

[Help](#)

```
#ifndef NUMERICS_H
#define NUMERICS_H

const double EULER = 0.57721566490153286;

template <class T> inline T PLUS(T X)
{
    return ((X) > 0 ? X : 0);
}
#define ISNUMBER(X) (((X)-(X))<1)

#include <cmath>
#include "
href../../common/math/progonka_h_src.pdfprogonka.h"

extern "C" {
#include "pnl/pnl_mathtools.h"
}

//Returns the incomplete gamma function P(a,x) (see NR)
double gammp(double a, double x);
//Returns the incomplete gamma function Q(a,x) = 1-P(a,x) (see NR)
double gammq(double a, double x);

//low-level routine for calculating P(a; x) and Q(a,x)
void gser(double *gamser, double a, double x, double *gln);
//low-level routine for calculating P(a,x) and Q(a,x)
void gcf(double *gamser, double a, double x, double *gln);

double normCDF(double x); // Standard gaussian CDF

double normPDF(double x); // Standard gaussian PDF

double expint(int n, double x);
//Evaluates the exponential integral E_n(x) (see Numerical Recipes)

double bessI1(double x);
//the modified Bessel function I_1(x) for any real x (see NR)
```

```

double bessk1(double x);
//the modified Bessel function  $K_1(x)$  for positive real x (see NR)

void polint(double xa[], double ya[], int n, double x, double *y, double *dy);
//Given arrays xa[1..n] and ya[1..n], and given a value x, this routine returns
//an error estimate dy. If  $P(x)$  is the polynomial of degree  $N - 1$  such that  $P(x_i) = y_i$ 
//for  $i = 1, \dots, n$ , then the returned value  $y = P(x)$ .

//Quadratures-----

template<class T>
double trapzd(T func, double a, double b, int n)
//This routine computes the nth stage of refinement of an extended trapezoidal rule
//func is input as a pointer to the function to be integrated between limits a and b
//When called with n=1, the routine returns the crudest estimate of  $\int_a^b f(x) dx$ 
//Subsequent calls with n=2,3,...
//(in that sequential order) will improve the accuracy by adding  $2n-2$  additional points
{
    double x, tnm, sum, del;
    static double s;
    int it, j;
    if (n == 1)
    {
        return (s = 0.5 * (b - a) * (func(a) + func(b)));
    }
    else
    {
        for (it = 1, j = 1; j < n - 1; j++) it <= 1;
        tnm = it;
        del = (b - a) / tnm; //This is the spacing of the points to be added.
        x = a + 0.5 * del;
        for (sum = 0.0, j = 1; j <= it; j++, x += del) sum += func(x);
        s = 0.5 * (s + (b - a) * sum / tnm); //This replaces s by its refined value
        return s;
    }
}

template<class T>
double gromb(T func, double a, double b)
//Returns the integral of the function func from a to b. Integration is performed using the

```

```

//method of order 2K, where, e.g., K=2 is Simpson's rule.
{
    const double EPS = 1.0e-8;
    const int JMAX1 = 50;
    const int JMAXP1 = JMAX1 + 1;
    const int K = 5;

    //Here EPS is the fractional accuracy desired, as determined by the extrapolat
    //JMAX limits the total number of steps; K is the number of points used in the

    double ss, dss;
    double s[JMAXP1], h[JMAXP1 + 1]; //These store the successive trapezoidal appr
    //and their relative stepsizes.
    int j;
    h[1] = 1.0;
    for (j = 1; j <= JMAX1; j++)
    {
        s[j] = trapzd(func, a, b, j);
        if (j >= K)
        {
            polint(&h[j - K], &s[j - K], K, 0.0, &ss, &dss);
            if (fabs(dss) <= EPS * fabs(ss)) return ss;
        }
        h[j + 1] = 0.25 * h[j];
    }
    myerror("Too many steps in routine qromb");
    return 0.0;    //Never get here.
}

#endif

```