Numerical Linear Algebra
Lecture 13

Rayleigh Quotient Iteration (RQI)

ALGORITHM. Rayleigh quotient iteration (RQI): Given x_0 with $||x_0|| = 1$, and a user-supplied stopping tolerance tol, we iterate

$$
\begin{aligned}
&\rho_0 = \rho(x_0, A) = \frac{x_0^T A x_0}{x_0^T x_0} \\
&i = 1 \\
&\textit{repeat} \\
&\quad y_i = (A - \rho_{i-1} I)^{-1} x_{i-1} \\
&\quad x_i = y_i / ||y_i||_2 \\
&\quad \rho_i = \rho(x_i, A) \\
&\quad i = i + 1 \\
&\textit{until convergence } (||A x_i - \rho_i x_i||_2 < \mathrm{tol})
\end{aligned}
$$

When the stopping criterion is satisfied, Theorem tells us that ρ_i is within tol of an eigenvalue of A.

If one uses the shift $\sigma_i = a_{nn}$ in QR iteration and starts Rayleigh quotient iteration with $x_0 = [0, \ldots, 0, 1]^T$, then the connection between QR and inverse iteration can be used to show that the sequence of σ_i and ρ_i from the two algorithms are identical. In this case we will prove that convergence is almost always cubic.

THEOREM. *Rayleigh quotient iteration is locally cubically convergent; i.e., the number of correct digits triples at each step once the error is small enough and the eigenvalue is simple.*

Proof. We claim that it is enough to analyze the case when A is diagonal. To see why, write $Q^T A Q = \Lambda$, where Q is the orthogonal matrix whose columns are eigenvectors, and $\Lambda = \mathrm{diag}(\alpha_1, \ldots, \alpha_n)$ is the diagonal matrix of eigenvalues. Now change variables in Rayleigh quotient iteration to $\hat{x}_i \equiv Q^T x_i$ and $\hat{y}_i \equiv Q^T y_i$. Then

$$\rho_i = \rho(x_i, A) = \frac{x_i^T A x_i}{x_i^T x_i} = \frac{\hat{x}_i^T Q^T A Q \hat{x}_i}{\hat{x}_i^T Q^T Q \hat{x}_i} = \frac{\hat{x}_i^T \Lambda \hat{x}_i}{\hat{x}_i^T \hat{x}_i} = \rho(\hat{x}_i, \Lambda)$$

and from algorithm RQI it follows that $y_{i+1} = (A - \rho_i I)^{-1} x_i$ (since $Q\hat{y}_{i+1} = y_{i+1}$) and thus $Q\hat{y}_{i+1} = (A - \rho_i I)^{-1} Q\hat{x}_i$, so

$$\hat{y}_{i+1} = Q^T (A - \rho_i I)^{-1} Q \hat{x}_i = (Q^T A Q - \rho_i I)^{-1} \hat{x}_i = (\Lambda - \rho_i I)^{-1} \hat{x}_i.$$

Therefore, running Rayleigh quotient iteration with A and x_0 is equivalent to running Rayleigh quotient iteration with Λ and $\hat{x}_0$. Thus we will assume without loss of generality that $A = \Lambda$ is already diagonal, so the eigenvectors of A are e_i, the columns of the identity matrix.

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

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

so that

$$2\underbrace{e_1^T d_i}_{d_{i1}} + \epsilon^2 = 0 \tag{1}$$

or $2d_{i1} + \epsilon^2 = 0$, or $d_{i1} = -\epsilon^2/2$. Therefore

$$\begin{aligned} \rho_i = \frac{x_i^T \Lambda x_i}{x_i^T x_i} &= (e_1 + d_i)^T \Lambda (e_1 + d_i) = \underbrace{e_1^T \Lambda e_1}_{\alpha_1} + 2e_1^T \Lambda d_i + d_i^T \Lambda d_i \\ &= \alpha_1 - \underbrace{(-2e_1^T \Lambda d_i - d_i^T \Lambda d_i)}_{\eta} = \alpha_1 - \eta = \alpha_1 - \alpha_1 \epsilon^2 + d_i^T \Lambda d_i, \end{aligned} \tag{2}$$

and since by (1) we have $-2e_1^T d_i = \epsilon^2$ then
$\eta \equiv -2e_1^T \Lambda d_i - d_i^T \Lambda d_i = \alpha_1 \epsilon^2 - d_i^T \Lambda d_i$.

