```
    int j;
    float bi,bim,bip,tox,ans;

    if (n < 2) nrerror("Index n less than 2 in bessi");
    if (x == 0.0)
        return 0.0;
    else {
        tox=2.0/fabs(x);
        bip=ans=0.0;
        bi=1.0;
        for (j=2*(n+(int) sqrt(ACC*n));j>0;j--) {     Downward recurrence from even
            bim=bip+j*tox*bi;                             m.
            bip=bi;
            bi=bim;
            if (fabs(bi) > BIGNO) {         Renormalize to prevent overflows.
                ans *= BIGNI;
                bi *= BIGNI;
                bip *= BIGNI;
            }
            if (j == n) ans=bip;
        }
        ans *= bessi0(x)/bi;                Normalize with bessi0.
        return x < 0.0 && (n & 1) ? -ans : ans;
    }
}
```

CITED REFERENCES AND FURTHER READING:

Abramowitz, M., and Stegun, I.A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, Volume 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), §9.8. [1]

Carrier, G.F., Krook, M. and Pearson, C.E. 1966, *Functions of a Complex Variable* (New York: McGraw-Hill), pp. 220ff.

6.7 Bessel Functions of Fractional Order, Airy Functions, Spherical Bessel Functions

Many algorithms have been proposed for computing Bessel functions of fractional order numerically. Most of them are, in fact, not very good in practice. The routines given here are rather complicated, but they can be recommended wholeheartedly.

Ordinary Bessel Functions

The basic idea is *Steed's method*, which was originally developed [1] for Coulomb wave functions. The method calculates J_ν, J'_ν, Y_ν, and Y'_ν simultaneously, and so involves four relations among these functions. Three of the relations come from two continued fractions, one of which is complex. The fourth is provided by the Wronskian relation

$$W \equiv J_\nu Y'_\nu - Y_\nu J'_\nu = \frac{2}{\pi x} \tag{6.7.1}$$

The first continued fraction, CF1, is defined by

$$\begin{aligned} f_\nu \equiv \frac{J'_\nu}{J_\nu} &= \frac{\nu}{x} - \frac{J_{\nu+1}}{J_\nu} \\ &= \frac{\nu}{x} - \frac{1}{2(\nu+1)/x -}\ \frac{1}{2(\nu+2)/x -} \cdots \end{aligned} \tag{6.7.2}$$

You can easily derive it from the three-term recurrence relation for Bessel functions: Start with equation (6.5.6) and use equation (5.5.18). Forward evaluation of the continued fraction by one of the methods of §5.2 is essentially equivalent to backward recurrence of the recurrence relation. The rate of convergence of CF1 is determined by the position of the *turning point* $x_{\rm tp} = \sqrt{\nu(\nu+1)} \approx \nu$, beyond which the Bessel functions become oscillatory. If $x \lesssim x_{\rm tp}$, convergence is very rapid. If $x \gtrsim x_{\rm tp}$, then each iteration of the continued fraction effectively increases ν by one until $x \lesssim x_{\rm tp}$; thereafter rapid convergence sets in. Thus the number of iterations of CF1 is of order x for large x. In the routine `bessjy` we set the maximum allowed number of iterations to 10,000. For larger x, you can use the usual asymptotic expressions for Bessel functions.

One can show that the sign of J_ν is the same as the sign of the denominator of CF1 once it has converged.

The complex continued fraction CF2 is defined by

$$p + iq \equiv \frac{J'_\nu + iY'_\nu}{J_\nu + iY_\nu} = -\frac{1}{2x} + i + \frac{i}{x}\;\frac{(1/2)^2 - \nu^2}{2(x+i)+}\;\frac{(3/2)^2-\nu^2}{2(x+2i)+}\cdots \tag{6.7.3}$$

(We sketch the derivation of CF2 in the analogous case of modified Bessel functions in the next subsection.) This continued fraction converges rapidly for $x \gtrsim x_{\rm tp}$, while convergence fails as $x \to 0$. We have to adopt a special method for small x, which we describe below. For x not too small, we can ensure that $x \gtrsim x_{\rm tp}$ by a stable recurrence of J_ν and J'_ν downwards to a value $\nu = \mu \lesssim x$, thus yielding the ratio f_μ at this lower value of ν. This is the stable direction for the recurrence relation. The initial values for the recurrence are

$$J_\nu = \text{arbitrary}, \qquad J'_\nu = f_\nu J_\nu, \tag{6.7.4}$$

with the sign of the arbitrary initial value of J_ν chosen to be the sign of the denominator of CF1. Choosing the initial value of J_ν very small minimizes the possibility of overflow during the recurrence. The recurrence relations are

$$\begin{aligned} J_{\nu-1} &= \frac{\nu}{x}J_\nu + J'_\nu \\ J'_{\nu-1} &= \frac{\nu-1}{x}J_{\nu-1} - J_\nu \end{aligned} \tag{6.7.5}$$

Once CF2 has been evaluated at $\nu = \mu$, then with the Wronskian (6.7.1) we have enough relations to solve for all four quantities. The formulas are simplified by introducing the quantity

$$\gamma \equiv \frac{p - f_\mu}{q} \tag{6.7.6}$$

Then

$$J_\mu = \pm\left(\frac{W}{q + \gamma(p - f_\mu)}\right)^{1/2} \tag{6.7.7}$$

$$J'_\mu = f_\mu J_\mu \tag{6.7.8}$$

$$Y_\mu = \gamma J_\mu \tag{6.7.9}$$

$$Y'_\mu = Y_\mu\left(p + \frac{q}{\gamma}\right) \tag{6.7.10}$$

The sign of J_μ in (6.7.7) is chosen to be the same as the sign of the initial J_ν in (6.7.4).

