-1}u$. Unfortunately, the formula can be unstable [58, 88, 232], in particular when two eigenvalues α_i and α_{i+1} are very close together. Intuitively, the problem is that $(D-\alpha_i I)^{-1}u$ and $(D-\alpha_{i+1}I)^{-1}n$ are "very close" formulas yet are supposed to yield orthogonal eigenvectors. More precisely, when α_i and α_{i+1} are very close, they must also be close to the d_i between them. Therefore, there is a great deal of cancellation, either when evaluating $d_i-\alpha_i$ and $d_i-\alpha_{i+1}$ or when evaluating the secular equation during Newton iteration. Either way, $d_i-\alpha_i$ and $d_i-\alpha_{i+1}$ may contain large relative errors, so the computed eigenvectors $(D-\alpha_i)^{-1}u$ and $(D-\alpha_{i+1})^{-1}u$ are quite inaccurate and far from orthogonal.

Early attempts to address this problem [88, 232] used double precision arithmetic (when the input data was single precision) to solve the secular equation to high accuracy so that $d_i-\alpha_i$ and $d_i-\alpha_{i+1}$ could be computed to high accuracy. But when the input data is already in double precision, this means quadruple precision would be needed, and this is not available in many machines and languages, or at least not cheaply. As described in section 1.5, it is possible to simulate quadruple precision using double precision [232, 202]. This can be done portably and relatively efficiently, as long as the underlying floating point arithmetic rounds sufficiently accurately. In particular, these simulations require that $\mathrm{fl}(a \pm b) = (a \pm b)(1+\delta)$ with $|\delta| = O(\varepsilon)$, barring overflow or underflow (see section 1.5 and Question 1.18). Unfortunately, the Cray 2, YMP, and C90 do not round accurately enough to use these efficient algorithms.

Finally, an alternative formula was found that makes simulating high precision arithmetic unnecessary. It is based on the following theorem of Löwner [127, 177].

THEOREM 5.10. Löwner. *Let $D = \mathrm{diag}(d_1, \ldots, d_n)$ be diagonal with $d_n < \cdots < d_1$. Let $\alpha_n < \cdots < \alpha_1$ be given, satisfying the* interlacing property

$$d_n < \alpha_n < \cdots < d_{i+1} < \alpha_{i+1} < d_i < \alpha_i < \cdots < d_1 < \alpha_1.$$

Then there is a vector $\hat{u}$ such that the α_i are the exact eigenvalues of $\hat{D} \equiv$

$D+\hat{u}\hat{u}^T$. The entries of $\hat{u}$ are given by

$$|\hat{u}_i| = \left[\frac{\prod_{j=1}^{n}(\alpha_j - d_i)}{\prod_{j=1,\, j\neq i}^{n}(d_j - d_i)} \right]^{1/2}.$$

Proof. The characteristic polynomial of $\hat{D}$ can be written both as $\det(\hat{D}-\lambda I) = \prod_{j=1}^{n}(\alpha_j - \lambda)$ and (using equations (5.13) and (5.14)) as

$$\begin{aligned}
\det(\hat{D}-\lambda I) &= \left[\prod_{j=1}^{n}(d_j-\lambda)\right] \cdot \left(1 + \sum_{j=1}^{n} \frac{\hat{u}_j^2}{d_j-\lambda}\right) \\
&= \left[\prod_{j=1}^{n}(d_j-\lambda)\right] \cdot \left(1 + \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{\hat{u}_j^2}{d_j-\lambda}\right) \\
&\quad + \left[\prod_{\substack{j=1 \\ j\neq i}}^{n} (d_j-\lambda)\right] \cdot \hat{u}_i^2.
\end{aligned}$$

Setting $\lambda = d_i$ and equating both expressions for $\det(\hat{D}-\lambda I)$ yield

$$\prod_{j=1}^{n}(\alpha_j - d_i) = \hat{u}_i^2 \cdot \prod_{\substack{j=1 \\ j\neq i}}^{n} (d_j - d_i)$$

or

$$\hat{u}_i^2 = \frac{\prod_{j=1}^{n}(\alpha_j - d_i)}{\prod_{j=1,\, j\neq i}^{n}(d_j - d_i)}.$$

Using the interlacing property, we can show that the fraction on the right is positive, so we can take its square root to get the desired expression for $\hat{u}_i$. □

Here is the stable algorithm for computing the eigenvalues and eigenvectors (where we assume for simplicity of presentation that $\rho = 1$).

ALGORITHM 5.3. *Compute the eigenvalues and eigenvectors of $D+uu^T$.*

Solve the secular equation $1 + \sum_{i=1}^{n} \frac{u_i^2}{d_i-\lambda} = 0$ to get the eigenvalues α_i of $D+uu^T$.
Use Löwner's theorem to compute $\hat{u}$ so that the α_i are "exact" eigenvalues of $D+\hat{u}\hat{u}^T$.
Use Lemma 5.2 to compute the eigenvectors of $D+\hat{u}\hat{u}^T$.