11.1 Jacobi Transformations of a Symmetric Matrix

The Jacobi method consists of a sequence of orthogonal similarity transformations of the form of equation (11.0.14). Each transformation (a Jacobi rotation) is just a plane rotation designed to annihilate one of the off-diagonal matrix elements. Successive transformations undo previously set zeros, but the off-diagonal elements nevertheless get smaller and smaller, until the matrix is diagonal to machine precision. Accumulating the product of the transformations as you go gives the matrix of eigenvectors, equation (11.0.15), while the elements of the final diagonal matrix are the eigenvalues.

The Jacobi method is absolutely foolproof for all real symmetric matrices. For matrices of order greater than about 10, say, the algorithm is slower, by a significant constant factor, than the \( QR \) method we shall give in §11.3. However, the Jacobi algorithm is much simpler than the more efficient methods. We thus recommend it for matrices of moderate order, where expense is not a major consideration.

The basic Jacobi rotation \( P_{pq} \) is a matrix of the form

\[
P_{pq} = \begin{bmatrix}
1 & \cdots & c & \cdots & s \\
\vdots & 1 & \vdots \\
-s & \cdots & c & \cdots & 1
\end{bmatrix}
\]  

(11.1.1)

Here all the diagonal elements are unity except for the two elements \( c \) in rows (and columns) \( p \) and \( q \). All off-diagonal elements are zero except the two elements \( s \) and \( -s \). The numbers \( c \) and \( s \) are the cosine and sine of a rotation angle \( \phi \), so \( c^2 + s^2 = 1 \).
A plane rotation such as (11.1.1) is used to transform the matrix $A$ according to

$$A' = P_{pq}^T \cdot A \cdot P_{pq} \quad (11.1.2)$$

Now, $P_{pq}^T \cdot A$ changes only rows $p$ and $q$ of $A$, while $A \cdot P_{pq}$ changes only columns $p$ and $q$. Notice that the subscripts $p$ and $q$ do not denote components of $P_{pq}$, but rather label which kind of rotation the matrix is, i.e., which rows and columns it affects. Thus the changed elements of $A$ in (11.1.2) are only in the $p$ and $q$ rows and columns indicated below:

$$A' = \begin{bmatrix}
\cdots & a'_{1p} & \cdots & a'_{1q} & \cdots \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
a'_{p1} & \cdots & a'_{pp} & \cdots & a'_{pn} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
a'_{q1} & \cdots & a'_{qp} & \cdots & a'_{qq} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\cdots & a'_{np} & \cdots & a'_{nq} & \cdots 
\end{bmatrix} \quad (11.1.3)$$

Multiplying out equation (11.1.2) and using the symmetry of $A$, we get the explicit formulas

$$a'_{rp} = ca_{rp} - sa_{rq} \quad r \neq p, \ r \neq q \quad (11.1.4)$$

$$a'_{r q} = ca_{rq} + sa_{rp}$$

$$a'_{pp} = c^2a_{pp} + s^2a_{qq} - 2sc_{pq} \quad (11.1.5)$$

$$a'_{qq} = s^2a_{pp} + c^2a_{qq} + 2sc_{pq} \quad (11.1.6)$$

$$a'_{pq} = (c^2 - s^2)a_{pq} + sc(a_{pp} - a_{qq}) \quad (11.1.7)$$

The idea of the Jacobi method is to try to zero the off-diagonal elements by a series of plane rotations. Accordingly, to set $a'_{pq} = 0$, equation (11.1.7) gives the following expression for the rotation angle $\phi$

$$\theta \equiv \cot 2\phi \equiv \frac{c^2 - s^2}{2sc} = \frac{a_{qq} - a_{pp}}{2a_{pq}} \quad (11.1.8)$$