Once all four functions have been determined at the value $\nu = \mu$, we can find them at the original value of ν. For J_ν and J'_ν, simply scale the values in (6.7.4) by the ratio of (6.7.7) to the value found after applying the recurrence (6.7.5). The quantities Y_ν and Y'_ν can be found by starting with the values in (6.7.9) and (6.7.10) and using the stable upwards recurrence

$$Y_{\nu+1} = \frac{2\nu}{x}Y_\nu - Y_{\nu-1} \tag{6.7.11}$$

together with the relation

$$Y'_\nu = \frac{\nu}{x}Y_\nu - Y_{\nu+1} \tag{6.7.12}$$

Now turn to the case of small x, when CF2 is not suitable. Temme [2] has given a good method of evaluating Y_ν and $Y_{\nu+1}$, and hence Y'_ν from (6.7.12), by series expansions that accurately handle the singularity as $x \to 0$. The expansions work only for $|\nu| \leq 1/2$, and so now the recurrence (6.7.5) is used to evaluate f_ν at a value $\nu = \mu$ in this interval. Then one calculates J_μ from

$$J_\mu = \frac{W}{Y'_\mu - Y_\mu f_\mu} \tag{6.7.13}$$

and J'_μ from (6.7.8). The values at the original value of ν are determined by scaling as before, and the Y's are recurred up as before.

Temme's series are

$$Y_\nu = -\sum_{k=0}^{\infty} c_k g_k \qquad Y_{\nu+1} = -\frac{2}{x}\sum_{k=0}^{\infty} c_k h_k \tag{6.7.14}$$

Here

$$c_k = \frac{(-x^2/4)^k}{k!} \tag{6.7.15}$$

while the coefficients g_k and h_k are defined in terms of quantities p_k, q_k, and f_k that can be found by recursion:

$$\begin{aligned} g_k &= f_k + \frac{2}{\nu}\sin^2\left(\frac{\nu\pi}{2}\right) q_k \\ h_k &= -k g_k + p_k \\ p_k &= \frac{p_{k-1}}{k-\nu} \\ q_k &= \frac{q_{k-1}}{k+\nu} \\ f_k &= \frac{k f_{k-1} + p_{k-1} + q_{k-1}}{k^2 - \nu^2} \end{aligned} \tag{6.7.16}$$

The initial values for the recurrences are

$$\begin{aligned} p_0 &= \frac{1}{\pi}\left(\frac{x}{2}\right)^{-\nu}\Gamma(1+\nu) \\ q_0 &= \frac{1}{\pi}\left(\frac{x}{2}\right)^{\nu}\Gamma(1-\nu) \\ f_0 &= \frac{2}{\pi}\frac{\nu\pi}{\sin\nu\pi}\left[\cosh\sigma\Gamma_1(\nu) + \frac{\sinh\sigma}{\sigma}\ln\left(\frac{2}{x}\right)\Gamma_2(\nu)\right] \end{aligned} \tag{6.7.17}$$

with

$$\begin{aligned} \sigma &= \nu\ln\left(\frac{2}{x}\right) \\ \Gamma_1(\nu) &= \frac{1}{2\nu}\left[\frac{1}{\Gamma(1-\nu)} - \frac{1}{\Gamma(1+\nu)}\right] \\ \Gamma_2(\nu) &= \frac{1}{2}\left[\frac{1}{\Gamma(1-\nu)} + \frac{1}{\Gamma(1+\nu)}\right] \end{aligned} \tag{6.7.18}$$

The whole point of writing the formulas in this way is that the potential problems as $\nu \to 0$ can be controlled by evaluating $\nu\pi/\sin\nu\pi$, $\sinh\sigma/\sigma$, and Γ_1 carefully. In particular, Temme gives Chebyshev expansions for $\Gamma_1(\nu)$ and $\Gamma_2(\nu)$. We have rearranged his expansion for Γ_1 to be explicitly an even series in ν so that we can use our routine `chebev` as explained in §5.8.

The routine assumes $\nu \geq 0$. For negative ν you can use the reflection formulas

$$\begin{aligned} J_{-\nu} &= \cos\nu\pi\, J_\nu - \sin\nu\pi\, Y_\nu \\ Y_{-\nu} &= \sin\nu\pi\, J_\nu + \cos\nu\pi\, Y_\nu \end{aligned} \tag{6.7.19}$$

The routine also assumes $x > 0$. For $x < 0$ the functions are in general complex, but expressible in terms of functions with $x > 0$. For $x = 0$, Y_ν is singular.

