SUBROUTINE asolve(n,b,x,itrnsp)
INTEGER n,itrnsp,i,ija,NMAX,i
DOUBLE PRECISION x(n),b(n),sa
PARAMETER (NMAX=1000)
COMMON /mat/ sa(NMAX),ija(NMAX)
do i=1,n
   x(i)=b(i)/sa(i)
enddo
return
END

2.8 Vandermonde Matrices and Toeplitz Matrices

In §2.4 the case of a tridiagonal matrix was treated specially, because that particular type of linear system admits a solution in only of order \( N \) operations, rather than of order \( N^3 \) for the general linear problem. When such particular types
exist, it is important to know about them. Your computational savings, should you ever happen to be working on a problem that involves the right kind of particular type, can be enormous.

This section treats two special types of matrices that can be solved in order $N^2$ operations, not as good as tridiagonal, but a lot better than the general case. (Other than the operations count, these two types having nothing in common.) Matrices of the first type, termed Vandermonde matrices, occur in some problems having to do with the fitting of polynomials, the reconstruction of distributions from their moments, and also other contexts. In this book, for example, a Vandermonde problem crops up in §3.5. Matrices of the second type, termed Toeplitz matrices, tend to occur in problems involving deconvolution and signal processing. In this book, a Toeplitz problem is encountered in §13.7.

These are not the only special types of matrices worth knowing about. The Hilbert matrices, whose components are of the form $a_{ij} = 1/(i+j-1)$, $i,j = 1,\ldots,N$ can be inverted by an exact integer algorithm, and are very difficult to invert in any other way, since they are notoriously ill-conditioned (see [1] for details). The Sherman-Morrison and Woodbury formulas, discussed in §2.7, can sometimes be used to convert new special forms into old ones. Reference [2] gives some other special forms. We have not found these additional forms to arise as frequently as the two that we now discuss.

### Vandermonde Matrices

A Vandermonde matrix of size $N \times N$ is completely determined by $N$ arbitrary numbers $x_1, x_2, \ldots, x_N$, in terms of which its $N^2$ components are the integer powers $x_i^{j-1}$, $i,j = 1,\ldots,N$. Evidently there are two possible such forms, depending on whether we view the $i$'s as rows, $j$'s as columns, or vice versa. In the former case, we get a linear system of equations that looks like this,

$$
\begin{bmatrix}
1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\
x_2 & x_2^2 & \cdots & x_2^{N-1} \\
\vdots & \vdots & \ddots & \vdots \\
x_N & x_N^2 & \cdots & x_N^{N-1}
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
\vdots \\
c_N
\end{bmatrix}
= 
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_N
\end{bmatrix}
$$

(2.8.1)

Performing the matrix multiplication, you will see that this equation solves for the unknown coefficients $c_i$ which fit a polynomial to the $N$ pairs of abscissas and ordinates $(x_i, y_i)$. Precisely this problem will arise in §3.5, and the routine given there will solve (2.8.1) by the method that we are about to describe.

The alternative identification of rows and columns leads to the set of equations

$$
\begin{bmatrix}
1 & 1 & \cdots & 1 \\
x_1 & x_1^2 & \cdots & x_1^N \\
x_2 & x_2^2 & \cdots & x_2^N \\
\vdots & \vdots & \ddots & \vdots \\
x_N & x_N^2 & \cdots & x_N^N
\end{bmatrix}
\begin{bmatrix}
w_1 \\
w_2 \\
w_3 \\
\vdots \\
w_N
\end{bmatrix}
= 
\begin{bmatrix}
q_1 \\
q_2 \\
q_3 \\
\vdots \\
q_N
\end{bmatrix}
$$

(2.8.2)

Write this out and you will see that it relates to the problem of moments: Given the values of $N$ points $x_i$, find the unknown weights $w_i$, assigned so as to match the given values $q_j$ of the first $N$ moments. (For more on this problem, consult [3].) The routine given in this section solves (2.8.2).
The method of solution of both (2.8.1) and (2.8.2) is closely related to Lagrange's polynomial interpolation formula, which we will not formally meet until §3.1 below. Notwithstanding, the following derivation should be comprehensible:

Let \( P_j(x) \) be the polynomial of degree \( N-1 \) defined by

\[
P_j(x) = \prod_{n \neq j}^{N} \frac{x - x_n}{x_j - x_n} = \sum_{k=1}^{N} A_{jk} x^{k-1}
\]  

(2.8.3)

Here the meaning of the last equality is to define the components of the matrix \( A_{ij} \) as the coefficients that arise when the product is multiplied out and like terms collected.