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

$$|\eta| \leq |\alpha_1|\epsilon^2 + ||\Lambda||_2 ||d_i||_2^2 \leq |\alpha_1|\epsilon^2 + ||\Lambda||_2 \epsilon^2 \leq 2||\Lambda||_2 \epsilon^2,$$

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

Now, from algorithm RQI we have $y_{i+1} = (A - \rho_i I)^{-1} x_i$, we can write

$$
\begin{aligned}
y_{i+1} &= (\Lambda - \rho_i I)^{-1} x_i \\
&= \left[\frac{x_{i1}}{\alpha_1 - \rho_i}, \frac{x_{i2}}{\alpha_2 - \rho_i}, \ldots, \frac{x_{in}}{\alpha_n - \rho_i} \right]^T \quad \textit{because } (\Lambda - \rho_i I)^{-1} = \operatorname{diag} \frac{1}{\alpha_j - \rho_i} \\
&= \left[\frac{1 + d_{i1}}{\alpha_1 - \rho_i}, \frac{d_{i2}}{\alpha_2 - \rho_i}, \ldots, \frac{d_{in}}{\alpha_n - \rho_i} \right]^T \quad \textit{because } x_i = e_1 + d_i \\
&= \left[\frac{1 - \epsilon^2/2}{\eta}, \frac{d_{i2}}{\alpha_2 - \alpha_1 + \eta}, \ldots, \frac{d_{in}}{\alpha_n - \alpha_1 + \eta} \right]^T \quad \textit{because } \rho_i = \alpha_1 - \eta \\
&= \frac{1 - \epsilon^2/2}{\eta} \cdot \left[1, \frac{d_{i2}\eta}{(1 - \epsilon^2/2)(\alpha_2 - \alpha_1 + \eta)}, \ldots, \right. \quad \textit{and } d_{i1} = -\epsilon^2/2 \\
&\qquad \left. \frac{d_{in}\eta}{(1 - \epsilon^2/2)(\alpha_n - \alpha_1 + \eta)} \right]^T \equiv \frac{1 - \epsilon^2/2}{\eta} \cdot (e_1 + \hat{d}_{i+1}).
\end{aligned}
\tag{3}
$$

To bound $||\hat{d}_{i+1}||_2$, we note that we can bound each denominator using $|\alpha_j - \alpha_1 + \eta| \geq \text{gap}(\alpha_1, \Lambda) - |\eta|$, so using also obtained bound

$$|\eta| \leq |\alpha_1|\epsilon^2 + ||\Lambda||_2||d_i||_2^2 \leq 2||\Lambda||_2\epsilon^2$$

we get

$$||\hat{d}_{i+1}||_2 \leq \frac{\|d_i\|_2|\eta|}{(1-\epsilon^2/2)(\text{gap}(\alpha_1,\Lambda) - |\eta|)} \leq \frac{\epsilon \cdot 2||\Lambda||_2\epsilon^2}{(1-\epsilon^2/2)(\text{gap}(\alpha_1,\Lambda) - 2||\Lambda||_2\epsilon^2)}$$

or $||\hat{d}_{i+1}||_2 = O(\epsilon^3)$. Finally, by algorithm how to compute Rayleigh quotient we have that $x_i = y_i/||y_i||_2$ and thus
$x_{i+1} = e_1 + d_{i+1} = y_{i+1}/||y_{i+1}||_2$ or

$$x_{i+1} = \frac{y_{i+1}}{||y_{i+1}||_2} = \frac{\left(\frac{1-\epsilon^2/2}{\eta} \cdot (e_1 + \hat{d}_{i+1})\right)}{\|\frac{1-\epsilon^2/2}{\eta} \cdot (e_1 + \hat{d}_{i+1})\|_2} = (e_1 + \hat{d}_{i+1})/||e_1 + \hat{d}_{i+1}||_2.$$

Since $x_{i+1} = e_1 + d_{i+1}$ we see $||d_{i+1}||_2 = O(\epsilon^3)$ as well. $\square$

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 [J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. Numer. Math., 36:177-195, 1981], the "right" implementation was not discovered until 1992 [M. Gu and S. Eisenstat. A stable algorithm for the rank-1 modification of the symmetric eigenproblem. Computer Science Dept. Report YALEU/DCS/RR-916, Yale University, September 1992;M. Gu and S. C. Eisenstat. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. SIAM J. Matrix Anal Appl, 16:172-191, 1995]).