Internal arithmetic in the routine is carried out in double precision. The complex arithmetic is carried out explicitly with real variables.

```
#include <math.h>
#include "nrutil.h"
#define EPS 1.0e-10
#define FPMIN 1.0e-30
#define MAXIT 10000
#define XMIN 2.0
#define PI 3.141592653589793

void bessjy(float x, float xnu, float *rj, float *ry, float *rjp, float *ryp)
```

Returns the Bessel functions $\mathtt{rj} = J_\nu$, $\mathtt{ry} = Y_\nu$ and their derivatives $\mathtt{rjp} = J'_\nu$, $\mathtt{ryp} = Y'_\nu$, for positive `x` and for $\mathtt{xnu} = \nu \geq 0$. The relative accuracy is within one or two significant digits of EPS, except near a zero of one of the functions, where EPS controls its absolute accuracy. FPMIN is a number close to the machine's smallest floating-point number. All internal arithmetic is in double precision. To convert the entire routine to double precision, change the `float` declarations above to `double` and decrease EPS to 10^{-16}. Also convert the function `beschb`.

```
{
    void beschb(double x, double *gam1, double *gam2, double *gampl,
        double *gammi);
    int i,isign,l,nl;
    double a,b,br,bi,c,cr,ci,d,del,del1,den,di,dlr,dli,dr,e,f,fact,fact2,
        fact3,ff,gam,gam1,gam2,gammi,gampl,h,p,pimu,pimu2,q,r,rjl,
        rjl1,rjmu,rjp1,rjpl,rjtemp,ry1,rymu,rymup,rytemp,sum,sum1,
        temp,w,x2,xi,xi2,xmu,xmu2;

    if (x <= 0.0 || xnu < 0.0) nrerror("bad arguments in bessjy");
    nl=(x < XMIN ? (int)(xnu+0.5) : IMAX(0,(int)(xnu-x+1.5)));
```