The polynomial \( P_j(x) \) is a function of \( x \) generally. But you will notice that it is specifically designed so that it takes on a value of zero at all \( x_i \) with \( i \neq j \), and has a value of unity at \( x = x_j \). In other words,

\[
P_j(x_i) = \delta_{ij} = \sum_{k=1}^{N} A_{jk} x_i^{k-1}
\]  

(2.8.4)

But (2.8.4) says that \( A_{jk} \) is exactly the inverse of the matrix of components \( x_i^{k-1} \), which appears in (2.8.2), with the subscript as the column index. Therefore the solution of (2.8.2) is just that matrix inverse times the right-hand side,

\[
w_j = \sum_{k=1}^{N} A_{jk} q_k
\]  

(2.8.5)

As for the transpose problem (2.8.1), we can use the fact that the inverse of the transpose is the transpose of the inverse, so

\[
c_j = \sum_{k=1}^{N} A_{kj} y_k
\]  

(2.8.6)

The routine in §3.5 implements this.

It remains to find a good way of multiplying out the monomial terms in (2.8.3), in order to get the components of \( A_{jk} \). This is essentially a bookkeeping problem, and we will let you read the routine itself to see how it can be solved. One trick is to define a master \( P(x) \) by

\[
P(x) \equiv \prod_{n=1}^{N} (x - x_n)
\]  

(2.8.7)

work out its coefficients, and then obtain the numerators and denominators of the specific \( P_j \)'s via synthetic division by the one supernumerary term. (See §5.3 for more on synthetic division.) Since each such division is only a process of order \( N \), the total procedure is of order \( N^2 \).

You should be warned that Vandermonde systems are notoriously ill-conditioned, by their very nature. (As an aside anticipating §5.8, the reason is the same as that which makes Chebyshev fitting so impressively accurate: there exist high-order polynomials that are very good uniform fits to zero. Hence roundoff error can introduce rather substantial coefficients of the leading terms of these polynomials.) It is a good idea always to compute Vandermonde problems in double precision.

The routine for (2.8.2) which follows is due to G.B. Rybicki.

SUBROUTINE vander(x, w, q, n)
INTEGER n, NMAX
DOUBLE PRECISION q(n), w(n), x(n)
PARAMETER (NMAX=100)
Solves the Vandermonde linear system \( \sum_{i=1}^{N} x_i^{k-1} w_i = q_k \) \( (k = 1, \ldots, N) \). Input consists of the vectors \( x(1:n) \) and \( q(1:n) \); the vector \( w(1:n) \) is output.
Parameters: NMAX is the maximum expected value of n.

INTEGER i,j,k
DOUBLE PRECISION b,s,t,xx,c(NMAX)
if(n.eq.1)then
  w(1)=q(1)
else
  do 11 i=1,n
    c(i)=0.d0
  enddo 11
  c(n)=-x(1)
  Coefficients of the master polynomial are found by recursion.
  do 13 i=2,n
    xx=-x(i)
    do 12 j=n+1-i,n-1
      c(j)=c(j)+xx*c(j+1)
    enddo 12
    c(n)=c(n)+xx
  enddo 13
  do 15 i=1,n
    Each subfactor in turn
    xx=x(i)
    t=1.d0
    b=1.d0
    s=q(n)
    do 14 k=n,2,-1
      b=c(k)+xx*b
      s=s+q(k-1)*b
      matrix-multiplied by the right-hand side,
      t=xx*t+b
    enddo 14
    w(i)=s/t
    and supplied with a denominator.
  enddo 15
endif
return
END

Toeplitz Matrices

An $N \times N$ Toeplitz matrix is specified by giving $2N - 1$ numbers $R_k, k = -N + 1, \ldots, -1, 0, 1, \ldots, N - 1$. Those numbers are then emplaced as matrix elements constant along the (upper-left to lower-right) diagonals of the matrix:

$\begin{bmatrix}
R_0 & R_{-1} & R_{-2} & \cdots & R_{-(N-2)} & R_{-(N-1)} \\
R_1 & R_0 & R_{-1} & \cdots & R_{-(N-3)} & R_{-(N-2)} \\
R_2 & R_1 & R_0 & \cdots & R_{-(N-4)} & R_{-(N-3)} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
R_{N-2} & R_{N-3} & R_{N-4} & \cdots & R_0 & R_{-1} \\
R_{N-1} & R_{N-2} & R_{N-3} & \cdots & R_1 & R_0
\end{bmatrix}$ (2.8.8)

The linear Toeplitz problem can thus be written as

$$\sum_{j=1}^{N} R_{i-j} x_j = y_i \quad (i = 1, \ldots, N)$$ (2.8.9)

where the $x_j$'s, $j = 1, \ldots, N$, are the unknowns to be solved for.

