2.10 QR Decomposition

There is another matrix factorization that is sometimes very useful, the so-called QR decomposition,

\[ A = Q \cdot R \]  \hspace{1cm} (2.10.1)

Here \( R \) is upper triangular, while \( Q \) is orthogonal, that is,

\[ Q^T \cdot Q = I \]  \hspace{1cm} (2.10.2)

where \( Q^T \) is the transpose matrix of \( Q \). Although the decomposition exists for a general rectangular matrix, we shall restrict our treatment to the case when all the matrices are square, with dimensions \( N \times N \).
Like the other matrix factorizations we have met (LU, SVD, Cholesky), QR decomposition can be used to solve systems of linear equations. To solve
\[ A \cdot x = b \] (2.10.3)
first form \( Q^T \cdot b \) and then solve
\[ R \cdot x = Q^T \cdot b \] (2.10.4)
by backsubstitution. Since QR decomposition involves about twice as many operations as LU decomposition, it is not used for typical systems of linear equations. However, we will meet special cases where QR is the method of choice.

The standard algorithm for the QR decomposition involves successive Householder transformations (to be discussed later in §11.2). We write a Householder matrix in the form \( I - u \otimes u / c \) where \( c = \frac{1}{2} u \cdot u \). An appropriate Householder matrix applied to a given matrix can zero all elements in a column of the matrix situated below a chosen element. Thus we arrange for the first Householder matrix \( Q_1 \) to zero all elements in the first column of \( A \) below the first element. Similarly \( Q_2 \) zeroes all elements in the second column below the second element, and so on up to \( Q_{n-1} \). Thus
\[ R = Q_{n-1} \cdots Q_1 \cdot A \] (2.10.5)

Since the Householder matrices are orthogonal,
\[ Q = (Q_{n-1} \cdots Q_1)^{-1} = Q_1 \cdots Q_{n-1} \] (2.10.6)

In most applications we don’t need to form \( Q \) explicitly; we instead store it in the factored form (2.10.6). Pivoting is not usually necessary unless the matrix \( A \) is very close to singular. A general QR algorithm for rectangular matrices including pivoting is given in [1]. For square matrices, an implementation is the following:

SUBROUTINE qrdcmp(a,n,np,c,d,sing)
INTEGER n,np
REAL a(np,np),c(n),d(n)
LOGICAL sing

Constructs the QR decomposition of a(1:n,1:n), with physical dimension np. The upper triangular matrix \( R \) is returned in the upper triangle of a, except for the diagonal elements of \( R \) which are returned in d(1:n). The orthogonal matrix \( Q \) is represented as a product of \( n - 1 \) Householder matrices \( Q_1 \cdots Q_{n-1} \), where \( Q_1 = I - u_1 \otimes u_1 / c_1 \). The 1th component of \( u_1 \) is zero for \( i = 1, \ldots, j - 1 \) while the nonzero components are returned in \( a(i,j) \) for \( i = j, \ldots, n \). sing returns as true if singularity is encountered during the decomposition, but the decomposition is still completed in this case.

INTEGER i,j,k
REAL scale,sigma,sum,tau
LOGICAL sing=.false.,sum=.false.
do 17 k=1,n-1
scale=0.
do 11 i=k,n
scale=max(scale,abs(a(i,k)))
enddo
if(scale.eq.0.)then
Singular case.
sing=.true.,
c(k)=0.
d(k)=0.
else
Form \( Q_k \) and \( Q_k \cdot A \).
do 12 i=k,n
a(i,k)=a(i,k)/scale
enddo
sum=0.
do 13 i=k,n
sum=sum+a(i,k)**2
enddo
sigma=sign(sqrt(sum),a(k,k))
a(k,k)=a(k,k)+sigma

10 continue

11 continue

12 continue

13 continue

17 continue

END
c(k)=sigma*a(k,k)
d(k)=-scale*sigma
do 16 j=k+1,n
    sum=0.
do 14 i=k,n
    sum=sum+a(i,k)*a(i,j)
  enddo
  tau=sum/c(k)
do 15 i=k,n
    a(i,j)=a(i,j)-tau*a(i,k)
  enddo
enddo
endif
enddo
17
d(n)=a(n,n)
if(d(n).eq.0.)sing=.true.
return
END

The next routine, qrsolv, is used to solve linear systems. In many applications only the part (2.10.4) of the algorithm is needed, so we separate it off into its own routine rsolv.