`nl` is the number of downward recurrences of the J's and upward recurrences of Y's. `xmu` lies between $-1/2$ and $1/2$ for $\mathtt{x} < \mathtt{XMIN}$, while it is chosen so that `x` is greater than the turning point for $\mathtt{x} \geq \mathtt{XMIN}$.

```
    xmu=xnu-nl;
    xmu2=xmu*xmu;
    xi=1.0/x;
    xi2=2.0*xi;
    w=xi2/PI;                          The Wronskian.
    isign=1;                           Evaluate CF1 by modified Lentz's method (§5.2).
    h=xnu*xi;                              isign keeps track of sign changes in the de-
    if (h < FPMIN) h=FPMIN;                nominator.
    b=xi2*xnu;
    d=0.0;
    c=h;
    for (i=1;i<=MAXIT;i++) {
        b += xi2;
        d=b-d;
        if (fabs(d) < FPMIN) d=FPMIN;
        c=b-1.0/c;
        if (fabs(c) < FPMIN) c=FPMIN;
        d=1.0/d;
        del=c*d;
        h=del*h;
        if (d < 0.0) isign = -isign;
        if (fabs(del-1.0) < EPS) break;
    }
    if (i > MAXIT) nrerror("x too large in bessjy; try asymptotic expansion");
    rjl=isign*FPMIN;                   Initialize J_ν and J'_ν for downward recurrence.
    rjpl=h*rjl;
    rjl1=rjl;                          Store values for later rescaling.
    rjp1=rjpl;
    fact=xnu*xi;
    for (l=nl;l>=1;l--) {
        rjtemp=fact*rjl+rjpl;
        fact -= xi;
```

```
        rjpl=fact*rjtemp-rjl;
        rjl=rjtemp;
    }
    if (rjl == 0.0) rjl=EPS;
    f=rjpl/rjl;                              Now have unnormalized J_μ and J'_μ.
    if (x < XMIN) {                          Use series.
        x2=0.5*x;
        pimu=PI*xmu;
        fact = (fabs(pimu) < EPS ? 1.0 : pimu/sin(pimu));
        d = -log(x2);
        e=xmu*d;
        fact2 = (fabs(e) < EPS ? 1.0 : sinh(e)/e);
        beschb(xmu,&gam1,&gam2,&gampl,&gammi);    Chebyshev evaluation of Γ1 and Γ2.
        ff=2.0/PI*fact*(gam1*cosh(e)+gam2*fact2*d);      f0.
        e=exp(e);
        p=e/(gampl*PI);                      p0.
        q=1.0/(e*PI*gammi);                  q0.
        pimu2=0.5*pimu;
        fact3 = (fabs(pimu2) < EPS ? 1.0 : sin(pimu2)/pimu2);
        r=PI*pimu2*fact3*fact3;
        c=1.0;
        d = -x2*x2;
        sum=ff+r*q;
        sum1=p;
        for (i=1;i<=MAXIT;i++) {
            ff=(i*ff+p+q)/(i*i-xmu2);
            c *= (d/i);
            p /= (i-xmu);
            q /= (i+xmu);
            del=c*(ff+r*q);
            sum += del;
            del1=c*p-i*del;
            sum1 += del1;
            if (fabs(del) < (1.0+fabs(sum))*EPS) break;
        }
        if (i > MAXIT) nrerror("bessy series failed to converge");
        rymu = -sum;
        ry1 = -sum1*xi2;
        rymup=xmu*xi*rymu-ry1;
        rjmu=w/(rymup-f*rymu);               Equation (6.7.13).
    } else {                                 Evaluate CF2 by modified Lentz's method (§5.2).
        a=0.25-xmu2;
        p = -0.5*xi;
        q=1.0;
        br=2.0*x;
        bi=2.0;
        fact=a*xi/(p*p+q*q);
        cr=br+q*fact;
        ci=bi+p*fact;
        den=br*br+bi*bi;
        dr=br/den;
        di = -bi/den;
        dlr=cr*dr-ci*di;
        dli=cr*di+ci*dr;
        temp=p*dlr-q*dli;
        q=p*dli+q*dlr;
        p=temp;
        for (i=2;i<=MAXIT;i++) {
            a += 2*(i-1);
            bi += 2.0;
            dr=a*dr+br;
            di=a*di+bi;
            if (fabs(dr)+fabs(di) < FPMIN) dr=FPMIN;
            fact=a/(cr*cr+ci*ci);
```

```
            cr=br+cr*fact;
            ci=bi-ci*fact;
            if (fabs(cr)+fabs(ci) < FPMIN) cr=FPMIN;
            den=dr*dr+di*di;
            dr /= den;
            di /= -den;
            dlr=cr*dr-ci*di;
            dli=cr*di+ci*dr;
            temp=p*dlr-q*dli;
            q=p*dli+q*dlr;
            p=temp;
            if (fabs(dlr-1.0)+fabs(dli) < EPS) break;
        }
        if (i > MAXIT) nrerror("cf2 failed in bessjy");
        gam=(p-f)/q;                        Equations (6.7.6) – (6.7.10).
        rjmu=sqrt(w/((p-f)*gam+q));
        rjmu=SIGN(rjmu,rjl);
        rymu=rjmu*gam;
        rymup=rymu*(p+q/gam);
        ry1=xmu*xi*rymu-rymup;
    }
    fact=rjmu/rjl;
    *rj=rjl1*fact;                          Scale original J_ν and J'_ν.
    *rjp=rjp1*fact;
    for (i=1;i<=nl;i++) {                   Upward recurrence of Y_ν.
        rytemp=(xmu+i)*xi2*ry1-rymu;
        rymu=ry1;
        ry1=rytemp;
    }
    *ry=rymu;
    *ryp=xnu*xi*rymu-ry1;
}
```

```
#define NUSE1 5
#define NUSE2 5

void beschb(double x, double *gam1, double *gam2, double *gampl, double *gammi)
```
Evaluates Γ_1 and Γ_2 by Chebyshev expansion for $|x| \leq 1/2$. Also returns $1/\Gamma(1+x)$ and $1/\Gamma(1-x)$. If converting to double precision, set $\mathtt{NUSE1} = 7$, $\mathtt{NUSE2} = 8$.
```
{
    float chebev(float a, float b, float c[], int m, float x);
    float xx;
    static float c1[] = {
        -1.142022680371168e0,6.5165112670737e-3,
        3.087090173086e-4,-3.4706269649e-6,6.9437664e-9,
        3.67795e-11,-1.356e-13};
    static float c2[] = {
        1.843740587300905e0,-7.68528408447867e-2,
        1.2719271366546e-3,-4.9717367042e-6,-3.31261198e-8,
        2.423096e-10,-1.702e-13,-1.49e-15};

    xx=8.0*x*x-1.0;                         Multiply x by 2 to make range be −1 to 1,
    *gam1=chebev(-1.0,1.0,c1,NUSE1,xx);         and then apply transformation for eval-
    *gam2=chebev(-1.0,1.0,c2,NUSE2,xx);         uating even Chebyshev series.
    *gampl= *gam2-x*(*gam1);
    *gammi= *gam2+x*(*gam1);
}
```

Modified Bessel Functions

Steed's method does not work for modified Bessel functions because in this case CF2 is purely imaginary and we have only three relations among the four functions. Temme [3] has given a normalization condition that provides the fourth relation.

The Wronskian relation is

$$W \equiv I_\nu K'_\nu - K_\nu I'_\nu = -\frac{1}{x} \tag{6.7.20}$$

The continued fraction CF1 becomes

$$f_\nu \equiv \frac{I'_\nu}{I_\nu} = \frac{\nu}{x} + \frac{1}{2(\nu+1)/x+}\ \frac{1}{2(\nu+2)/x+}\cdots \tag{6.7.21}$$