The Toeplitz matrix is symmetric if $R_k = R_{-k}$ for all $k$. Levinson [4] developed an algorithm for fast solution of the symmetric Toeplitz problem, by a bordering method, that is, a recursive procedure that solves the $M$-dimensional Toeplitz problem

$$\sum_{j=1}^{M} R_{i-j} x_j^{(M)} = y_i \quad (i = 1, \ldots, M)$$ (2.8.10)
in turn for \( M = 1, 2, \ldots \) until \( M = N \), the desired result, is finally reached. The vector \( x_j^{(M)} \) is the result at the \( M \)th stage, and becomes the desired answer only when \( N \) is reached.

Levinson’s method is well documented in standard texts (e.g., [5]). The useful fact that the method generalizes to the nonsymmetric case seems to be less well known. At some risk of excessive detail, we therefore give a derivation here, due to G.B. Rybicki.

In following a recursion from step \( M \) to step \( M + 1 \) we find that our developing solution \( x_j^{(M)} \) changes in this way:

\[
\sum_{j=1}^{M} R_{i-j} x_j^{(M)} = y_i \quad i = 1, \ldots, M
\]  

(2.8.11)

becomes

\[
\sum_{j=1}^{M} R_{i-j} x_j^{(M+1)} + R_{i-(M+1)} x_{M+1}^{(M+1)} = y_i \quad i = 1, \ldots, M + 1
\]  

(2.8.12)

By eliminating \( y_i \), we find

\[
\sum_{j=1}^{M} R_{i-j} \left( \frac{x_j^{(M)}}{x_{M+1}^{(M+1)}} - \frac{x_j^{(M+1)}}{x_{M+1}^{(M+1)}} \right) = R_{i-(M+1)} \quad i = 1, \ldots, M
\]  

(2.8.13)

or by letting \( i \to M+1-i \) and \( j \to M+1-j \),

\[
\sum_{j=1}^{M} R_{j-i} G_j^{(M)} = R_{-i}
\]  

(2.8.14)

where

\[
G_j^{(M)} \equiv \frac{x_{M+1-j}^{(M+1)} - x_{M+1-i}^{(M+1)}}{x_{M+1}^{(M+1)}}
\]  

(2.8.15)

To put this another way,

\[
x_{M+1-j}^{(M+1)} = x_j^{(M)} - x_{M+1-i}^{(M+1)} G_j^{(M)} \quad j = 1, \ldots, M
\]  

(2.8.16)

Thus, if we can use recursion to find the order \( M \) quantities \( x_j^{(M)} \) and \( G_j^{(M)} \) and the single order \( M + 1 \) quantity \( x_{M+1}^{(M+1)} \), then all of the other \( x_j^{(M+1)} \) will follow. Fortunately, the quantity \( x_{M+1}^{(M+1)} \) follows from equation (2.8.12) with \( i = M + 1 \),

\[
\sum_{j=1}^{M} R_{M+1-j} x_j^{(M+1)} + R_{0} x_{M+1}^{(M+1)} = y_{M+1}
\]  

(2.8.17)

For the unknown order \( M + 1 \) quantities \( x_j^{(M+1)} \) we can substitute the previous order quantities in \( G \) since

\[
G_j^{(M)} = \frac{x_j^{(M)} - x_j^{(M+1)}}{x_{M+1}^{(M+1)}}
\]  

(2.8.18)

The result of this operation is

\[
x_{M+1}^{(M+1)} = \frac{\sum_{j=1}^{M} R_{M+1-j} x_j^{(M)} - y_{M+1}}{\sum_{j=1}^{M} R_{M+1-j} x_j^{(M)} - R_0}
\]  

(2.8.19)

The only remaining problem is to develop a recursion relation for \( G \). Before we do that, however, we should point out that there are actually two distinct sets of solutions to the original linear problem for a nonsymmetric matrix, namely right-hand solutions (which we
have been discussing) and left-hand solutions $z_i$. The formalism for the left-hand solutions differs only in that we deal with the equations

$$\sum_{j=1}^{M} R_{j-i} z_{j}^{(M)} = y_i \quad i = 1, \ldots, M \quad (2.8.20)$$

Then, the same sequence of operations on this set leads to

$$\sum_{j=1}^{M} R_{i-j} H_{j}^{(M)} = R_i \quad (2.8.21)$$

where

$$H_{j}^{(M)} = \frac{z_{j+1}^{(M)} - z_{j+1}^{(M+1)}}{z_{j+1}^{(M+1)}} \quad (2.8.22)$$

(compare with 2.8.14 – 2.8.15). The reason for mentioning the left-hand solutions now is that, by equation (2.8.21), the $H_j$ satisfy exactly the same equation as the $x_j$ except for the substitution $y_i \rightarrow R_i$ on the right-hand side. Therefore we can quickly deduce from equation (2.8.19) that

$$H_{j}^{(M+1)} = \frac{\sum_{j=1}^{M} R_{j-M+1} H_{j}^{(M)} - R_{M+1}}{\sum_{j=1}^{M} R_{j-M+1} G_{j}^{(M)} - R_0} \quad (2.8.23)$$

By the same token, $G$ satisfies the same equation as $z$, except for the substitution $y_i \rightarrow R_{-i}$. This gives

$$G_{j}^{(M+1)} = \frac{\sum_{j=1}^{M} R_{j-M+1} G_{j}^{(M)} - R_{M+1}}{\sum_{j=1}^{M} R_{j-M+1} H_{j}^{(M)} - R_0} \quad (2.8.24)$$

The same “morphism” also turns equation (2.8.16), and its partner for $z$, into the final equations

$$G_{j}^{(M+1)} = G_{j}^{(M)} - G_{M+1}^{(M+1)} H_{j+1}^{(M)} \quad (2.8.25)$$

$$H_{j}^{(M+1)} = H_{j}^{(M)} - H_{M+1}^{(M+1)} G_{j}^{(M)}$$

Now, starting with the initial values

$$x_{1}^{(1)} = y_{1}/R_0 \quad G_{1}^{(1)} = R_{-1}/R_0 \quad H_{1}^{(1)} = R_{1}/R_0 \quad (2.8.26)$$

we can recurse away. At each stage $M$ we use equations (2.8.23) and (2.8.24) to find $H_{M+1}^{(M+1)}, G_{M+1}^{(M+1)}$, and then equation (2.8.25) to find the other components of $H_{M+1}^{(M+1)}, G_{M+1}^{(M+1)}$. From there the vectors $x^{(M+1)}$ and/or $z^{(M+1)}$ are easily calculated.

The program below does this. It incorporates the second equation in (2.8.25) in the form

$$H_{j}^{(M+1)} = H_{j-M+1}^{(M+1)} - H_{j-M+1}^{(M+1)} G_{j}^{(M)} \quad (2.8.27)$$

so that the computation can be done “in place.”

Notice that the above algorithm fails if $R_0 = 0$. In fact, because the bordering method does not allow pivoting, the algorithm will fail if any of the diagonal principal minors of the original Toeplitz matrix vanish. (Compare with discussion of the tridiagonal algorithm in §2.4.) If the algorithm fails, your matrix is not necessarily singular — you might just have to solve your problem by a slower and more general algorithm such as LU decomposition with pivoting.

The routine that implements equations (2.8.23)–(2.8.27) is also due to Rybicki. Note that the routine’s $r$ ($\alpha+j$) is equal to $R_j$ above, so that subscripts on the $r$ array vary from 1 to $2N - 1$. 