$$T = \begin{bmatrix} a_1 & b_1 & 0 & \dots & \dots & 0 \\ b_1 & a_2 & b_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & a_{m-1} & b_{m-1} & \dots & \dots & 0 \\ 0 & b_{m-1} & a_m & b_m & 0 & 0 \\ 0 & 0 & b_m & a_{m+1} & b_{m+1} & 0 \\ 0 & 0 & 0 & b_{m+1} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & b_{n-1} \\ 0 & 0 & 0 & 0 & b_{n-1} & a_n \end{bmatrix}$$

$$T = \begin{bmatrix} a_1 & b_1 & 0 & \dots & \dots & 0 \\ b_1 & a_2 & b_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & a_{m-1} & b_{m-1} & \dots & \dots & 0 \\ 0 & b_{m-1} & a_m - b_m & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{m+1} - b_m & b_{m+1} & 0 \\ 0 & 0 & 0 & b_{m+1} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & b_{n-1} \\ 0 & 0 & 0 & 0 & b_{n-1} & a_n \end{bmatrix} + \begin{bmatrix} \dots & \dots \\ \dots b_m & b_m \dots \\ \dots b_m & b_m \dots \\ \dots & \dots \end{bmatrix}$$

$$= \begin{bmatrix} T_1 & 0 \dots 0 \\ 0 \dots 0 & T_2 \end{bmatrix} + b_m \cdot \begin{bmatrix} 0 \\ \dots \\ 0 \\ 1 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \dots 0 & 1 & 1 & 0 \dots 0 \end{bmatrix} = \begin{bmatrix} T_1 & 0 \dots 0 \\ 0 \dots 0 & T_2 \end{bmatrix} + b_m v \cdot v^T$$

Assume that we have eigendecomposition of T_1, T_2 such that $T_1 = Q_1 \Lambda_1 Q_1^T$ and $T_2 = Q_2 \Lambda_2 Q_2^T$. Then we can write that

$$T = \begin{bmatrix} T_1 & 0...0 \\ 0...0 & T_2 \end{bmatrix} + b_m v \cdot v^T = \begin{bmatrix} Q_1 \Lambda_1 Q_1^T & 0...0 \\ 0...0 & Q_2 \Lambda_2 Q_2^T \end{bmatrix} + b_m v \cdot v^T$$

$$= \begin{bmatrix} Q_1 & 0...0 \\ 0...0 & Q_2 \end{bmatrix} \cdot \left(\begin{bmatrix} \Lambda_1 & 0...0 \\ 0...0 & \Lambda_2 \end{bmatrix} + b_m u \cdot u^T \right) \cdot \begin{bmatrix} Q_1^T & 0...0 \\ 0...0 & Q_2^T \end{bmatrix}$$

Let define the diagonal matrix

$$D = \begin{bmatrix} \Lambda_1 & 0...0 \\ 0...0 & \Lambda_2 \end{bmatrix}.$$

We observe that the eigenvalues of T are the same as of $D + b_m u \cdot u^T = D + \rho u \cdot u^T$.

- 1. We want to find eigenvalues of $D + b_m u \cdot u^T = D + \rho u \cdot u^T$
- 2. Assumption: we assume that diagonal elements of D are sorted such that $d_1 \geq ... \geq d_n$ and $D - \lambda I$ is nonsingular
- 3. To find eigenvalues we compute the characteristic polynomial $D + \rho u \cdot u^T - \lambda I$ noting that
  $D + \rho u \cdot u^T - \lambda I = (D - \lambda I)(I + \rho (D - \lambda I)^{-1} u \cdot u^T)$
- By assumption we have $det(D - \lambda I) \neq 0$ and thus
  $det(I + \rho (D - \lambda I)^{-1} u \cdot u^T) = 0.$

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

Thus, using this lemma we can get that

$$det(I+\underbrace{\rho(D-\lambda I)^{-1}u}_{x}\cdot\underbrace{u^T}_{y^T}) = 1+\underbrace{u^T}_{y^T}\underbrace{\rho(D-\lambda I)^{-1}u}_{x} = 1+\rho\sum_{i=1}^{n}\frac{u_i^2}{d_i-\lambda} = f(\lambda)$$

.

