Help

```
#include <cmath>
#include <iostream>
#include "config.h"

#include "numerics.h"

const double SQ2P = sqrt(2.0*M_PI);
const double FPMIN = 1.0e-100;
const double EPS=0.00000001;
const double ITMAX=100;

double gammp(double a, double x){
  //Returns P(a,x)
  double gamser, gammcf, gln;
  if (x < 0.0 || a <= 0.0) myerror("Invalid arguments in ro
    utine gammp");
  if (x < (a + 1.0)) {
    gser(&gamser, a, x, &gln);
    return gamser;
  } else {
    gcf(&gammcf, a, x, &gln);
    return 1.0 - gammcf;
  }
}

double gammq(double a, double x){
  //Returns Q(a,x)
  double gamser, gammcf, gln;
  if (x < 0.0 || a <= 0.0) myerror("Invalid arguments in ro
    utine gammq");
  if (x < (a + 1.0)) {
    gser(&gamser, a, x, &gln);
    return 1.0 - gamser;
  } else {
    gcf(&gammcf, a, x, &gln);
    return gammcf;
  }
}
```

```
void gser(double *gamser, double a, double x, double *gln)
//low-level routine for calculating Q(a; x)
{
  int n;
  double sum,del,ap;

  *gln=lgamma(a);
  if (x <= 0.0) {
    if (x < 0.0) myerror("x less than 0 in routine gser");
    *gamser=0.0;
    return;
  } else {
    ap=a;
    del=sum=1.0/a;
    for (n=1;n<=ITMAX;n++) {
      ++ap;
      del *= x/ap;
      sum += del;
      if (fabs(del) < fabs(sum)*EPS) {
  *gamser=sum*exp(-x+a*log(x)-(*gln));
  return;
      }
    }
    myerror("a too large, ITMAX too small in routine gser")
    ;
    return;
  }
}

void gcf(double *gammcf, double a, double x, double *gln) {
  //low-level routine for calculating Q(a; x)
  double an,b,c,d,del,h;
  *gln=lgamma(a);
  b=x+1.0-a;
  c=1.0/FPMIN;
  d=1.0/b;
  h=d;
  int i;
  for (i=1;i<=ITMAX;i++) {
    an = -i*(i-a);
```

```
    b += 2.0;
    d=an*d+b;
    if (std::fabs(d) < FPMIN) d=FPMIN;
    c=b+an/c;
    if (std::fabs(c) < FPMIN) c=FPMIN;
    d=1.0/d;
    del=d*c;
    h *= del;
    if (std::fabs(del-1.0) < EPS) break;
  }
  if (i > ITMAX)  myerror("a too large, ITMAX too small
    in routine gcf");
  *gammcf = std::exp(-x+a*std::log(x)-(*gln))*h;
}


double normPDF(double x) {
  return std::exp(-x*x/2)/SQ2P;
}

double normCDF(double x) {
  if(x > 0) return 1.0 - normCDF(-x);
  if(x * x < 3.0) return 0.5 * (1 - gammp(0.5, 0.5 * x * x)
    );
  else return 0.5 * gammq(0.5, 0.5 * x * x);
}


double expint(int n, double x) {
  int nm1;
  double a,b,c,d,del,fact,h,psi,ans=0;

  nm1=n-1;
  if (n<0 || x<0.0 || (x==0.0 && (n==0 || n==1)))
    myerror("bad argument in expint");
  else {
    if (n==0) ans=exp(-x)/x;  //special case
    else {
      if (x==0.0) ans=1.0/nm1; //another special case
      else {
  if (x>1.0) {
```

```
    b=x+n;
    c=1.0/FPMIN;
    d=1.0/b;
    h=d;
    for (int i=1; i<ITMAX; i++) {
      a = -i*(nm1+i);
      b += 2.0;
      d=1.0/(a*d+b);
      c=b+a/c;
      del=c*d;
      h *= del;
      if (fabs(del-1.0) < EPS) {
        ans = h*exp(-x);
        return ans;
      }
    }
    myerror("continued fraction failed in exprint");
  } else {
    ans = (nm1 != 0 ? 1.0/nm1 : -log(x)-EULER);
    fact=1.0;
    for (int i=1; i<= ITMAX; i++) {
      fact *= -x/i;
      if (i != nm1) del = -fact/(i-nm1);
      else {
        psi = -EULER;
        for (int ii=1; ii<=nm1; ii++) psi += 1.0/ii;
        del=fact*(-log(x)+psi);
      }
      ans += del;
      if (fabs(del) < fabs(ans)*EPS) return ans;
    }
    myerror("series failed in exprint");
  }
      }
    }
  }
  return ans;
}

double bessi1(double x){
  double 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*(0.15
    084934 + y*(0.2658733e-1
                      + y*(0.301532e-2 + y*0.32411
    e-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;
}

double bessk1(double x){
  double y,ans;  //Accumulate polynomials in double precisi
    on

  if(x <= 2.0){  //Polynomial fit
    y = x*x/4;
    ans = (log(x/2)*bessi1(x)) + (1./x)*(1+y*(0.15443144 +
    y*(-0.67278579
                    +y*(-0.18156897 + y*(-0.1919402
    e-1 + y*(-0.110404e-2 + y*(-0.4686e-4)))))));
  } else{
    y = 2./x;
    ans = (exp(-x)/sqrt(x))*(1.25331414+y*(0.23498619 + y*(
    -0.3655620e-1 + y*(0.1504268e-1
                        +y*(-0.780353e-2+y*(0.3256
    14e-2+y*(-0.68245e-3)))))));
  }
  return ans;

}
```

```
//-------------------------------------------------------
    ----------------

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 a value y, and
//an error estimate dy. If P(x) is the polynomial of degre
    e N ? 1 such that P(xai) = yai, i =
//1, . . . , n, then the returned value y = P(x).
{
  int ns=1;
  double den,dif,dift,ho,hp,w;
  double* c = new double[n];
  double* d = new double[n];
  c--; //for c and range between 1 and n
  d--; //instead of 0 and n-1
  dif=fabs(x-xa[1]);
  for (int i=1;i<=n;i++) { //Here we ?nd the index ns of th
    e closest table entry,
    if ( (dift=fabs(x-xa[i])) < dif) {
      ns=i;
      dif=dift;
    }
    c[i]=ya[i]; //and initialize the tableau of c's and d'
    s.
    d[i]=ya[i];
  }
  *y=ya[ns--]; //This is the initial approximation to y.
  for (int m=1;m<n;m++) { //For each column of the tableau,
    for (int i=1;i<=n-m;i++) { //we loop over the current
    c's and d's and update them.
      ho=xa[i]-x;
      hp=xa[i+m]-x;
      w=c[i+1]-d[i];
      if ( (den=ho-hp) == 0.0) myerror("Error in routine po
    lint");
      //This error can occur only if two input xa's are (
    to within roundoff) identical.
      den=w/den;
```

```
      d[i]=hp*den; //Here the c's and d's are updated.
      c[i]=ho*den;
    }
    *y += (*dy=(2*ns < (n-m) ? c[ns+1] : d[ns--]));
    //After each column in the tableau is completed, we de
    cide which correction, c or d,
    //we want to add to our accumulating value of y, i.e.,
    which path to take through the
    //tableau-forking up or down. We do this in such a way
    as to take the most "straight
    //line" route through the tableau to its apex, updatin
    g ns accordingly to keep track of
    //where we are. This route keeps the partial approxima
    tions centered (insofar as possible)
    //on the target x. The last dy added is thus the error
    indication.
  }
  c++;
  d++;
  delete[] d;
  delete[] c;
}
```

References