Copyright (C) 1986-1992 by Cambridge University Press.
Programs Copyright (C) 1986-1992 by Numerical Recipes Software. Permission is granted for personal use and for making one copy (including for reprinting in hard copy form for non-commercial use) to any server computer. Further reproduction of any of these files (including copying, posting, or distribution to a server computer) is expressly prohibited. To order Numerical Recipes books or software, visit our website: www.nr.com or call 1-800-872-7423 (North America only), or send email to trade@cup.cam.ac.uk (outside North America).
SUBROUTINE toeplz(r,x,y,n)
INTEGER n,NMAX
REAL r(2*n-1),x(n),y(n)
PARAMETER (NMAX=100)
Solves the Toeplitz system \( \sum_{j=1}^{N} R_{(N+i-j)} x_j = y_i \) (i = 1, ..., N). The Toeplitz matrix need not be symmetric. y and r are input arrays of length n and 2*n-1, respectively. x is the output array, of length n.
Parameter: NMAX is the maximum anticipated value of n.
INTEGER j,k,m,m1,m2
REAL pp,pt1,pt2,qq,qt1,qt2,sd,sgd,sgn,shn,sxn,
\* g(NMAX),h(NMAX)
if(r(n).eq.0.) goto 99
x(1)=y(1)/r(n) Initialize for the recursion.
if(n.eq.1) return
g(1)=r(n-1)/r(n)
h(1)=r(n+1)/r(n)
do 15 m=1,n
m1=m+1
sxn=-y(m1)
Compute numerator and denominator for x,
sd=-r(n)
da 11 j=1,m
sxn=sxn+r(n+m1-j)*x(j)
sd=sd+r(n+m1-j)*g(m-j+1)
enddo 11
if(sd.eq.0.) goto 99
x(m1)=sxn/sd whence x.
da 12 j=1,m
x(j)=x(j)-x(m1)*g(m-j+1)
dendo 12
if(m1.eq.n) return
sgn=-r(n-m1)
Compute numerator and denominator for G and H,
shn=-r(n+m1)
sgd=-r(n)
da 13 j=1,m
sgn=sgn+r(n+j-m1)*g(j)
shn=shn+r(n+j-m1)*h(j)
sgd=sgd+r(n+j-m1)*h(m-j+1)
dendo 13
if(sd.eq.0.or.sgd.eq.0.) goto 99
g(m1)=sgn/sgd whence G and H.
h(m1)=shn/sd
k=m
m2=(m+1)/2
pp=g(m1)
qq=h(m1)
da 14 j=1,m2
pt1=g(j)
pt2=g(k)
qt1=h(j)
qt2=h(k)
g(j)=pt1-pp*qt2
g(k)=pt2-qq*qt1
h(j)=qt1-qq*pt2
h(k)=qt2-qq*pt1
k=k-1
enddo 14
Back for another recurrence.
pause 'never get here in toeplz'
pause 'singular principal minor in toeplz'
END

If you are in the business of solving very large Toeplitz systems, you should find out about so-called "new, fast" algorithms, which require only on the order of \( N (\log N)^2 \) operations,
2.9 Cholesky Decomposition

If a square matrix \( A \) happens to be symmetric and positive definite, then it has a special, more efficient, triangular decomposition. Symmetric means that \( a_{ij} = a_{ji} \) for \( i, j = 1, \ldots, N \), while positive definite means that

\[
\mathbf{v} \cdot \mathbf{A} \cdot \mathbf{v} > 0 \quad \text{for all vectors } \mathbf{v}
\]

(2.9.1)

(In Chapter 11 we will see that positive definite has the equivalent interpretation that \( \mathbf{A} \) has all positive eigenvalues.) While symmetric, positive definite matrices are rather special, they occur quite frequently in some applications, so their special factorization, called Cholesky decomposition, is good to know about. When you can use it, Cholesky decomposition is about a factor of two better than alternative methods for solving linear equations.

Instead of seeking arbitrary lower and upper triangular factors \( \mathbf{L} \) and \( \mathbf{U} \), Cholesky decomposition constructs a lower triangular matrix \( \mathbf{L} \) whose transpose \( \mathbf{L}^T \) can itself serve as the upper triangular part. In other words we replace equation (2.3.1) by

\[
\mathbf{L} \cdot \mathbf{L}^T = \mathbf{A} \quad \text{(2.9.2)}
\]

This factorization is sometimes referred to as “taking the square root” of the matrix \( \mathbf{A} \). The components of \( \mathbf{L}^T \) are of course related to those of \( \mathbf{L} \) by

\[
L_{ij}^T = L_{ji} \quad \text{(2.9.3)}
\]

Writing out equation (2.9.2) in components, one readily obtains the analogs of equations (2.3.12)–(2.3.13),

\[
L_{ii} = \left(\alpha_{ii} - \sum_{k=1}^{i-1} L_{ik}^2\right)^{1/2} \quad \text{(2.9.4)}
\]

and

\[
L_{ji} = \frac{1}{L_{ii}} \left(\alpha_{ij} - \sum_{k=1}^{i-1} L_{ik} L_{jk}\right) \quad j = i + 1, i + 2, \ldots, N \quad \text{(2.9.5)}
\]