We see that eigenvalues of T are roots of the secular equation $f(\lambda) = 0$. The secular equation can be solved using Newton's method with starting point in (d_i, d_{i+1}).

LEMMA. If α is an eigenvalue of $D + \rho uu^T$, then $(D - \alpha I)^{-1}u$ is its eigenvector. Since $D - \alpha I$ is diagonal, this costs $O(n)$ flops to compute.
PROOF.
We need to proove that$(D - \alpha I)^{-1}u$ is an eigenvector in the eigenvalue problem

$$(D + \rho uu^T)(D - \alpha I)^{-1}u = \alpha(D - \alpha I)^{-1}u. \tag{4}$$

Consider

$$\begin{aligned}(D + \rho uu^T)(D - \alpha I)^{-1}u &= (D - \alpha I + \alpha I + \rho uu^T)(D - \alpha I)^{-1}u \\ &= \frac{(D - \alpha I)}{D - \alpha I}u + \alpha(D - \alpha I)^{-1}u + \rho uu^T(D - \alpha I)^{-1}u \\ &= u + \alpha(D - \alpha I)^{-1}u - u.\end{aligned} \tag{5}$$

We have used that $\rho u^T(D - \alpha I)^{-1} + 1 = 0$ is the seqular equation.
Thus, $\rho u^T(D - \alpha I)^{-1} = -1$ and $\rho uu^T(D - \alpha I)^{-1}u = u \cdot (-1) = -u$.

ALGORITHM. 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* Λ_1, Λ_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

end if

Computing the Eigenvectors Stably

- Lemma below provides formula for the computing of eigenvectors.

  LEMMA. If α is an eigenvalue of $D + \rho uu^T$, then $(D - \alpha I)^{-1}u$ is its eigenvector.

- This formula is not stable in the case when two eigenvalues are close to each other. Let α_i, α_{i+1} are close to each other. Then $(D - \alpha_i)^{-1}u$ and $(D - \alpha_{i+1})^{-1}u$ are inaccurate and far from orthogonal.

- An alternative formula was found which is based on the Löwner's theorem.

THEOREM. (*Löwner*). Let $D = \mathrm{diag}(d_1, \ldots, d_n)$ be diagonal with $d_n < \ldots < d_1$. Let $\alpha_n < \ldots < \alpha_1$ 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 as $\det(\hat{D} - \lambda I) = \det(D + \hat{u}\hat{u}^T - \lambda I) = \det(D(I + D^{-1}\hat{u}\hat{u}^T) - \lambda I) = \det((D - \lambda I)(I + (D - \lambda I)^{-1}\hat{u}\hat{u}^T))$ or

$$\prod_{j=1}^{n}(\alpha_j - \lambda) = \det(\hat{D} - \lambda I) = \left[\prod_{j=1}^{n}(d_j - \lambda)\right] \cdot \left(1 + \sum_{j=1}^{n} \frac{\hat{u}_j^2}{d_j - \lambda}\right)$$

$$= \left[\prod_{j=1}^{n}(d_j - \lambda)\right] \cdot \left(1 + \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{\hat{u}_j^2}{d_j - \lambda}\right)$$

$$+ \underbrace{\left[\prod_{j=1}^{n}(d_j - \lambda)\right] \cdot \frac{\hat{u}_i^2}{d_i - \lambda}}_{(1)} \cdot$$

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

Note that (1) = (2).

Setting $\lambda = d_i$ and noting that $\prod_{j=1}^{n}(d_j - d_i) = 0$ for $i = j$ 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

$$\hat{u}_i^2 = \frac{\prod_{j=1}^{n}(\alpha_j - d_i)}{\prod_{j=1, j\neq i}^{n}(d_j - d_i)} > 0.$$

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 $\hat{u}_i$.

Stable divide-and-conquer algorithm

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

  α_i of $D + uu^T$ via Newton's method.

- Use Löwner's theorem to compute $\hat{u}$ so that the α_i are "exact"

  eigenvalues of $D + \hat{u}\hat{u}^T$.

- Use following Lemma to compute the eigenvectors of $\hat{D} = D + \hat{u}\hat{u}^T$ reformulated for $\hat{D} = D + \hat{u}\hat{u}^T$:

  Lemma:

  If α is an eigenvalue of $D + \rho\hat{u}\hat{u}^T$, then $(D - \alpha I)^{-1}\hat{u}$ is its eigenvector.

