```
    enddo 11
    x(i)=sum/p(i)
enddo 12
do 14 i=n,1,-1                        Solve L^T · x = y.
    sum=x(i)
    do 13 k=i+1,n
        sum=sum-a(k,i)*x(k)
    enddo 13
    x(i)=sum/p(i)
enddo 14
return
END
```

A typical use of `choldc` and `cholsl` is in the inversion of covariance matrices describing the fit of data to a model; see, e.g., §15.6. In this, and many other applications, one often needs $\mathbf{L}^{-1}$. The lower triangle of this matrix can be efficiently found from the output of `choldc`:

```
    do 13 i=1,n
        a(i,i)=1./p(i)
        do 12 j=i+1,n
            sum=0.
            do 11 k=i,j-1
                sum=sum-a(j,k)*a(k,i)
            enddo 11
            a(j,i)=sum/p(j)
        enddo 12
    enddo 13
```

CITED REFERENCES AND FURTHER READING:

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag), Chapter I/1.

Gill, P.E., Murray, W., and Wright, M.H. 1991, *Numerical Linear Algebra and Optimization*, vol. 1 (Redwood City, CA: Addison-Wesley), §4.9.2.

Dahlquist, G., and Bjorck, A. 1974, *Numerical Methods* (Englewood Cliffs, NJ: Prentice-Hall), §5.3.5.

Golub, G.H., and Van Loan, C.F. 1989, *Matrix Computations*, 2nd ed. (Baltimore: Johns Hopkins University Press), §4.2.

2.10 QR Decomposition

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

$$\mathbf{A} = \mathbf{Q} \cdot \mathbf{R} \tag{2.10.1}$$

Here $\mathbf{R}$ is upper triangular, while $\mathbf{Q}$ is orthogonal, that is,

$$\mathbf{Q}^T \cdot \mathbf{Q} = \mathbf{1} \tag{2.10.2}$$

where $\mathbf{Q}^T$ is the transpose matrix of $\mathbf{Q}$. 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

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{2.10.3}$$

first form $\mathbf{Q}^T \cdot \mathbf{b}$ and then solve

$$\mathbf{R} \cdot \mathbf{x} = \mathbf{Q}^T \cdot \mathbf{b} \tag{2.10.4}$$

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 $\mathbf{1} - \mathbf{u} \otimes \mathbf{u}/c$ where $c = \frac{1}{2}\mathbf{u} \cdot \mathbf{u}$. 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 $\mathbf{Q}_1$ to zero all elements in the first column of $\mathbf{A}$ below the first element. Similarly $\mathbf{Q}_2$ zeroes all elements in the second column below the second element, and so on up to $\mathbf{Q}_{n-1}$. Thus

$$\mathbf{R} = \mathbf{Q}_{n-1} \cdots \mathbf{Q}_1 \cdot \mathbf{A} \tag{2.10.5}$$

Since the Householder matrices are orthogonal,

$$\mathbf{Q} = (\mathbf{Q}_{n-1} \cdots \mathbf{Q}_1)^{-1} = \mathbf{Q}_1 \cdots \mathbf{Q}_{n-1} \tag{2.10.6}$$

In most applications we don't need to form $\mathbf{Q}$ explicitly; we instead store it in the factored form (2.10.6). Pivoting is not usually necessary unless the matrix $\mathbf{A}$ 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 $\mathbf{R}$ is returned in the upper triangle of `a`, except for the diagonal elements of $\mathbf{R}$ which are returned in `d(1:n)`. The orthogonal matrix $\mathbf{Q}$ is represented as a product of $n-1$ Householder matrices $\mathbf{Q}_1 \ldots \mathbf{Q}_{n-1}$, where $\mathbf{Q}_j = \mathbf{1} - \mathbf{u}_j \otimes \mathbf{u}_j / c_j$. The ith component of $\mathbf{u}_j$ 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
sing=.false.
do 17 k=1,n-1
    scale=0.
    do 11 i=k,n
        scale=max(scale,abs(a(i,k)))
    enddo 11
    if(scale.eq.0.)then                Singular case.
        sing=.true.
        c(k)=0.
        d(k)=0.
    else                               Form Qk and Qk · A.
        do 12 i=k,n
            a(i,k)=a(i,k)/scale
        enddo 12
        sum=0.
        do 13 i=k,n
            sum=sum+a(i,k)**2
        enddo 13
        sigma=sign(sqrt(sum),a(k,k))
        a(k,k)=a(k,k)+sigma
```

```
        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 14
            tau=sum/c(k)
            do 15 i=k,n
                a(i,j)=a(i,j)-tau*a(i,k)
            enddo 15
        enddo 16
    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 11
          tau=sum/c(j)
          do 12 i=j,n
              b(i)=b(i)-tau*a(i,j)
          enddo 12
      enddo 13
      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 11
          b(i)=(b(i)-sum)/d(i)
```

```
    enddo 12
    return
    END
```

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.)

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,

$$\mathbf{A} \rightarrow \mathbf{A} + \mathbf{s} \otimes \mathbf{t} \tag{2.10.7}$$

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

$$\mathbf{A} = \mathbf{Q} \cdot \mathbf{R} \quad \rightarrow \quad \mathbf{A}' = \mathbf{Q}' \cdot \mathbf{R}' = \mathbf{Q} \cdot (\mathbf{R} + \mathbf{u} \otimes \mathbf{v}) \tag{2.10.8}$$

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

$$\mathbf{t} = \mathbf{v} \quad \text{and either} \quad \mathbf{s} = \mathbf{Q} \cdot \mathbf{u} \quad \text{or} \quad \mathbf{u} = \mathbf{Q}^T \cdot \mathbf{s} \tag{2.10.9}$$

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

```
      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 × n matrix, calculates the QR decomposition of
         the matrix Q · (R + u ⊗ 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) ≠ 0.
          if(u(k).ne.0.)goto 1
      enddo 11
      k=1
1     do 12 i=k-1,1,-1                               Transform R + u ⊗ v to upper Hes-
          call rotate(r,qt,n,np,i,u(i),-u(i+1))         senberg.
          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 12
      do 13 j=1,n
          r(1,j)=r(1,j)+u(1)*v(j)
      enddo 13
      do 14 i=1,k-1                                  Transform upper Hessenberg matrix
          call rotate(r,qt,n,np,i,r(i,i),-r(i+1,i))     to upper triangular.
      enddo 14
```

```
return
END

SUBROUTINE rotate(r,qt,n,np,i,a,b)
INTEGER n,np,i
REAL a,b,r(np,np),qt(np,np)
```

Given $\mathtt{n}\times\mathtt{n}$ matrices `r` and `qt` of physical dimension `np`, carry out a Jacobi rotation on rows `i` and $\mathtt{i}+1$ of each matrix. `a` and `b` are the parameters of the rotation: $\cos\theta = a/\sqrt{a^2+b^2}$, $\sin\theta = b/\sqrt{a^2+b^2}$.

```
INTEGER j
REAL c,fact,s,w,y
if(a.eq.0.)then                        Avoid unnecessary overflow or underflow.
    c=0.
    s=sign(1.,b)
else if(abs(a).gt.abs(b))then
    fact=b/a
    c=sign(1./sqrt(1.+fact**2),a)
    s=fact*c
else
    fact=a/b
    s=sign(1./sqrt(1.+fact**2),b)
    c=fact*s
endif
do 11 j=i,n                            Premultiply r by Jacobi rotation.
    y=r(i,j)
    w=r(i+1,j)
    r(i,j)=c*y-s*w
    r(i+1,j)=s*y+c*w
enddo 11
do 12 j=1,n                            Premultiply qt by Jacobi rotation.
    y=qt(i,j)
    w=qt(i+1,j)
    qt(i,j)=c*y-s*w
    qt(i+1,j)=s*y+c*w
enddo 12
return
END
```

We will make use of QR decomposition, and its updating, in §9.7.

CITED REFERENCES AND FURTHER READING:

Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag), Chapter I/8. [1]

Golub, G.H., and Van Loan, C.F. 1989, *Matrix Computations*, 2nd ed. (Baltimore: Johns Hopkins University Press), §§5.2, 5.3, 12.6. [2]

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: