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), Chapter 9.

Hart, J.F., et al. 1968, *Computer Approximations* (New York: Wiley), §6.8, p. 141. [1]

6.6 Modified Bessel Functions of Integer Order

The modified Bessel functions $I_n(x)$ and $K_n(x)$ are equivalent to the usual Bessel functions J_n and Y_n evaluated for purely imaginary arguments. In detail, the relationship is

$$\begin{aligned} I_n(x) &= (-i)^n J_n(ix) \\ K_n(x) &= \frac{\pi}{2} i^{n+1}[J_n(ix) + iY_n(ix)] \end{aligned} \tag{6.6.1}$$

The particular choice of prefactor and of the linear combination of J_n and Y_n to form K_n are simply choices that make the functions real-valued for real arguments x.

For small arguments $x \ll n$, both $I_n(x)$ and $K_n(x)$ become, asymptotically, simple powers of their argument

$$\begin{aligned} I_n(x) &\approx \frac{1}{n!}\left(\frac{x}{2}\right)^n \qquad n \geq 0 \\ K_0(x) &\approx -\ln(x) \\ K_n(x) &\approx \frac{(n-1)!}{2}\left(\frac{x}{2}\right)^{-n} \qquad n > 0 \end{aligned} \tag{6.6.2}$$

These expressions are virtually identical to those for $J_n(x)$ and $Y_n(x)$ in this region, except for the factor of $-2/\pi$ difference between $Y_n(x)$ and $K_n(x)$. In the region $x \gg n$, however, the modified functions have quite different behavior than the Bessel functions,

$$\begin{aligned} I_n(x) &\approx \frac{1}{\sqrt{2\pi x}} \exp(x) \\ K_n(x) &\approx \frac{\pi}{\sqrt{2\pi x}} \exp(-x) \end{aligned} \tag{6.6.3}$$

The modified functions evidently have exponential rather than sinusoidal behavior for large arguments (see Figure 6.6.1). The smoothness of the modified Bessel functions, once the exponential factor is removed, makes a simple polynomial approximation of a few terms quite suitable for the functions I_0, I_1, K_0, and K_1. The following routines, based on polynomial coefficients given by Abramowitz and Stegun [1], evaluate these four functions, and will provide the basis for upward recursion for $n > 1$ when $x > n$.

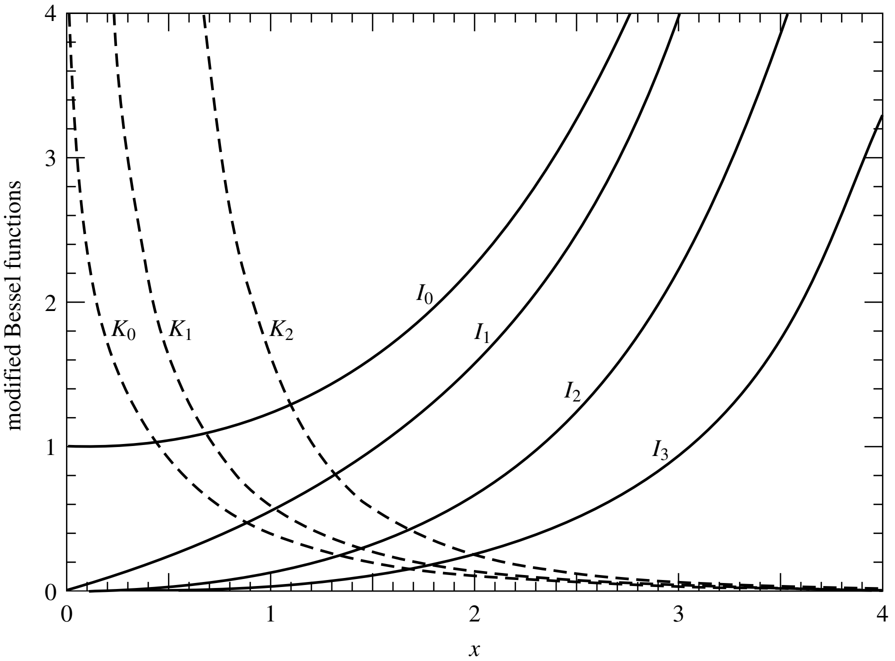

Figure 6.6.1. Modified Bessel functions $I_0(x)$ through $I_3(x)$, $K_0(x)$ through $K_2(x)$.

```
#include <math.h>

float bessi0(float x)
```
Returns the modified Bessel function I_0(x) for any real x.
```
{
    float ax,ans;
    double y;                           Accumulate polynomials in double precision.

    if ((ax=fabs(x)) < 3.75) {          Polynomial fit.
        y=x/3.75;
        y*=y;
        ans=1.0+y*(3.5156229+y*(3.0899424+y*(1.2067492
            +y*(0.2659732+y*(0.360768e-1+y*0.45813e-2)))));
    } else {
        y=3.75/ax;
        ans=(exp(ax)/sqrt(ax))*(0.39894228+y*(0.1328592e-1
            +y*(0.225319e-2+y*(-0.157565e-2+y*(0.916281e-2
            +y*(-0.2057706e-1+y*(0.2635537e-1+y*(-0.1647633e-1
            +y*0.392377e-2))))))));
    }
    return ans;
}
```

```
#include <math.h>

float bessk0(float x)
```
Returns the modified Bessel function K_0(x) for positive real x.
```
{
    float bessi0(float x);
    double y,ans;                 Accumulate polynomials in double precision.
```

```
    if (x <= 2.0) {                 Polynomial fit.
        y=x*x/4.0;
        ans=(-log(x/2.0)*bessi0(x))+(-0.57721566+y*(0.42278420
            +y*(0.23069756+y*(0.3488590e-1+y*(0.262698e-2
            +y*(0.10750e-3+y*0.74e-5))))));
    } else {
        y=2.0/x;
        ans=(exp(-x)/sqrt(x))*(1.25331414+y*(-0.7832358e-1
            +y*(0.2189568e-1+y*(-0.1062446e-1+y*(0.587872e-2
            +y*(-0.251540e-2+y*0.53208e-3))))));
    }
    return ans;
}
```

```
#include <math.h>

float bessi1(float x)
```
Returns the modified Bessel function $I_1(\mathtt{x})$ for any real `x`.
```
{
    float ax,ans;
    double y;                         Accumulate polynomials in double precision.

    if ((ax=fabs(x)) < 3.75) {        Polynomial fit.
        y=x/3.75;
        y*=y;
        ans=ax*(0.5+y*(0.87890594+y*(0.51498869+y*(0.15084934
            +y*(0.2658733e-1+y*(0.301532e-2+y*0.32411e-3))))));
    } else {
        y=3.75/ax;
        ans=0.2282967e-1+y*(-0.2895312e-1+y*(0.1787654e-1
            -y*0.420059e-2));
        ans=0.39894228+y*(-0.3988024e-1+y*(-0.362018e-2
            +y*(0.163801e-2+y*(-0.1031555e-1+y*ans))));
        ans *= (exp(ax)/sqrt(ax));
    }
    return x < 0.0 ? -ans : ans;
}
```

```
#include <math.h>

float bessk1(float x)
```
Returns the modified Bessel function $K_1(\mathtt{x})$ for positive real `x`.
```
{
    float bessi1(float x);
    double y,ans;                   Accumulate polynomials in double precision.

    if (x <= 2.0) {                 Polynomial fit.
        y=x*x/4.0;
        ans=(log(x/2.0)*bessi1(x))+(1.0/x)*(1.0+y*(0.15443144
            +y*(-0.67278579+y*(-0.18156897+y*(-0.1919402e-1
            +y*(-0.110404e-2+y*(-0.4686e-4)))))));
    } else {
        y=2.0/x;
        ans=(exp(-x)/sqrt(x))*(1.25331414+y*(0.23498619
            +y*(-0.3655620e-1+y*(0.1504268e-1+y*(-0.780353e-2
            +y*(0.325614e-2+y*(-0.68245e-3)))))));
    }
    return ans;
}
```

The recurrence relation for $I_n(x)$ and $K_n(x)$ is the same as that for $J_n(x)$ and $Y_n(x)$ provided that ix is substituted for x. This has the effect of changing a sign in the relation,

$$\begin{aligned} I_{n+1}(x) &= -\left(\frac{2n}{x}\right) I_n(x) + I_{n-1}(x) \\ K_{n+1}(x) &= +\left(\frac{2n}{x}\right) K_n(x) + K_{n-1}(x) \end{aligned} \tag{6.6.4}$$

These relations are always *unstable* for upward recurrence. For K_n, itself growing, this presents no problem. For I_n, however, the strategy of downward recursion is therefore required once again, and the starting point for the recursion may be chosen in the same manner as for the routine bessj. The only fundamental difference is that the normalization formula for $I_n(x)$ has an alternating minus sign in successive terms, which again arises from the substitution of ix for x in the formula used previously for J_n

$$1 = I_0(x) - 2I_2(x) + 2I_4(x) - 2I_6(x) + \cdots \tag{6.6.5}$$

In fact, we prefer simply to normalize with a call to bessi0.

With this simple modification, the recursion routines bessj and bessy become the new routines bessi and bessk:

```
float bessk(int n, float x)
Returns the modified Bessel function Kn(x) for positive x and n ≥ 2.
{
    float bessk0(float x);
    float bessk1(float x);
    void nrerror(char error_text[]);
    int j;
    float bk,bkm,bkp,tox;

    if (n < 2) nrerror("Index n less than 2 in bessk");
    tox=2.0/x;
    bkm=bessk0(x);                    Upward recurrence for all x...
    bk=bessk1(x);
    for (j=1;j<n;j++) {               ...and here it is.
        bkp=bkm+j*tox*bk;
        bkm=bk;
        bk=bkp;
    }
    return bk;
}
```

```
#include <math.h>
#define ACC 40.0                             Make larger to increase accuracy.
#define BIGNO 1.0e10
#define BIGNI 1.0e-10

float bessi(int n, float x)
Returns the modified Bessel function In(x) for any real x and n ≥ 2.
{
    float bessi0(float x);
    void nrerror(char error_text[]);
```

```
    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

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