Example of the Matlab's program: eigenvalues will be on the diagonal of L, eigenvectors - columns of Q

```
  function [Q,L] = DivideandConq(T)
% Compute size of input matrix T:
[m,n] = size(T);
% here we will divide the matrix
m2 = floor(m/2);
%if m=0 we shall return
if m2 == 0 %1 by 1
  Q = 1; L = T;
  return;
  %else we perform recursive computations
else
  [T,T1,T2,bm,v] = formT(T,m2);
  %recursive computations
  [Q1,L1] = DivideandConq(T1);
  [Q2,L2] = DivideandConq(T2);
  %pick out the last and first columns of the transposes:
  Q1T = Q1';
  Q2T = Q2';
  u = [Q1T(:,end); Q2T(:,1)];
  %Creating the D-matrix:
  D = zeros(n);
  D(1:m2,1:m2) = L1;
  D((m2+1):end,(m2+1):end) = L2;
```

```
  % The Q matrix (with Q1 and Q2 on the "diagonals")
  Q = zeros(n);
  Q(1:m2,1:m2) = Q1;
  Q((m2+1):end,(m2+1):end) = Q2;

  %Creating the matrix B, which determinant is the secular equation:
  % det B = f(\lambda)=0
  B = D+bm*u*u';

  % Compute eigenvalues as roots of the secular equation
  %  f(\lambda)=0  using Newton's method
  eigs = NewtonMethod(D,bm,u);
  Q3 = zeros(m,n);

  % compute eigenvectors for corresponding eigenvalues
  for i = 1:length(eigs)
    Q3(:,i) = (D-eigs(i)*eye(m))\u;
    Q3(:,i) = Q3(:,i)/norm(Q3(:,i));
  end

  %Compute  eigenvectors of the original input matrix T
  Q = Q*Q3;

  % Present eigenvalues  of the original matrix input T
  %(they will be on diagonal)
  L = zeros(m,n);
  L(1:(m+1):end) = eigs;

  return;
end

end
```

Bisection and Inverse Iteration

- The Bisection algorithm exploits Sylvester's inertia theorem to find only those k eigenvalues that one wants, at cost $O(nk)$. Recall that $\mathrm{Inertia}(A) = (\nu, \zeta, \pi)$, where ν, ζ and π are the number of negative, zero, and positive eigenvalues of A, respectively. Suppose that X is nonsingular; Sylvester's inertia theorem asserts that $\mathrm{Inertia}(A) = \mathrm{Inertia}(X^T A X)$.
- Now suppose that one uses Gaussian elimination to factorize $A - zI = LDL^T$, where L is nonsingular and D diagonal. Then $\mathrm{Inertia}(A - zI) = \mathrm{Inertia}(D)$. Since D is diagonal, its inertia is trivial to compute. (In what follows, we use notation such as "$\#d_{ii} < 0$" to mean "the number of values of d_{ii} that are less than zero.")

We can write $\text{Inertia}(A - zI) = \text{Inertia}(D)$ as:

$$\begin{aligned}
\text{Inertia}(A - zI) &= (\#d_{ii} < 0, \#d_{ii} = 0, \#d_{ii} > 0) \\
&= (\#\text{ negative eigenvalues of } A - zI, \\
&\quad \#\text{ zero eigenvalues of } A - zI, \\
&\quad \#\text{ positive eigenvalues of } A - zI) \\
&= (\#\text{ eigenvalues of } A < z, \\
&\quad \#\text{ eigenvalues of } A = z, \\
&\quad \#\text{ eigenvalues of } A > z).
\end{aligned}$$

The number of eigenvalues in the interval $[z_1, z_2)$

- Suppose $z_1 < z_2$ and we compute $\mathrm{Inertia}(A - z_1 I)$ and $\mathrm{Inertia}(A - z_2 I)$.
- Then the number of eigenvalues in the interval $[z_1, z_2)$ equals (# eigenvalues of $A < z_2$) – (# eigenvalues of $A < z_1$).
- To make this observation into an algorithm, define

