4.5.14 below) and phase (i) can be omitted. However, if you are confronted with a "nonclassical" weight function $W(x)$, and you don't know the coefficients a_j and b_j, the construction of the associated set of orthogonal polynomials is not trivial. We discuss it at the end of this section.

Computation of the Abscissas and Weights

This task can range from easy to difficult, depending on how much you already know about your weight function and its associated polynomials. In the case of classical, well-studied, orthogonal polynomials, practically everything is known, including good approximations for their zeros. These can be used as starting guesses, enabling Newton's method (to be discussed in §9.4) to converge very rapidly. Newton's method requires the derivative $p'_N(x)$, which is evaluated by standard relations in terms of p_N and p_{N-1}. The weights are then conveniently evaluated by equation (4.5.9). For the following named cases, this direct root-finding is faster, by a factor of 3 to 5, than any other method.

Here are the weight functions, intervals, and recurrence relations that generate the most commonly used orthogonal polynomials and their corresponding Gaussian quadrature formulas.

Gauss-Legendre:

$$W(x) = 1 \qquad -1 < x < 1$$

$$(j+1)P_{j+1} = (2j+1)xP_j - jP_{j-1} \tag{4.5.10}$$

Gauss-Chebyshev:

$$W(x) = (1-x^2)^{-1/2} \qquad -1 < x < 1$$

$$T_{j+1} = 2xT_j - T_{j-1} \tag{4.5.11}$$

Gauss-Laguerre:

$$W(x) = x^\alpha e^{-x} \qquad 0 < x < \infty$$

$$(j+1)L^\alpha_{j+1} = (-x+2j+\alpha+1)L^\alpha_j - (j+\alpha)L^\alpha_{j-1} \tag{4.5.12}$$

Gauss-Hermite:

$$W(x) = e^{-x^2} \qquad -\infty < x < \infty$$

$$H_{j+1} = 2xH_j - 2jH_{j-1} \tag{4.5.13}$$

Gauss-Jacobi:

$$W(x) = (1-x)^\alpha(1+x)^\beta \qquad -1 < x < 1$$

$$c_j P_{j+1}^{(\alpha,\beta)} = (d_j + e_j x) P_j^{(\alpha,\beta)} - f_j P_{j-1}^{(\alpha,\beta)} \tag{4.5.14}$$

where the coefficients c_j, d_j, e_j, and f_j are given by

$$\begin{aligned} c_j &= 2(j+1)(j+\alpha+\beta+1)(2j+\alpha+\beta) \\ d_j &= (2j+\alpha+\beta+1)(\alpha^2-\beta^2) \\ e_j &= (2j+\alpha+\beta)(2j+\alpha+\beta+1)(2j+\alpha+\beta+2) \\ f_j &= 2(j+\alpha)(j+\beta)(2j+\alpha+\beta+2) \end{aligned} \tag{4.5.15}$$

We now give individual routines that calculate the abscissas and weights for these cases. First comes the most common set of abscissas and weights, those of Gauss-Legendre. The routine, due to G.B. Rybicki, uses equation (4.5.9) in the special form for the Gauss-Legendre case,

$$w_j = \frac{2}{(1-x_j^2)[P_N'(x_j)]^2} \tag{4.5.16}$$

The routine also scales the range of integration from (x_1, x_2) to $(-1, 1)$, and provides abscissas x_j and weights w_j for the Gaussian formula

$$\int_{x_1}^{x_2} f(x)dx = \sum_{j=1}^{N} w_j f(x_j) \tag{4.5.17}$$

```
SUBROUTINE gauleg(x1,x2,x,w,n)
INTEGER n
REAL x1,x2,x(n),w(n)
DOUBLE PRECISION EPS
PARAMETER (EPS=3.d-14)                    EPS is the relative precision.
   Given the lower and upper limits of integration x1 and x2, and given n, this routine returns
   arrays x(1:n) and w(1:n) of length n, containing the abscissas and weights of the Gauss-
   Legendre n-point quadrature formula.
INTEGER i,j,m
DOUBLE PRECISION p1,p2,p3,pp,xl,xm,z,z1
   High precision is a good idea for this routine.
m=(n+1)/2                                 The roots are symmetric in the interval, so we
xm=0.5d0*(x2+x1)                             only have to find half of them.
xl=0.5d0*(x2-x1)
do 12 i=1,m                               Loop over the desired roots.
   z=cos(3.141592654d0*(i-.25d0)/(n+.5d0))
     Starting with the above approximation to the ith root, we enter the main loop of re-
     finement by Newton's method.
1  continue
     p1=1.d0
     p2=0.d0
     do 11 j=1,n                          Loop up the recurrence relation to get the Leg-
       p3=p2                                 endre polynomial evaluated at z.
       p2=p1
       p1=((2.d0*j-1.d0)*z*p2-(j-1.d0)*p3)/j
     enddo 11
       p1 is now the desired Legendre polynomial. We next compute pp, its derivative, by
       a standard relation involving also p2, the polynomial of one lower order.
     pp=n*(z*p1-p2)/(z*z-1.d0)
```

```
        z1=z
        z=z1-p1/pp                          Newton's method.
      if(abs(z-z1).gt.EPS)goto 1
      x(i)=xm-xl*z                          Scale the root to the desired interval,
      x(n+1-i)=xm+xl*z                      and put in its symmetric counterpart.
      w(i)=2.d0*xl/((1.d0-z*z)*pp*pp)       Compute the weight
      w(n+1-i)=w(i)                         and its symmetric counterpart.
    enddo 12
    return
    END
```