If we let $t \equiv s/c$, the definition of $\theta$ can be rewritten

$$t^2 + 2t\theta - 1 = 0 \quad (11.1.9)$$

The smaller root of this equation corresponds to a rotation angle less than $\pi/4$ in magnitude; this choice at each stage gives the most stable reduction. Using the form of the quadratic formula with the discriminant in the denominator, we can write this smaller root as

$$t = \frac{\text{sgn}(\theta)}{|\theta| + \sqrt{\theta^2 + 1}} \quad (11.1.10)$$
If $\theta$ is so large that $\theta^2$ would overflow on the computer, we set $t = 1/(2\theta)$. It now follows that

$$c = \frac{1}{\sqrt{t^2 + 1}} \quad (11.1.11)$$

$$s = tc \quad (11.1.12)$$

When we actually use equations (11.1.4)–(11.1.7) numerically, we rewrite them to minimize roundoff error. Equation (11.1.7) is replaced by

$$a_{pq}' = 0 \quad (11.1.13)$$

The idea in the remaining equations is to set the new quantity equal to the old quantity plus a small correction. Thus we can use (11.1.7) and (11.1.13) to eliminate $a_{qq}$ from (11.1.5), giving

$$a_{pp}' = a_{pp} - ta_{pq} \quad (11.1.14)$$

Similarly,

$$a_{qq}' = a_{qq} + ta_{pq} \quad (11.1.15)$$

$$a_{rp}' = a_{rp} - s(a_{rq} + \tau a_{rp}) \quad (11.1.16)$$

$$a_{rq}' = a_{rq} + s(a_{rp} - \tau a_{rq}) \quad (11.1.17)$$

where $\tau (= \tan \phi/2)$ is defined by

$$\tau \equiv \frac{s}{1 + c} \quad (11.1.18)$$

One can see the convergence of the Jacobi method by considering the sum of the squares of the off-diagonal elements

$$S = \sum_{r \neq s} |a_{rs}|^2 \quad (11.1.19)$$

Equations (11.1.4)–(11.1.7) imply that

$$S' = S - 2|a_{pq}|^2 \quad (11.1.20)$$

(Since the transformation is orthogonal, the sum of the squares of the diagonal elements increases correspondingly by $2|a_{pq}|^2$.) The sequence of $S$'s thus decreases monotonically. Since the sequence is bounded below by zero, and since we can choose $a_{pq}$ to be whatever element we want, the sequence can be made to converge to zero.

Eventually one obtains a matrix $D$ that is diagonal to machine precision. The diagonal elements give the eigenvalues of the original matrix $A$, since

$$D = V^T \cdot A \cdot V \quad (11.1.21)$$
where

\[ V = P_1 \cdot P_2 \cdot P_3 \cdot \ldots \]  \hspace{1cm} (11.1.22)

the \( P_i \)'s being the successive Jacobi rotation matrices. The columns of \( V \) are the eigenvectors (since \( A \cdot V = V \cdot D \)). They can be computed by applying

\[ V' = V \cdot P_i \]  \hspace{1cm} (11.1.23)

at each stage of calculation, where initially \( V \) is the identity matrix. In detail, equation (11.1.23) is

\[ v'_{rs} = v_{rs} \quad (s \neq p, s \neq q) \]
\[ v'_{rp} = c v_{rp} - s v_{rq} \]
\[ v'_{rq} = s v_{rp} + c v_{rq} \]  \hspace{1cm} (11.1.24)

We rewrite these equations in terms of \( \tau \) as in equations (11.1.16) and (11.1.17) to minimize roundoff.

The only remaining question is the strategy one should adopt for the order in which the elements are to be annihilated. Jacobi's original algorithm of 1846 searched the whole upper triangle at each stage and set the largest off-diagonal element to zero. This is a reasonable strategy for hand calculation, but it is prohibitive on a computer since the search alone makes each Jacobi rotation a process of order \( N^2 \) instead of \( N \).

A better strategy for our purposes is the cyclic Jacobi method, where one annihilates elements in strict order. For example, one can simply proceed down the rows: \( P_{12}, P_{13}, \ldots, P_{1n}; \) then \( P_{23}, P_{24}, \) etc. One can show that convergence is generally quadratic for both the original or the cyclic Jacobi methods, for nondegenerate eigenvalues. One such set of \( n(n-1)/2 \) Jacobi rotations is called a sweep.