$$\mathrm{Negcount}(A, z) = \#\,\text{eigenvalues of } A < z.$$

Bisection algorithm

ALGORITHM. *Bisection: Find all eigenvalues of* A *inside* $[a, b)$ *to a given error tolerance* tol:
$n_a = Negcount(A, a)$
$n_b = Negcount(A, b)$
if $n_a = n_b$, *quit ... because there are no eigenvalues in* $[a, b)$
put $[a, n_a, b, n_b]$ *onto Worklist*
  /* *Worklist contains a list of intervals* $[a, b)$ *containing*
    eigenvalues $n - n_a$ *through* $n - n_b + 1$, *which the algorithm*
    will repeatedly bisect until they are narrower than tol. */
while Worklist is not empty do
  remove $[low, n_{low}, up, n_{up}]$ *from Worklist*
  if $(up - low < \mathrm{tol})$ *then*
    print "there are nup nlow eigenvalues in [low, up)"
  else
    $mid = (low + up)/2$
    $n_{mid} = Negcount(A, mid)$
    if $n_{mid} > n_{low}$ *then ... there are eigenvalues in* $[low, mid)$
      put $[low, n_{low}, mid, n_{mid}]$ *onto Worklist*
    end if
    if $n_{up} > n_{mid}$ *then ... there are eigenvalues in* $[mid, up)$
      put $[mid, n_{mid}, up, n_{up}]$ *onto Worklist*
    end if
  end if
end while

Since $\text{Inertia}(A - zI) = \text{Inertia}(D)$ then it is easier to compute $\text{Inertia}(A - zI)$ from $\text{Inertia}(D)$. Let us write

$$A - zI = \begin{bmatrix} a_1 - z & b_1 & \dots & \dots \\ b_1 & a_2 - z & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & b_{n-2} & a_{n-1} - z & b_{n-1} \\ \dots & \dots & b_{n-1} & a_n - z \end{bmatrix} = LDL^T$$

$$= \begin{bmatrix} 1 & \dots & \dots \\ l_1 & 1 & \dots \\ \dots & \dots & \dots \\ \dots & l_{n-1} & 1 \end{bmatrix} \cdot \begin{bmatrix} d_1 & \dots & \dots \\ \dots & d_2 & \dots \\ \dots & \dots & \dots \\ \dots & .. & d_n \end{bmatrix} \cdot \begin{bmatrix} 1 & l_1 \dots & \dots \\ .. & 1 & \dots \\ \dots & \dots & l_{n-1} \\ \dots & \dots & 1 \end{bmatrix}$$

and

$$a_1 - z = d_1, \tag{6}$$

$$d_1 l_1 = b_1, \tag{7}$$

$$l_{i-1}^2 d_{i-1} + d_i = a_i - z, \tag{8}$$

$$d_i l_i = b_i. \tag{9}$$

Since $d_i l_i = b_i$ then we can compute $l_{i-1}^2 = b_{i-1}^2 / d_{i-1}^2$ and substitute it into $l_{i-1}^2 d_{i-1} + d_i = a_i - z$ to get:

$$d_i = (a_i - z) - \frac{b_{i-1}^2}{d_{i-1}},$$

This is the stable procedure since $A - zI$ is tridiagonal matrix.

Implementation of Negcount(A,z) in Matlab: it is enough to compute number of negative eigenvalues, for example

```
function [ neg ] = Negcount( A,z )
  d=zeros(length(A),1);
  d(1)=A(1,1)-z;
  for i = 2:length(A)
    d(i)=(A(i,i)-z)-(A(i,i-1)^2)/d(i-1);
  end
  %compute number of negative eigenvalues of A
  neg=0;
  for i = 1:length(A)
    if d(i)<0
      neg = neg+1;
    end
  end
end
```

Jacobi's Method

Given a symmetric matrix $A = A_0$, Jacobi's method produces a sequence A_1, A_2, ... of orthogonally similar matrices, which eventually converge to a diagonal matrix with the eigenvalues on the diagonal. A_{i+1} is obtained from A_i by the formula $A_{i+1} = J_i^T A_i J_i$, where J_i is an orthogonal matrix called a *Jacobi rotation*. Thus

$$\begin{aligned} A_m &= J_{m-1}^T A_{m-1} J_{m-1} \\ &= J_{m-1}^T J_{m-2}^T A_{m-2} J_{m-2} J_{m-1} = \cdots \\ &= J_{m-1}^T \cdots J_0^T A_0 J_0 \cdots J_{m-1} \\ &= J^T A J. \end{aligned}$$