To get CF2 and the normalization condition in a convenient form, consider the sequence of confluent hypergeometric functions

$$z_n(x) = U(\nu + 1/2 + n, 2\nu + 1, 2x) \tag{6.7.22}$$

for fixed ν. Then

$$K_\nu(x) = \pi^{1/2}(2x)^\nu e^{-x} z_0(x) \tag{6.7.23}$$

$$\frac{K_{\nu+1}(x)}{K_\nu(x)} = \frac{1}{x}\left[\nu + \frac{1}{2} + x + \left(\nu^2 - \frac{1}{4}\right)\frac{z_1}{z_0}\right] \tag{6.7.24}$$

Equation (6.7.23) is the standard expression for K_ν in terms of a confluent hypergeometric function, while equation (6.7.24) follows from relations between contiguous confluent hypergeometric functions (equations 13.4.16 and 13.4.18 in Abramowitz and Stegun). Now the functions z_n satisfy the three-term recurrence relation (equation 13.4.15 in Abramowitz and Stegun)

$$z_{n-1}(x) = b_n z_n(x) + a_{n+1} z_{n+1} \tag{6.7.25}$$

with

$$\begin{aligned} b_n &= 2(n+x) \\ a_{n+1} &= -[(n+1/2)^2 - \nu^2] \end{aligned} \tag{6.7.26}$$

Following the steps leading to equation (5.5.18), we get the continued fraction CF2

$$\frac{z_1}{z_0} = \frac{1}{b_1+}\ \frac{a_2}{b_2+}\cdots \tag{6.7.27}$$

from which (6.7.24) gives $K_{\nu+1}/K_\nu$ and thus K'_ν/K_ν.

Temme's normalization condition is that

$$\sum_{n=0}^{\infty} C_n z_n = \left(\frac{1}{2x}\right)^{\nu+1/2} \tag{6.7.28}$$

where

$$C_n = \frac{(-1)^n}{n!}\frac{\Gamma(\nu+1/2+n)}{\Gamma(\nu+1/2-n)} \tag{6.7.29}$$

Note that the C_n's can be determined by recursion:

$$C_0 = 1, \qquad C_{n+1} = -\frac{a_{n+1}}{n+1}C_n \tag{6.7.30}$$

We use the condition (6.7.28) by finding

$$S = \sum_{n=1}^{\infty} C_n \frac{z_n}{z_0} \tag{6.7.31}$$

Then

$$z_0 = \left(\frac{1}{2x}\right)^{\nu+1/2}\frac{1}{1+S} \tag{6.7.32}$$

and (6.7.23) gives K_ν.

Thompson and Barnett [4] have given a clever method of doing the sum (6.7.31) simultaneously with the forward evaluation of the continued fraction CF2. Suppose the continued fraction is being evaluated as

$$\frac{z_1}{z_0} = \sum_{n=0}^{\infty} \Delta h_n \tag{6.7.33}$$

where the increments Δh_n are being found by, e.g., Steed's algorithm or the modified Lentz's algorithm of §5.2. Then the approximation to S keeping the first N terms can be found as

$$S_N = \sum_{n=1}^{N} Q_n \Delta h_n \tag{6.7.34}$$

Here

$$Q_n = \sum_{k=1}^{n} C_k q_k \tag{6.7.35}$$

and q_k is found by recursion from

$$q_{k+1} = (q_{k-1} - b_k q_k)/a_{k+1} \tag{6.7.36}$$

starting with $q_0 = 0$, $q_1 = 1$. For the case at hand, approximately three times as many terms are needed to get S to converge as are needed simply for CF2 to converge.

To find K_ν and $K_{\nu+1}$ for small x we use series analogous to (6.7.14):

$$K_\nu = \sum_{k=0}^{\infty} c_k f_k \qquad K_{\nu+1} = \frac{2}{x}\sum_{k=0}^{\infty} c_k h_k \tag{6.7.37}$$

Here

$$\begin{aligned} c_k &= \frac{(x^2/4)^k}{k!} \\ h_k &= -k f_k + p_k \\ p_k &= \frac{p_{k-1}}{k-\nu} \\ q_k &= \frac{q_{k-1}}{k+\nu} \\ f_k &= \frac{k f_{k-1} + p_{k-1} + q_{k-1}}{k^2 - \nu^2} \end{aligned} \tag{6.7.38}$$

The initial values for the recurrences are

$$\begin{aligned} p_0 &= \frac{1}{2}\left(\frac{x}{2}\right)^{-\nu} \Gamma(1+\nu) \\ q_0 &= \frac{1}{2}\left(\frac{x}{2}\right)^{\nu} \Gamma(1-\nu) \\ f_0 &= \frac{\nu\pi}{\sin\nu\pi}\left[\cosh\sigma\Gamma_1(\nu) + \frac{\sinh\sigma}{\sigma}\ln\left(\frac{2}{x}\right)\Gamma_2(\nu)\right] \end{aligned} \tag{6.7.39}$$

Both the series for small x, and CF2 and the normalization relation (6.7.28) require $|\nu| \le 1/2$. In both cases, therefore, we recurse I_ν down to a value $\nu = \mu$ in this interval, find K_μ there, and recurse K_ν back up to the original value of ν.