The program below, based on the implementations in [1,2], uses two further refinements:

- In the first three sweeps, we carry out the \( pq \) rotation only if \( |a_{pq}| > \epsilon \) for some threshold value

\[ \epsilon = \frac{1}{5} \frac{S_0}{n^2} \]  \hspace{1cm} (11.1.25)

where \( S_0 \) is the sum of the off-diagonal moduli,

\[ S_0 = \sum_{r<s} |a_{rs}| \]  \hspace{1cm} (11.1.26)

- After four sweeps, if \( |a_{pq}| \ll |a_{pp}| \) and \( |a_{pq}| \ll |a_{qq}| \), we set \( |a_{pq}| = 0 \) and skip the rotation. The criterion used in the comparison is \( |a_{pq}| < 10^{-D+2} |a_{pp}| \) where \( D \) is the number of significant decimal digits on the machine, and similarly for \( |a_{qq}| \).
In the following routine the $n \times n$ symmetric matrix $a(1:n,1:n)$ is stored in an $np \times np$ array. On output, the superdiagonal elements of $a$ are destroyed, but the diagonal and subdiagonal are unchanged and give full information on the original symmetric matrix $a$. The parameter $d$ is a vector of length $np$. On output, it returns the eigenvalues of $a$ in its first $n$ elements. During the computation, it contains the current diagonal of $a$. The matrix $v$ outputs the normalized eigenvector belonging to $d(k)$ in its $k$th column. The parameter $nrot$ is the number of Jacobi rotations that were needed to achieve convergence.

Typical matrices require 6 to 10 sweeps to achieve convergence, or $3n^2$ to $5n^2$ Jacobi rotations. Each rotation requires of order $4n$ operations, each consisting of a multiply and an add, so the total labor is of order $12n^3$ to $20n^3$ operations. Calculation of the eigenvectors as well as the eigenvalues changes the operation count from $4n^3$ to $6n^3$ per rotation, which is only a 50 percent overhead.

SUBROUTINE jacobi(a,n,np,d,v,nrot)
INTEGER n,np,nrot,NMAX
REAL a(np,np),d(np),v(np,np)
PARAMETER (NMAX=500)
Computes all eigenvalues and eigenvectors of a real symmetric matrix $a$, which is of size $n$ by $n$, stored in a physical $np$ by $np$ array. On output, elements of $a$ above the diagonal are destroyed. $d$ returns the eigenvalues of $a$ in its first $n$ elements. $v$ is a matrix with the same logical and physical dimensions as $a$, whose columns contain, on output, the normalized eigenvectors of $a$. $nrot$ returns the number of Jacobi rotations that were required.

INTEGER i,ip,iq,j
REAL c,g,h,s,sm,t,tau,theta,tresh,b(NMAX),z(NMAX)
do 12 ip=1,n  
Initialize to the identity matrix.
do 11 iq=1,n  
v(ip,iq)=0.  
enddo 11  
v(ip,ip)=1.  
enddo 12  
do 13 ip=1,n  
b(ip)=a(ip,ip)  
Initialize $b$ and $d$ to the diagonal of $a$.
d(ip)=b(ip)  
z(ip)=0.  
This vector will accumulate terms of the form $ta_{pq}$ as in equation (11.1.14).  
enddo 13  
nrot=0  
do 24 i=1,50  
sm=0.  
Sum off-diagonal elements.
do 21 iq=ip+1,n  
sm=sm+abs(a(ip,iq))  
enddo 21  
if(sm.eq.0.)return  
The normal return, which relies on quadratic convergence to machine underflow.
else  
tresh=0.2*sm/n**2  
...on the first three sweeps.
tresh=0.  
...thereafter.
endif  
do 15 ip=1,n-1  
g=100.*abs(a(ip,iq))  
After four sweeps, skip the rotation if the off-diagonal element is small.
if((i.gt.4).and.(abs(d(ip))+g.eq.abs(d(ip)))
   .and.(abs(d(iq))+g.eq.abs(d(iq))))then  
a(ip,iq)=0.  
else if(abs(a(ip,iq)).gt.tresh)then  
$h=d(iq)-d(ip)$
if(abs(h)+g.eq.abs(h))then