If we choose each J_i appropriately, A_m approaches a diagonal matrix Λ for large m. Thus we can write $\Lambda \approx J^T A J$ or $J \Lambda J^T \approx A$. Therefore, the columns of J are approximate eigenvectors.
We will make $J^T A J$ nearly diagonal by iteratively choosing J_i to make *one* pair of offdiagonal entries of $A_{i+1} = J_i^T A_i J_i$ zero at a time. We will do this by choosing J_i to be a Givens rotation,

$$J_i = R(j,k,\theta) \equiv \begin{array}{c} \\ j \\ \\ k \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & & j & & k & & & \end{array} \\ \begin{bmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & \ddots & & & & & & \\ & & & \cos\theta & & -\sin\theta & & & \\ & & & & \ddots & & & & \\ & & & \sin\theta & & \cos\theta & & & \\ & & & & & & \ddots & & \\ & & & & & & & 1 & \\ & & & & & & & & 1 \end{bmatrix} \end{array},$$

where θ is chosen to zero out the j,k and k,j entries of A_{i+1}. To determine θ (or actually $\cos\theta$ and $\sin\theta$),

write

$$\begin{bmatrix} a_{jj}^{(i+1)} & a_{jk}^{(i+1)} \\ a_{kj}^{(i+1)} & a_{kk}^{(i+1)} \end{bmatrix} = \begin{bmatrix} \underbrace{\cos\theta}_{c} & -\sin\theta \\ \underbrace{\sin\theta}_{s} & \cos\theta \end{bmatrix}^T \begin{bmatrix} a_{jj}^{(i)} & a_{jk}^{(i)} \\ a_{kj}^{(i)} & a_{kk}^{(i)} \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix},$$

where λ_1 and λ_2 are the eigenvalues of

$$\begin{bmatrix} a_{jj}^{(i)} & a_{jk}^{(i)} \\ a_{kj}^{(i)} & a_{kk}^{(i)} \end{bmatrix}.$$

It is easy to compute $\cos\theta$ and $\sin\theta$: Multiplying out the last expression, using symmetry, abbreviating $c \equiv \cos\theta$ and $s \equiv \sin\theta$, and dropping the superscript (i) for simplicity yield

$$\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} a_{jj}c^2 + a_{kk}s^2 + 2sca_{jk} & \underbrace{sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2)}_{=0} \\ \underbrace{sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2)}_{=0} & a_{jj}s^2 + a_{kk}c^2 - 2sca_{jk} \end{bmatrix}.$$

Setting the offdiagonals to 0 and solving for θ we get $0 = sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2)$, which can be rewritten as

$$\frac{a_{jj} - a_{kk}}{2a_{jk}} = \frac{c^2 - s^2}{2sc} = \frac{\cos 2\theta}{\sin 2\theta} = \cot 2\theta \equiv \tau.$$

We now let $t = \frac{s}{c} = \tan\theta$ and note that

$$c^2 - s^2 = 2sc\tau, \tag{10}$$

$$c^2 - s^2 - 2sc\tau = 0. \tag{11}$$

Dividing both sides by c^2 and noting that $t = \frac{s}{c}$ we get:

$$1 - t^2 = 2t\tau, \tag{12}$$

$$-(t^2 + 2\tau t - 1) = 0. \tag{13}$$

Thus, we solve $t^2 + 2\tau t - 1 = 0$ to get (via the quadratic formula) $t = \frac{\text{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}}$. Since $cos^2\theta + \sin^2\theta = s^2 + c^2 = 1$ then $c^2 = 1 - s^2 = 1 - t^2c^2 \rightarrow c^2 - 1 + t^2c^2 = 0 \rightarrow c^2(1 + t^2) - 1 = 0 \rightarrow c = \frac{1}{\sqrt{1+t^2}}$ and $s = t \cdot c$. We summarize this derivation in the following algorithm.

ALGORITHM. *Compute and apply a Jacobi rotation to A in coordinates j, k:*

proc Jacobi-Rotation (A, j, k)

if $|a_{jk}|$ *is not too small*

$\tau = (a_{jj} - a_{kk})/(2 \cdot a_{jk})$

$t = \mathrm{sign}(\tau)/(|\tau| + \sqrt{1 + \tau^2})$

$c = 1/\sqrt{1 + t^2}$

$s = t \cdot c$

$A = R^T(j, k, \theta) \cdot A \cdot R(j, k, \theta)$ *... where* $c = \cos\theta$ *and* $s = \sin\theta$

if eigenvectors are desired

$J = J \cdot R(j, k, \theta)$

end if

end if