The routine assumes $\nu \ge 0$. For negative ν use the reflection formulas

$$\begin{aligned} I_{-\nu} &= I_\nu + \frac{2}{\pi}\sin(\nu\pi)\, K_\nu \\ K_{-\nu} &= K_\nu \end{aligned} \tag{6.7.40}$$

Note that for large x, $I_\nu \sim e^x$, $K_\nu \sim e^{-x}$, and so these functions will overflow or underflow. It is often desirable to be able to compute the scaled quantities $e^{-x}I_\nu$ and $e^x K_\nu$. Simply omitting the factor e^{-x} in equation (6.7.23) will ensure that all four quantities will have the appropriate scaling. If you also want to scale the four quantities for small x when the series in equation (6.7.37) are used, you must multiply each series by e^x.

```
#include <math.h>
#define EPS 1.0e-10
#define FPMIN 1.0e-30
#define MAXIT 10000
#define XMIN 2.0
#define PI 3.141592653589793

void bessik(float x, float xnu, float *ri, float *rk, float *rip, float *rkp)
```

Returns the modified Bessel functions $\texttt{ri} = I_\nu$, $\texttt{rk} = K_\nu$ and their derivatives $\texttt{rip} = I'_\nu$, $\texttt{rkp} = K'_\nu$, for positive `x` and for $\texttt{xnu} = \nu \ge 0$. The relative accuracy is within one or two significant digits of EPS. FPMIN is a number close to the machine's smallest floating-point number. All internal arithmetic is in double precision. To convert the entire routine to double precision, change the `float` declarations above to `double` and decrease EPS to 10^{-16}. Also convert the function `beschb`.

```
{
    void beschb(double x, double *gam1, double *gam2, double *gampl,
        double *gammi);
    void nrerror(char error_text[]);
    int i,l,nl;
    double a,a1,b,c,d,del,del1,delh,dels,e,f,fact,fact2,ff,gam1,gam2,
        gammi,gampl,h,p,pimu,q,q1,q2,qnew,ril,ril1,rimu,rip1,ripl,
        ritemp,rk1,rkmu,rkmup,rktemp,s,sum,sum1,x2,xi,xi2,xmu,xmu2;

    if (x <= 0.0 || xnu < 0.0) nrerror("bad arguments in bessik");
    nl=(int)(xnu+0.5);                     nl is the number of downward re-
    xmu=xnu-nl;                                currences of the I's and upward
    xmu2=xmu*xmu;                              recurrences of K's. xmu lies be-
    xi=1.0/x;                                  tween -1/2 and 1/2.
    xi2=2.0*xi;
    h=xnu*xi;                              Evaluate CF1 by modified Lentz's
    if (h < FPMIN) h=FPMIN;                    method (§5.2).
    b=xi2*xnu;
    d=0.0;
    c=h;
    for (i=1;i<=MAXIT;i++) {
        b += xi2;
        d=1.0/(b+d);                       Denominators cannot be zero here,
        c=b+1.0/c;                             so no need for special precau-
        del=c*d;                               tions.
        h=del*h;
        if (fabs(del-1.0) < EPS) break;
    }
    if (i > MAXIT) nrerror("x too large in bessik; try asymptotic expansion");
    ril=FPMIN;                             Initialize I_ν and I'_ν for downward re-
    ripl=h*ril;                                currence.
    ril1=ril;                              Store values for later rescaling.
    rip1=ripl;
    fact=xnu*xi;
    for (l=nl;l>=1;l--) {
        ritemp=fact*ril+ripl;
        fact -= xi;
        ripl=fact*ritemp+ril;
        ril=ritemp;
    }
    f=ripl/ril;                            Now have unnormalized I_μ and I'_μ.
    if (x < XMIN) {                        Use series.
        x2=0.5*x;
        pimu=PI*xmu;
        fact = (fabs(pimu) < EPS ? 1.0 : pimu/sin(pimu));
        d = -log(x2);
        e=xmu*d;
        fact2 = (fabs(e) < EPS ? 1.0 : sinh(e)/e);
        beschb(xmu,&gam1,&gam2,&gampl,&gammi);  Chebyshev evaluation of Γ1 and Γ2.
        ff=fact*(gam1*cosh(e)+gam2*fact2*d);    f0.
```

```
        sum=ff;
        e=exp(e);
        p=0.5*e/gampl;                          p0.
        q=0.5/(e*gammi);                        q0.
        c=1.0;
        d=x2*x2;
        sum1=p;
        for (i=1;i<=MAXIT;i++) {
            ff=(i*ff+p+q)/(i*i-xmu2);
            c *= (d/i);
            p /= (i-xmu);
            q /= (i+xmu);
            del=c*ff;
            sum += del;
            del1=c*(p-i*ff);
            sum1 += del1;
            if (fabs(del) < fabs(sum)*EPS) break;
        }
        if (i > MAXIT) nrerror("bessk series failed to converge");
        rkmu=sum;
        rk1=sum1*xi2;
    } else {                                    Evaluate CF2 by Steed's algorithm
        b=2.0*(1.0+x);                              (§5.2), which is OK because there
        d=1.0/b;                                    can be no zero denominators.
        h=delh=d;
        q1=0.0;                                 Initializations for recurrence (6.7.35).
        q2=1.0;
        a1=0.25-xmu2;
        q=c=a1;                                 First term in equation (6.7.34).
        a = -a1;
        s=1.0+q*delh;
        for (i=2;i<=MAXIT;i++) {
            a -= 2*(i-1);
            c = -a*c/i;
            qnew=(q1-b*q2)/a;
            q1=q2;
            q2=qnew;
            q += c*qnew;
            b += 2.0;
            d=1.0/(b+a*d);
            delh=(b*d-1.0)*delh;
            h += delh;
            dels=q*delh;
            s += dels;
            if (fabs(dels/s) < EPS) break;
            Need only test convergence of sum since CF2 itself converges more quickly.
        }
        if (i > MAXIT) nrerror("bessik: failure to converge in cf2");
        h=a1*h;
        rkmu=sqrt(PI/(2.0*x))*exp(-x)/s;        Omit the factor exp(-x) to scale
        rk1=rkmu*(xmu+x+0.5-h)*xi;                  all the returned functions by exp(x)
    }                                               for x >= XMIN.
    rkmup=xmu*xi*rkmu-rk1;
    rimu=xi/(f*rkmu-rkmup);                     Get I_mu from Wronskian.
    *ri=(rimu*ril1)/ril;                        Scale original I_nu and I'_nu.
    *rip=(rimu*rip1)/ril;
    for (i=1;i<=nl;i++) {                       Upward recurrence of K_nu.
        rktemp=(xmu+i)*xi2*rk1+rkmu;
        rkmu=rk1;
        rk1=rktemp;
    }
    *rk=rkmu;
    *rkp=xnu*xi*rkmu-rk1;
}
```