SUBROUTINE qrsolv(a,n,np,c,d,b)
INTEGER n,np
REAL a(np,np),b(n),c(n),d(n)
C USES rsolv
Solves the set of n linear equations A·x = b, where A is a matrix with physical dimension np. a, c, and d are input as the output of the routine qrdcmp and are not modified. b(1:n) is input as the right-hand side vector, and is overwritten with the solution vector on output.
INTEGER i,j
REAL sum,tau
   do 13 j=1,n-1
      Form Q^T·b.
      sum=0.
      do 11 i=j,n
         sum=sum+a(i,j)*b(i)
      enddo
      tau=sum/c(j)
      do 12 i=j,n
         b(i)=b(i)-tau*a(i,j)
      enddo
   enddo
   call rsolv(a,n,np,d,b)     Solve R·x = Q^T·b.
return
END

SUBROUTINE rsolv(a,n,np,d,b)
INTEGER n,np
REAL a(np,np),b(n),d(n)
Solves the set of n linear equations R·x = b, where R is an upper triangular matrix stored in a and d. a and d are input as the output of the routine qrdcmp and are not modified. b(1:n) is input as the right-hand side vector, and is overwritten with the solution vector on output.
INTEGER i,j
REAL sum
   b(n)=b(n)/d(n)
   do 12 i=n-1,1,-1
      sum=0.
      do 11 j=i+1,n
         sum=sum+a(i,j)*b(j)
      enddo
      b(i)=(b(i)-sum)/d(i)
### Updating a QR decomposition

Some numerical algorithms involve solving a succession of linear systems each of which differs only slightly from its predecessor. Instead of doing \( O(N^3) \) operations each time to solve the equations from scratch, one can often update a matrix factorization in \( O(N^2) \) operations and use the new factorization to solve the next set of linear equations. The \( LU \) decomposition is complicated to update because of pivoting. However, \( QR \) turns out to be quite simple for a very common kind of update,

\[
A \rightarrow A + s \otimes t \quad (2.10.7)
\]

(compare equation 2.7.1). In practice it is more convenient to work with the equivalent form

\[
A = QR \rightarrow A' = Q' \cdot R' = Q \cdot (R + u \otimes v) \quad (2.10.8)
\]

One can go back and forth between equations (2.10.7) and (2.10.8) using the fact that \( Q \) is orthogonal, giving

\[
t = v \quad \text{and either} \quad s = Q \cdot u \quad \text{or} \quad u = Q^T \cdot s \quad (2.10.9)
\]

The algorithm [2] has two phases. In the first we apply \( N - 1 \) Jacobi rotations (§11.1) to reduce \( R + u \otimes v \) to upper Hessenberg form. Another \( N - 1 \) Jacobi rotations transform this upper Hessenberg matrix to the new upper triangular matrix \( R' \). The matrix \( Q' \) is simply the product of \( Q \) with the \( 2(N - 1) \) Jacobi rotations. In applications we usually want \( Q^T \), and the algorithm can easily be rearranged to work with this matrix instead of with \( Q \).

**SUBROUTINE qrupdt(r,qt,n,np,u,v)**

INTEGER n,np
REAL r(np,np),qt(np,np),u(np),v(np)

C USES rotate

Given the QR decomposition of some \( n \times n \) matrix, calculates the QR decomposition of the matrix \( Q \cdot (R + u \otimes v) \). The matrices \( r \) and \( qt \) have physical dimension \( np \). Note that \( Q^T \) is input and returned in \( qt \).

**INTEGER i,j,k**

**DO 11 k=n,1,-1**

Find largest \( k \) such that \( u(k) \neq 0 \).

**IF (u(k).ne.0.)GOTO 1**

**ENDDO**

1 **DO 10 i=k-1,1,-1**

Transform \( R + u \otimes v \) to upper Hessenberg.

**CALL rotate(r,qt,n,np,i,u(i),-u(i+1))**

**IF (u(i).EQ.0.)THEN**

**U(i)=ABS(U(I+1))**

**ELSE IF (ABS(U(I)).GT.ABS(U(I+1)))THEN**

**U(I)=ABS(U(I))*SQRT(1.+(U(I+1)/U(I))**2)**

**ELSE**

**U(I)=ABS(U(I+1))*SQRT(1.+(U(I)/U(I+1))**2)**

**ENDIF**

**ENDDO**

**DO 15 J=1,N**

**R(1,J)=R(1,J)+U(1)*V(J)**

**ENDDO**

**DO 16 I=1,K-1**

Transform upper Hessenberg matrix to upper triangular.

**CALL rotate(r,qt,n,np,i(r(i,i)),-r(i+1,i))**

**ENDDO**

See [2] for details on how to use QR decomposition for constructing orthogonal bases, and for solving least-squares problems. (We prefer to use SVD, §2.6, for these purposes, because of its greater diagnostic capability in pathological cases.)
2.11 Is Matrix Inversion an \(N^3\) Process?

We close this chapter with a little entertainment, a bit of algorithmic prestidigitation which probes more deeply into the subject of matrix inversion. We start with a seemingly simple question: