(4.6.9). 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:

$$\begin{aligned} W(x) &= 1 \qquad -1 < x < 1 \\ (j+1)P_{j+1} &= (2j+1)xP_j - jP_{j-1} \end{aligned} \tag{4.6.10}$$

Gauss-Chebyshev:

$$\begin{aligned} W(x) &= (1-x^2)^{-1/2} \qquad -1 < x < 1 \\ T_{j+1} &= 2xT_j - T_{j-1} \end{aligned} \tag{4.6.11}$$

Gauss-Laguerre:

$$\begin{aligned} W(x) &= x^\alpha e^{-x} \qquad 0 < x < \infty \\ (j+1)L^\alpha_{j+1} &= (-x+2j+\alpha+1)L^\alpha_j - (j+\alpha)L^\alpha_{j-1} \end{aligned} \tag{4.6.12}$$

Gauss-Hermite:

$$\begin{aligned} W(x) &= e^{-x^2} \qquad -\infty < x < \infty \\ H_{j+1} &= 2xH_j - 2jH_{j-1} \end{aligned} \tag{4.6.13}$$

Gauss-Jacobi:

$$\begin{aligned} W(x) &= (1-x)^\alpha(1+x)^\beta \qquad -1 < x < 1 \\ c_j P^{(\alpha,\beta)}_{j+1} &= (d_j + e_j x)P^{(\alpha,\beta)}_j - f_j P^{(\alpha,\beta)}_{j-1} \end{aligned} \tag{4.6.14}$$

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

$$\begin{aligned} c_j &= 2(j+1)(j+\alpha+\beta+1)(2j+\alpha+\beta) \\ d_j &= (2j+\alpha+\beta+1)(\alpha^2-\beta^2) \\ e_j &= (2j+\alpha+\beta)(2j+\alpha+\beta+1)(2j+\alpha+\beta+2) \\ f_j &= 2(j+\alpha)(j+\beta)(2j+\alpha+\beta+2) \end{aligned} \tag{4.6.15}$$

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 (4.6.9) in the special form for the Gauss-Legendre case,

$$w_j = \frac{2}{(1-x_j^2)[P'_N(x_j)]^2} \tag{4.6.16}$$

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

$$\int_{x_1}^{x_2} f(x)dx = \sum_{j=0}^{N-1} w_j f(x_j) \tag{4.6.17}$$

182
183
184

gauss wgts.h

```
void gauleg(const Doub x1, const Doub x2, VecDoub_O &x, VecDoub_O &w)
Given the lower and upper limits of integration x1 and x2, this routine returns arrays x[0..n-1]
and w[0..n-1] of length n, containing the abscissas and weights of the Gauss-Legendre n-point
quadrature formula.
{
    const Doub EPS=1.0e-14;                 EPS is the relative precision.
    Doub z1,z,xm,xl,pp,p3,p2,p1;
    Int n=x.size();
    Int m=(n+1)/2;                          The roots are symmetric in the interval, so
    xm=0.5*(x2+x1);                             we only have to find half of them.
    xl=0.5*(x2-x1);
    for (Int i=0;i<m;i++) {                 Loop over the desired roots.
        z=cos(3.141592654*(i+0.75)/(n+0.5));
        Starting with this approximation to the ith root, we enter the main loop of refinement
        by Newton's method.
        do {
            p1=1.0;
            p2=0.0;
            for (Int j=0;j<n;j++) {         Loop up the recurrence relation to get the
                p3=p2;                          Legendre polynomial evaluated at z.
                p2=p1;
                p1=((2.0*j+1.0)*z*p2-j*p3)/(j+1);
            }
            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-1.0);
            z1=z;
            z=z1-p1/pp;                     Newton's method.
        } while (abs(z-z1) > EPS);
        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]=2.0*xl/((1.0-z*z)*pp*pp);      Compute the weight
        w[n-1-i]=w[i];                      and its symmetric counterpart.
    }
}
```

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

$$\int_0^\infty x^\alpha e^{-x} f(x)dx = \sum_{j=0}^{N-1} w_j f(x_j) \tag{4.6.18}$$

gauss wgts.h

```
void gaulag(VecDoub_O &x, VecDoub_O &w, const Doub alf)
Given alf, the parameter α of the Laguerre polynomials, this routine returns arrays x[0..n-1]
and w[0..n-1] containing the abscissas and weights of the n-point Gauss-Laguerre quadrature
formula. The smallest abscissa is returned in x[0], the largest in x[n-1].
{
    const Int MAXIT=10;
    const Doub EPS=1.0e-14;                 EPS is the relative precision.
    Int i,its,j;
    Doub ai,p1,p2,p3,pp,z,z1;
    Int n=x.size();
    for (i=0;i<n;i++) {                     Loop over the desired roots.
        if (i == 0) {                       Initial guess for the smallest root.
            z=(1.0+alf)*(3.0+0.92*alf)/(1.0+2.4*n+1.8*alf);
        } else if (i == 1) {                Initial guess for the second root.
            z += (15.0+6.25*alf)/(1.0+0.9*alf+2.5*n);
        } else {                            Initial guess for the other roots.
            ai=i-1;
```