Help

```
/*
 * Copyright (C) 1998, 1999 John Smith
 *
 * This file is part of Octave.
 * Or it might be one day....
 *
 * Octave is free software; you can redistribute it and/or
   modify it
 * under the terms of the GNU General Public License as pub
   lished by the
 * Free Software Foundation; either version 2, or (at your
   option) any
 * later version.h *
 * Octave is distributed in the hope that it will be useful
   , but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERC
   HANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General
   Public License
 * for more details.
 *
 * You should have received a copy of the GNU General Pub
   lic License
 * along with Octave; see the file COPYING.  If not, write
   to the Free
 * Software Foundation, 59 Temple Place - Suite 330, Bosto
   n, MA  02111-1307, USA.
 *
 */

/*
 * Put together by John Smith john at arrows dot demon dot
   co dot uk,
 * using ideas by others.
 *
 * Calculate erf(z) for complex z.
 * Three methods are implemented; which one is used depend
   s on z.
 *
 * The code includes some hard coded constants that are
```

```
   intended to
 * give about 14 decimal places of accuracy. This is appro
   priate for
 * 64-bit floating point numbers.
 *
 * Oct 1999: Fixed a typo that in
 *     const double complex cerf_continued_fraction( const
   double complex z )
 * that caused erroneous answers for erf(z) where real(z)
   negative
 *
 */

/*
 * Put in Premia by Jérôme Lelong
 * used by Parisian Options
 */

#include <cmath>
#include <complex>

using namespace std;

typedef complex<double> complex_double ;

#ifndef M_SQRT2
#define M_SQRT2     1.41421356237309504880168872420969808
     /* sqrt(2) */
#endif

#ifndef M_PI
#define M_PI        3.14159265358979323846264338327950288
     /* pi */
#endif


/*#include <octave/oct.h>*/

/*#include "f77-fcn.h"*/

/*
```

```
 * Abramowitz and Stegun: (eqn: 7.1.14) gives this continued
 * fraction for erfc(z)
 *
 * erfc(z) = sqrt(pi).exp(-z^2).  1   1/2   1   3/2   2   5
    /2
 *                               ---  ---  ---  ---  ---  -
    -- ...
 *                               z +  z +  z +  z +  z +
    z +
 *
 * This is evaluated using Lentz's method, as described in
    the narative
 * of Numerical Recipes in C.
 *
 * The continued fraction is true providing real(z)>0. In
    practice we
 * like real(z) to be significantly greater than 0, say gre
    ater than 0.5.
 */

static complex_double cerfc_continued_fraction( complex_dou
    ble z )
{
  double tiny = 1e-20 ;      /* a small number, large enoug
    h to calculate 1/tiny */
  double eps = 1e-15 ;       /* large enough so that 1.0+ep
    s > 1.0, when using */
  /* the floating point arithmetic */
  /*
   * first calculate z+ 1/2   1
   *                   ---  --- ...
   *                  z +  z +
   */
  complex_double f = z ;
  complex_double C = f ;
  complex_double D = 0.0 ;
  complex_double delta ;
  double a ;

  a = 0.0 ;
  do
```

```
    {
      a = a + 0.5 ;
      D = z + a*D ;
      C = z + a/C ;

      if (D.real() == 0.0 && D.imag() ==  0.0)
        D = tiny ;

      D = 1.0 / D ;

      delta = (C * D) ;

      f = f * delta ;

    } while (abs(1.0-delta) > eps ) ;

  /*
   * Do the first term of the continued fraction
   */
  f = 1.0 / f ;

  /*
   * and do the final scaling
   */
  f = f * exp(-z*z)/ sqrt(M_PI) ;

  return f ;
}

static complex_double cerf_continued_fraction( complex_dou
    ble z )
{
  if (z.real() > 0)
    return 1.0 - cerfc_continued_fraction( z ) ;
  else
    return -1.0 + cerfc_continued_fraction( -z ) ;
}

/*
 * Abramawitz and Stegun, Eqn. 7.1.5 gives a series for erf
    (z)
```

```
 * good for all z, but converges faster for smallish abs(z)
    , say abs(z)<2.
 */

static complex_double cerf_series( complex_double z )
{
  double tiny = 1e-20 ;         /* a small number compared
    with 1.*/
  /* warning("cerf_series:");*/
  complex_double sum = 0.0 ;
  complex_double term = z ;
  complex_double z2 = z*z ;
  int n;
  for (n=0; n<3 || abs(term) > abs(sum)*tiny; n++)
    {
      sum = sum + term / (2.0*n+1.0) ;
      term = -term * z2 / (n+1.0) ;
    }

  return sum * 2.0 / sqrt(M_PI) ;
}

/*
 * Numerical Recipes quotes a formula due to Rybicki for ev
    aluating
 * Dawson's Integral:
 *
 * exp(-x^2) integral  exp(t^2).dt = 1/sqrt(pi) lim   sum
    exp(-(z-n.h)^2) / n
 *           0 to x                            h->0 n odd
 *
 * This can be adapted to erf(z).
 */
static complex_double cerf_rybicki( complex_double z )
{
  /* warning("cerf_rybicki:"); */
  double h = 0.2 ;          /* numerical experiment suggests
     this is small enough */
  complex_double I = complex_double(0.0,1.0);
  /*
   * choose an even n0, and then shift z->z-n0.h and n->n-
```

```
    h.
   * n0 is chosen so that real((z-n0.h)^2) is as small as
    possible.
   */

  int n0 = 2*(int) (floor( z.imag()/(2.0*h) + 0.5 )) ;

  complex_double z0 = I* (double)n0 *h;
  complex_double zp = z-z0 ;
  complex_double sum = 0.0;
  /*
   * limits of sum chosen so that the end sums of the sum
    are
   * fairly small. In this case exp(-(35.h)^2)=5e-22
   *
   */
  int np;
  for (np=-35; np<=35; np+=2)
    {
      complex_double t = zp -np*h*I ;
      complex_double b = exp(t*t) / (double)(np+n0);
      sum += b ;
    }

  sum = sum * 2.0 * exp(-z*z) / M_PI ;

  return (I*sum);
}

static complex_double cerf( complex_double z )
{
  /* Use the method appropriate to size of z -
   * there probably ought to be an extra option for NaN z,
    or infinite z
   *
   */
  if (abs(z) < 2.0)
    return cerf_series( z ) ;
  else if (fabs(z.real()) < 0.5)
    return cerf_rybicki( z ) ;
  else
```

```
    return cerf_continued_fraction( z ) ;
}

complex_double normal_cerf ( complex_double  z)
{
  return (0.5 * (cerf(z/M_SQRT2) + 1.0));
}
```

References