Airy Functions

For positive x, the Airy functions are defined by

$$\mathrm{Ai}(x) = \frac{1}{\pi}\sqrt{\frac{x}{3}}K_{1/3}(z) \tag{6.7.41}$$

$$\mathrm{Bi}(x) = \sqrt{\frac{x}{3}}[I_{1/3}(z) + I_{-1/3}(z)] \tag{6.7.42}$$

where

$$z = \frac{2}{3}x^{3/2} \tag{6.7.43}$$

By using the reflection formula (6.7.40), we can convert (6.7.42) into the computationally more useful form

$$\mathrm{Bi}(x) = \sqrt{x}\left[\frac{2}{\sqrt{3}}I_{1/3}(z) + \frac{1}{\pi}K_{1/3}(z)\right] \tag{6.7.44}$$

so that Ai and Bi can be evaluated with a single call to `bessik`.

The derivatives should not be evaluated by simply differentiating the above expressions because of possible subtraction errors near $x = 0$. Instead, use the equivalent expressions

$$\begin{aligned} \mathrm{Ai}'(x) &= -\frac{x}{\pi\sqrt{3}}K_{2/3}(z) \\ \mathrm{Bi}'(x) &= x\left[\frac{2}{\sqrt{3}}I_{2/3}(z) + \frac{1}{\pi}K_{2/3}(z)\right] \end{aligned} \tag{6.7.45}$$

The corresponding formulas for negative arguments are

$$\begin{aligned} \mathrm{Ai}(-x) &= \quad \frac{\sqrt{x}}{2}\left[J_{1/3}(z) - \frac{1}{\sqrt{3}}Y_{1/3}(z)\right] \\ \mathrm{Bi}(-x) &= -\frac{\sqrt{x}}{2}\left[\frac{1}{\sqrt{3}}J_{1/3}(z) + Y_{1/3}(z)\right] \\ \mathrm{Ai}'(-x) &= \quad \frac{x}{2}\left[J_{2/3}(z) + \frac{1}{\sqrt{3}}Y_{2/3}(z)\right] \\ \mathrm{Bi}'(-x) &= \quad \frac{x}{2}\left[\frac{1}{\sqrt{3}}J_{2/3}(z) - Y_{2/3}(z)\right] \end{aligned} \tag{6.7.46}$$

```
#include <math.h>
#define PI 3.1415927
#define THIRD (1.0/3.0)
#define TWOTHR (2.0*THIRD)
#define ONOVRT 0.57735027

void airy(float x, float *ai, float *bi, float *aip, float *bip)
```
Returns Airy functions $\mathrm{Ai}(x)$, $\mathrm{Bi}(x)$, and their derivatives $\mathrm{Ai}'(x)$, $\mathrm{Bi}'(x)$.
```
{
    void bessik(float x, float xnu, float *ri, float *rk, float *rip,
        float *rkp);
    void bessjy(float x, float xnu, float *rj, float *ry, float *rjp,
        float *ryp);
    float absx,ri,rip,rj,rjp,rk,rkp,rootx,ry,ryp,z;

    absx=fabs(x);
    rootx=sqrt(absx);
    z=TWOTHR*absx*rootx;
    if (x > 0.0) {
        bessik(z,THIRD,&ri,&rk,&rip,&rkp);
        *ai=rootx*ONOVRT*rk/PI;
```

```
        *bi=rootx*(rk/PI+2.0*ONOVRT*ri);
        bessik(z,TWOTHR,&ri,&rk,&rip,&rkp);
        *aip = -x*ONOVRT*rk/PI;
        *bip=x*(rk/PI+2.0*ONOVRT*ri);
    } else if (x < 0.0) {
        bessjy(z,THIRD,&rj,&ry,&rjp,&ryp);
        *ai=0.5*rootx*(rj-ONOVRT*ry);
        *bi = -0.5*rootx*(ry+ONOVRT*rj);
        bessjy(z,TWOTHR,&rj,&ry,&rjp,&ryp);
        *aip=0.5*absx*(ONOVRT*ry+rj);
        *bip=0.5*absx*(ONOVRT*rj-ry);
    } else {                        Case x = 0.
        *ai=0.35502805;
        *bi=(*ai)/ONOVRT;
        *aip = -0.25881940;
        *bip = -(*aip)/ONOVRT;
    }
}
```

