NUMERICAL RECIPES

Webnote No. 8, Rev. 2

Bessel Function Implementations

besselfrac.h

```
void Bessel::besseljy(const Doub nu, const Doub x)
```

Sets jo, yo, jpo, and ypo respectively to the Bessel functions $J_\nu(x)$, $Y_\nu(x)$ and their derivatives $J'_\nu(x)$, $Y'_\nu(x)$, for positive x and for 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.

```
{
    const Int MAXIT=10000;
    const Doub EPS=numeric_limits<Doub>::epsilon();
    const Doub FPMIN=numeric_limits<Doub>::min()/EPS;
    const Doub XMIN=2.0, PI=3.141592653589793;
    Doub 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,xx;
    Int i,isign,l,nl;

    if (x <= 0.0 || nu < 0.0) throw("bad arguments in besseljy");
    nl=(x < XMIN ? Int(nu+0.5) : MAX(0,Int(nu-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 x $<$ XMIN, while it is chosen so that x is greater than the turning point for x $\geq$ XMIN.

```
    xmu=nu-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=nu*xi;                                      isign keeps track of sign changes in the de-
    if (h < FPMIN) h=FPMIN;                       nominator.
    b=xi2*nu;
    d=0.0;
    c=h;
    for (i=0;i<MAXIT;i++) {
        b += xi2;
        d=b-d;
        if (abs(d) < FPMIN) d=FPMIN;
        c=b-1.0/c;
        if (abs(c) < FPMIN) c=FPMIN;
        d=1.0/d;
        del=c*d;
        h=del*h;
        if (d < 0.0) isign = -isign;
        if (abs(del-1.0) <= EPS) break;
    }
    if (i >= MAXIT)
        throw("x too large in besseljy; try asymptotic expansion");
    rjl=isign*FPMIN;                           Initialize $J_\nu$ and $J'_\nu$ for downward recurrence.
    rjpl=h*rjl;
```

```
rjl1=rjl;                                Store values for later rescaling.
rjp1=rjpl;
fact=nu*xi;
for (l=nl-1;l>=0;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 $J_\mu$ and $J'_\mu$.
if (x < XMIN) {                          Use series.
    x2=0.5*x;
    pimu=PI*xmu;
    fact = (abs(pimu) < EPS ? 1.0 : pimu/sin(pimu));
    d = -log(x2);
    e=xmu*d;
    fact2 = (abs(e) < EPS ? 1.0 : sinh(e)/e);
    xx=8.0*SQR(xmu)-1.0;             Evaluates $\Gamma_1$ and $\Gamma_2$ by Chebyshev expansion for
    gam1=chebev(c1,NUSE1,xx);               $|x| \le 1/2$. Also returns $1/\Gamma(1+x)$ and
    gam2=chebev(c2,NUSE2,xx);               $1/\Gamma(1-x)$.
    gampl= gam2-xmu*gam1;
    gammi= gam2+xmu*gam1;                          Chebyshev evaluation of $\Gamma_1$ and $\Gamma_2$.
    ff=2.0/PI*fact*(gam1*cosh(e)+gam2*fact2*d);        $f_0$.
    e=exp(e);
    p=e/(gampl*PI);                      $p_0$.
    q=1.0/(e*PI*gammi);                  $q_0$.
    pimu2=0.5*pimu;
    fact3 = (abs(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 (abs(del) < (1.0+abs(sum))*EPS) break;
    }
    if (i > MAXIT) throw("bessy series failed to converge");
    rymu = -sum;
    ry1 = -sum1*xi2;
    rymup=xmu*xi*rymu-ry1;
    rjmu=w/(rymup-f*rymu);               Equation (6.6.13).
} 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=1;i<MAXIT;i++) {
            a += 2*i;
            bi += 2.0;
            dr=a*dr+br;
            di=a*di+bi;
            if (abs(dr)+abs(di) < FPMIN) dr=FPMIN;
            fact=a/(cr*cr+ci*ci);
            cr=br+cr*fact;
            ci=bi-ci*fact;
            if (abs(cr)+abs(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 (abs(dlr-1.0)+abs(dli) <= EPS) break;
        }
        if (i >= MAXIT) throw("cf2 failed in besseljy");
        gam=(p-f)/q;                          Equations (6.6.6) – (6.6.10).
        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;
    jo=rjl1*fact;                             Scale original J_ν and J'_ν.
    jpo=rjp1*fact;
    for (i=1;i<=nl;i++) {                     Upward recurrence of Y_ν.
        rytemp=(xmu+i)*xi2*ry1-rymu;
        rymu=ry1;
        ry1=rytemp;
    }
    yo=rymu;
    ypo=nu*xi*rymu-ry1;
    xjy = x;
    nujy = nu;
}
```

besselfrac.h

```
void Bessel::besselik(const Doub nu, const Doub x)
```

Sets io, ko, ipo, and kpo respectively to the Bessel functions $I_\nu(x)$, $K_\nu(x)$ and their derivatives $I'_\nu(x)$, $K'_\nu(x)$, for positive x and for xnu $= \nu \geq 0$. 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.