11.1 Jacobi Transformations of a Symmetric Matrix

\[ t = a_{ip, iq}/h \quad t = 1/(2\theta) \]

else
\[ \theta = 0.5h/a_{ip, iq} \quad \text{Equation (11.1.10).} \]
\[ t = 1/\left(\text{abs}(\theta) + \sqrt{1 + \theta^2}\right) \]
if(\theta < 0.) t = -t
endif
\[ c = 1/\sqrt{1 + t^2} \]
\[ s = t*c \]
\[ \tau = s/(1 + c) \]
\[ h = t*a_{ip, iq} \]
\[ z_{ip} = z_{ip} - h \]
\[ z_{iq} = z_{iq} + h \]
\[ d_{ip} = d_{ip} - h \]
\[ d_{iq} = d_{iq} + h \]
a_{ip, iq} = 0.
do 16 j = 1, ip - 1, Case of rotations 1 \leq j < p.
\[ g = a_{j, ip} \]
\[ h = a_{j, iq} \]
\[ a_{j, ip} = g - s(h + g*\tau) \]
\[ a_{j, iq} = h + s(g - h*\tau) \]
enddo 16

do 17 j = ip + 1, iq - 1, Case of rotations p < j < q.
\[ g = a_{ip, j} \]
\[ h = a_{j, iq} \]
\[ a_{ip, j} = g - s(h + g*\tau) \]
\[ a_{j, iq} = h + s(g - h*\tau) \]
enddo 17

do 18 j = iq + 1, n, Case of rotations q < j \leq n.
\[ g = a_{ip, j} \]
\[ h = a_{iq, j} \]
\[ a_{ip, j} = g - s(h + g*\tau) \]
\[ a_{iq, j} = h + s(g - h*\tau) \]
enddo 18

do 19 j = 1, n
\[ g = v_{j, ip} \]
\[ h = v_{j, iq} \]
\[ v_{j, ip} = g - s(h + g*\tau) \]
\[ v_{j, iq} = h + s(g - h*\tau) \]
enddo 19
\[ \text{nrot} = \text{nrot} + 1 \]
endif
enddo 22
do 23 ip = 1, n
\[ b_{ip} = b_{ip} + z_{ip} \]
\[ d_{ip} = b_{ip} \]
\[ z_{ip} = 0. \quad \text{Update } d \text{ with the sum of } t_{ipq}, \]
\[ \text{and reinitialize } z. \]
enddo 23
pause 'too many iterations in jacobi'
return
END

Note that the above routine assumes that underflows are set to zero. On machines where this is not true, the program must be modified.

The eigenvalues are not ordered on output. If sorting is desired, the following routine can be invoked to reorder the output of jacobi or of later routines in this chapter. (The method, straight insertion, is \(N^2\) rather than \(N \log N\); but since you have just done an \(N^3\) procedure to get the eigenvalues, you can afford yourself this little indulgence.)
SUBROUTINE eigsrt(d,v,n,np)
  INTEGER n,np
  REAL d(np),v(np,np)
Given the eigenvalues d and eigenvectors v as output from jacobi (§11.1) or tqli (§11.3),
this routine sorts the eigenvalues into descending order, and rearranges the columns of v
correspondingly. The method is straight insertion.
  INTEGER i,j,k
  REAL p
  do 13 i=1,n-1
    k=i
    p=d(i)
    do 11 j=i+1,n
      if(d(j).ge.p)then
        k=j
        p=d(j)
      endif
    enddo
  11 if(k.ne.i)then
    d(k)=d(i)
    d(i)=p
    do 12 j=1,n
      p=v(j,i)
      v(j,i)=v(j,k)
      v(j,k)=p
    enddo
  endif
  enddo
13 return
END

CITED REFERENCES AND FURTHER READING:
Golub, G.H., and Van Loan, C.F. 1989, Matrix Computations, 2nd ed. (Baltimore: Johns Hopkins
University Press), §8.4.

11.2 Reduction of a Symmetric Matrix
to Tridiagonal Form: Givens and
Householder Reductions

As already mentioned, the optimum strategy for finding eigenvalues and
eigenvectors is, first, to reduce the matrix to a simple form, only then beginning an
iterative procedure. For symmetric matrices, the preferred simple form is tridiagonal.
The Givens reduction is a modification of the Jacobi method. Instead of trying to
reduce the matrix all the way to diagonal form, we are content to stop when the
matrix is tridiagonal. This allows the procedure to be carried out in a finite number
of steps, unlike the Jacobi method, which requires iteration to convergence.