Spherical Bessel Functions

For integer n, spherical Bessel functions are defined by

$$
\begin{aligned}
j_n(x) &= \sqrt{\frac{\pi}{2x}} J_{n+(1/2)}(x) \\
y_n(x) &= \sqrt{\frac{\pi}{2x}} Y_{n+(1/2)}(x)
\end{aligned}
\tag{6.7.47}
$$

They can be evaluated by a call to bessjy, and the derivatives can safely be found from the derivatives of equation (6.7.47).

Note that in the continued fraction CF2 in (6.7.3) just the first term survives for $\nu = 1/2$. Thus one can make a very simple algorithm for spherical Bessel functions along the lines of bessjy by always recursing j_n down to $n = 0$, setting p and q from the first term in CF2, and then recursing y_n up. No special series is required near $x = 0$. However, bessjy is already so efficient that we have not bothered to provide an independent routine for spherical Bessels.

```
#include <math.h>
#define RTPIO2 1.2533141

void sphbes(int n, float x, float *sj, float *sy, float *sjp, float *syp)
Returns spherical Bessel functions jn(x), yn(x), and their derivatives j'n(x), y'n(x) for integer n.
{
    void bessjy(float x, float xnu, float *rj, float *ry, float *rjp,
        float *ryp);
    void nrerror(char error_text[]);
    float factor,order,rj,rjp,ry,ryp;

    if (n < 0 || x <= 0.0) nrerror("bad arguments in sphbes");
    order=n+0.5;
    bessjy(x,order,&rj,&ry,&rjp,&ryp);
    factor=RTPIO2/sqrt(x);
    *sj=factor*rj;
    *sy=factor*ry;
    *sjp=factor*rjp-(*sj)/(2.0*x);
    *syp=factor*ryp-(*sy)/(2.0*x);
}
```

CITED REFERENCES AND FURTHER READING:

Barnett, A.R., Feng, D.H., Steed, J.W., and Goldfarb, L.J.B. 1974, *Computer Physics Communications*, vol. 8, pp. 377–395. [1]

Temme, N.M. 1976, *Journal of Computational Physics*, vol. 21, pp. 343–350 [2]; 1975, *op. cit.*, vol. 19, pp. 324–337. [3]

Thompson, I.J., and Barnett, A.R. 1987, *Computer Physics Communications*, vol. 47, pp. 245–257. [4]

Barnett, A.R. 1981, *Computer Physics Communications*, vol. 21, pp. 297–314.

Thompson, I.J., and Barnett, A.R. 1986, *Journal of Computational Physics*, vol. 64, pp. 490–509.

Abramowitz, M., and Stegun, I.A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, Volume 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), Chapter 10.

6.8 Spherical Harmonics

Spherical harmonics occur in a large variety of physical problems, for example, whenever a wave equation, or Laplace's equation, is solved by separation of variables in spherical coordinates. The spherical harmonic $Y_{lm}(\theta, \phi)$, $-l \leq m \leq l$, is a function of the two coordinates θ, ϕ on the surface of a sphere.

The spherical harmonics are orthogonal for different l and m, and they are normalized so that their integrated square over the sphere is unity:

$$\int_0^{2\pi} d\phi \int_{-1}^{1} d(\cos\theta) Y_{l'm'}{}^*(\theta,\phi) Y_{lm}(\theta,\phi) = \delta_{l'l}\delta_{m'm} \tag{6.8.1}$$

Here asterisk denotes complex conjugation.

Mathematically, the spherical harmonics are related to *associated Legendre polynomials* by the equation

$$Y_{lm}(\theta,\phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_l^m(\cos\theta) e^{im\phi} \tag{6.8.2}$$

By using the relation

$$Y_{l,-m}(\theta,\phi) = (-1)^m Y_{lm}{}^*(\theta,\phi) \tag{6.8.3}$$

we can always relate a spherical harmonic to an associated Legendre polynomial with $m \geq 0$. With $x \equiv \cos\theta$, these are defined in terms of the ordinary Legendre polynomials (cf. §4.5 and §5.5) by

$$P_l^m(x) = (-1)^m (1-x^2)^{m/2} \frac{d^m}{dx^m} P_l(x) \tag{6.8.4}$$