```
{
    const Int MAXIT=10000;
    const Doub EPS=numeric_limits<Doub>::epsilon();
    const Doub FPMIN=numeric_limits<Doub>::min()/EPS;
    const Doub XMIN=2.0, PI=3.141592653589793;
    Doub 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,xx;
    Int i,l,nl;
    if (x <= 0.0 || nu < 0.0) throw("bad arguments in besselik");
    nl=Int(nu+0.5);                                  nl is the number of downward re-
    xmu=nu-nl;                                          currences of the I's and upward
                                                        recurrences of K's. xmu lies be-
                                                        tween −1/2 and 1/2.
```

```
xmu2=xmu*xmu;
xi=1.0/x;
xi2=2.0*xi;
h=nu*xi;                                        Evaluate CF1 by modified Lentz's
if (h < FPMIN) h=FPMIN;                             method (§5.2).
b=xi2*nu;
d=0.0;
c=h;
for (i=0;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 (abs(del-1.0) <= EPS) break;
}
if (i >= MAXIT)
    throw("x too large in besselik; try asymptotic expansion");
ril=FPMIN;                                      Initialize $I_\nu$ and $I'_\nu$ for downward
ripl=h*ril;                                         recurrence.
ril1=ril;                                       Store values for later rescaling.
rip1=ripl;
fact=nu*xi;
for (l=nl-1;l >= 0;l--) {
    ritemp=fact*ril+ripl;
    fact -= xi;
    ripl=fact*ritemp+ril;
    ril=ritemp;
}
f=ripl/ril;                                     Now have unnormalized $I_\mu$ and $I'_\mu$.
if (x < XMIN) {                                 Use series.
    x2=0.5*x;
    pimu=PI*xmu;
    fact = (abs(pimu) < EPS ? 1.0 : pimu/sin(pimu));
    d = -log(x2);
    e=xmu*d;
    fact2 = (abs(e) < EPS ? 1.0 : sinh(e)/e);
    xx=8.0*SQR(xmu)-1.0;                        Chebyshev evaluation of $\Gamma_1$ and $\Gamma_2$.
    gam1=chebev(c1,NUSE1,xx);
    gam2=chebev(c2,NUSE2,xx);
    gampl= gam2-xmu*gam1;
    gammi= gam2+xmu*gam1;
    ff=fact*(gam1*cosh(e)+gam2*fact2*d);        $f_0$.
    sum=ff;
    e=exp(e);
    p=0.5*e/gampl;                              $p_0$.
    q=0.5/(e*gammi);                            $q_0$.
    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 (abs(del) < abs(sum)*EPS) break;
    }
    if (i > MAXIT) throw("bessk series failed to converge");
    rkmu=sum;
    rk1=sum1*xi2;
    } else {
```

Evaluate CF2 by Steed's algorithm (§5.2), which is OK because there can be no zero denominators.

```
        b=2.0*(1.0+x);
        d=1.0/b;
        h=delh=d;
        q1=0.0;
```

Initializations for recurrence (6.6.35).

```
        q2=1.0;
        a1=0.25-xmu2;
        q=c=a1;
```

First term in equation (6.6.34).

```
        a = -a1;
        s=1.0+q*delh;
        for (i=1;i<MAXIT;i++) {
            a -= 2*i;
            c = -a*c/(i+1.0);
            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 (abs(dels/s) <= EPS) break;
```

Need only test convergence of sum since CF2 itself converges more quickly.

```
        }
        if (i >= MAXIT) throw("besselik: failure to converge in cf2");
        h=a1*h;
        rkmu=sqrt(PI/(2.0*x))*exp(-x)/s;
```

Omit the factor $\exp(-x)$ to scale all the returned functions by $\exp(x)$ for $x \geq$ XMIN.

```
        rk1=rkmu*(xmu+x+0.5-h)*xi;
    }
    rkmup=xmu*xi*rkmu-rk1;
    rimu=xi/(f*rkmu-rkmup);
```

Get I_μ from Wronskian.

```
    io=(rimu*ril1)/ril;
```

Scale original I_ν and I'_ν.

```
    ipo=(rimu*rip1)/ril;
    for (i=1;i <= nl;i++) {
```

Upward recurrence of K_ν.

```
        rktemp=(xmu+i)*xi2*rk1+rkmu;
        rkmu=rk1;
        rk1=rktemp;
    }
    ko=rkmu;
    kpo=nu*xi*rkmu-rk1;
    xik = x;
    nuik = nu;
}
```