Next we give three routines that use initial approximations for the roots given by Stroud and Secrest [2]. The first is for Gauss-Laguerre abscissas and weights, to be used with the integration formula

$$\int_0^\infty x^\alpha e^{-x} f(x)dx = \sum_{j=1}^{N} w_j f(x_j) \tag{4.5.18}$$

```
      SUBROUTINE gaulag(x,w,n,alf)
      INTEGER n,MAXIT
      REAL alf,w(n),x(n)
      DOUBLE PRECISION EPS
      PARAMETER (EPS=3.D-14,MAXIT=10)     Increase EPS if you don't have this precision.
C     USES gammln
        Given alf, the parameter α of the Laguerre polynomials, this routine returns arrays x(1:n)
        and w(1:n) containing the abscissas and weights of the n-point Gauss-Laguerre quadrature
        formula. The smallest abscissa is returned in x(1), the largest in x(n).
      INTEGER i,its,j
      REAL ai,gammln
      DOUBLE PRECISION p1,p2,p3,pp,z,z1
        High precision is a good idea for this routine.
      do 13 i=1,n                         Loop over the desired roots.
        if(i.eq.1)then                    Initial guess for the smallest root.
          z=(1.+alf)*(3.+.92*alf)/(1.+2.4*n+1.8*alf)
        else if(i.eq.2)then               Initial guess for the second root.
          z=z+(15.+6.25*alf)/(1.+.9*alf+2.5*n)
        else                              Initial guess for the other roots.
          ai=i-2
          z=z+((1.+2.55*ai)/(1.9*ai)+1.26*ai*alf/
     *         (1.+3.5*ai))*(z-x(i-2))/(1.+.3*alf)
        endif
        do 12 its=1,MAXIT                 Refinement by Newton's method.
          p1=1.d0
          p2=0.d0
          do 11 j=1,n                     Loop up the recurrence relation to get the Laguerre
            p3=p2                           polynomial evaluated at z.
            p2=p1
            p1=((2*j-1+alf-z)*p2-(j-1+alf)*p3)/j
          enddo 11
          p1 is now the desired Laguerre polynomial. We next compute pp, its derivative, by
          a standard relation involving also p2, the polynomial of one lower order.
          pp=(n*p1-(n+alf)*p2)/z
          z1=z
          z=z1-p1/pp                      Newton's formula.
          if(abs(z-z1).le.EPS)goto 1
        enddo 12
        pause 'too many iterations in gaulag'
1       x(i)=z                            Store the root and the weight.
```

```
        w(i)=-exp(gammln(alf+n)-gammln(float(n)))/(pp*n*p2)
enddo 13
return
END
```

Next is a routine for Gauss-Hermite abscissas and weights. If we use the "standard" normalization of these functions, as given in equation (4.5.13), we find that the computations overflow for large N because of various factorials that occur. We can avoid this by using instead the orthonomal set of polynomials $\tilde{H}_j$. They are generated by the recurrence

$$\tilde{H}_{-1} = 0, \quad \tilde{H}_0 = \frac{1}{\pi^{1/4}}, \quad \tilde{H}_{j+1} = x\sqrt{\frac{2}{j+1}}\tilde{H}_j - \sqrt{\frac{j}{j+1}}\tilde{H}_{j-1} \tag{4.5.19}$$

The formula for the weights becomes

$$w_j = \frac{2}{(\tilde{H}'_j)^2} \tag{4.5.20}$$

while the formula for the derivative with this normalization is

$$\tilde{H}'_j = \sqrt{2j}\tilde{H}_{j-1} \tag{4.5.21}$$

The abscissas and weights returned by gauher are used with the integration formula

$$\int_{-\infty}^{\infty} e^{-x^2} f(x)dx = \sum_{j=1}^{N} w_j f(x_j) \tag{4.5.22}$$

```
SUBROUTINE gauher(x,w,n)
INTEGER n,MAXIT
REAL w(n),x(n)
DOUBLE PRECISION EPS,PIM4
PARAMETER (EPS=3.D-14,PIM4=.7511255444649425D0,MAXIT=10)
    Given n, this routine returns arrays x(1:n) and w(1:n) containing the abscissas and
    weights of the n-point Gauss-Hermite quadrature formula. The largest abscissa is returned
    in x(1), the most negative in x(n).
    Parameters: EPS is the relative precision, PIM4 = 1/π^1/4, MAXIT = maximum iterations.
INTEGER i,its,j,m
DOUBLE PRECISION p1,p2,p3,pp,z,z1
  High precision is a good idea for this routine.
m=(n+1)/2
  The roots are symmetric about the origin, so we have to find only half of them.
do 13 i=1,m                              Loop over the desired roots.
    if(i.eq.1)then                       Initial guess for the largest root.
        z=sqrt(float(2*n+1))-1.85575*(2*n+1)**(-.16667)
    else if(i.eq.2)then                  Initial guess for the second largest root.
        z=z-1.14*n**.426/z
    else if (i.eq.3)then                 Initial guess for the third largest root.
        z=1.86*z-.86*x(1)
    else if (i.eq.4)then                 Initial guess for the fourth largest root.
        z=1.91*z-.91*x(2)
    else                                 Initial guess for the other roots.
        z=2.*z-x(i-2)
```