A SHORT NOTE ON THE NUMERICAL APPROXIMATION OF THE STANDARD NORMAL CUMULATIVE DISTRIBUTION AND ITS INVERSE

by

Chokri Dridi

REAL 03-T-7 March, 2003

A SHORT NOTE ON THE NUMERICAL APPROXIMATION OF THE STANDARD NORMAL CUMULATIVE DISTRIBUTION AND ITS INVERSE[§]

(March, 2003)

Chokri Dridi
Department of Agriculture and Consumer Economics, Regional Economics Applications Laboratory, University of Illinois at Urbana-Champaign.
cdridi@uiuc.edu

ABSTARCT: We provide computer codes in ANSI-C and Python for a fast and accurate computation of the cumulative distribution function (cdf) of the standard normal distribution and the inverse cdf of the same function. For the cdf we use the 5th order Gauss-Legendre quadrature which gives more accurate results compared to Excel and Matlab. The Inverse cdf computed using rational fraction approximations gives a result that is seven-decimal place accurate.

Keywords: C/C++, Gauss-Legendre Quadarture, Normal distribution, Numerical Integration, Rational fraction approximations, Software.
JEL Classification: C63, C88, C89

1. INTRODUCTION

Various research projects require building software components from the ground up. If the integration and calls to functions in commercial software are impossible then unless large financial resources are committed for the development of solutions similar to the ones used in commercial software, code components for the project have to be built in-house with the explicit purpose of having a reasonable accuracy, speed, and low development costs.

Numerical integration is certainly one of the most used techniques for running simulations and statistical analysis especially that more and more researchers in economics and other areas of social sciences started tackling problems that cannot be solved in closed form, Geweke (1995) gives examples in macroeconomics involving numerical integration. Popular integration methods range from very simple and inaccurate methods like rectangular and trapezoidal rules (Kreyszig, 1993) to more complex and relatively more accurate approaches like the Newton-Cotes formulae and Gaussian quadratures, surveyed in Press et al. (1992), and Judd (1998).

[§] All copyrighted material and trademarks cited are the property of their owners. Computer code included herewith is provided as is without any warranties.

In the next section, we introduce and evaluate the results of two numerical integration methods: the rational fraction approximations, and the composite Gauss-Legendre quadrature. In section 3, the computation of the inverse cdf using rational fraction approximations is evaluated. Section 4, concludes this short note, and the appendix provides commented computer code in Python and ANSI-C.

We compare results provided by computer programs written in Python and ANSI-C against results provided by built-in functions in Microsoft's Excel, software usually used by students for quick calculations, and Mathworks' Matlab, a more computation oriented software used by researchers.

2. Numerical Approximation of the Normal Standard Cumulative Function

Most commercial statistical and mathematical applications include the error function and its complement in their set of built-in functions, however many powerful programming environment such as C/C++ and Python do not recognize such functions. The error function is useful for most statistical and econometric purpose as it allows deriving the cumulative normal distribution (cdf) easily and accurately. The error function and its complement are defined as (Kennedy & Gentle, 1980):

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp\left(-t^2\right) dt \qquad ;\forall x \geq 0 \tag{1}$$

$$\begin{aligned} \operatorname{erfc}(x) &= \frac{2}{\sqrt{\pi}} \int_x^\infty \exp\left(-t^2\right) dt \\ &= 1 - \operatorname{erf}(x) \end{aligned} \qquad ;\forall x < 0 \tag{2}$$

Form (1) and (2), the exact standard normal cumulative distribution function is given by:

$$F(x) = \begin{cases} \dfrac{1 + \operatorname{erf}\left(x/\sqrt{2}\right)}{2} & ;x \geq 0 \\ \dfrac{1 - \operatorname{erf}\left(-x/\sqrt{2}\right)}{2} & ;x < 0 \end{cases} \tag{3}$$

Obviously, the accuracy of the results in (3) depends on the integration method used to evaluate the integral function in the error function and on machine round-off errors. Most integration method perform rather well on regions far from the tails of the distribution however, as noted by Greene (2000, p.177), the tail areas of the normal distribution are of importance for econometricians. In what follows, we examine two

different approximation techniques, rational fraction approximations, and Gauss-Legendre quadrature, to illustrate the trade-off between speed and accuracy.

2.1. RATIONAL FRACTION APPROXIMATIONS

Cody (1969) provides a rational fraction approximations that is relatively accurate for the time the article was published, the error function is defined by:

$$\operatorname{erf}(x) = xR_1(x) \qquad ;\forall 0 < x \le 0.5 \tag{4}$$

$$\operatorname{erfc}(x) = \exp\left(-x^2\right)R_2(x) \qquad ;\forall 0.46875 \le x \le 4.0 \tag{5}$$

$$\operatorname{erfc}(x) = \frac{\exp\left(-x^2\right)}{x}\left(\frac{1}{\sqrt{\pi}} + x^{-2}R_3\left(x^{-2}\right)\right) \qquad ;\forall x \ge 4.0 \tag{6}$$

In (4)-(6), the rational fractions are defined by (7) where the coefficients p_i and q_i are provided by Kennedy & Gentle (1980, p. 91-92):

$$R_1(x) = \frac{\sum_{i=0}^{3} p_i x^{2i}}{\sum_{i=0}^{3} q_i x^{2i}}; \qquad R_2(x) = \frac{\sum_{i=0}^{7} p_i x^{i}}{\sum_{i=0}^{7} q_i x^{i}}; \qquad R_3(x) = \frac{\sum_{i=0}^{4} p_i x^{-2i}}{\sum_{i=0}^{4} q_i x^{-2i}} \tag{7}$$

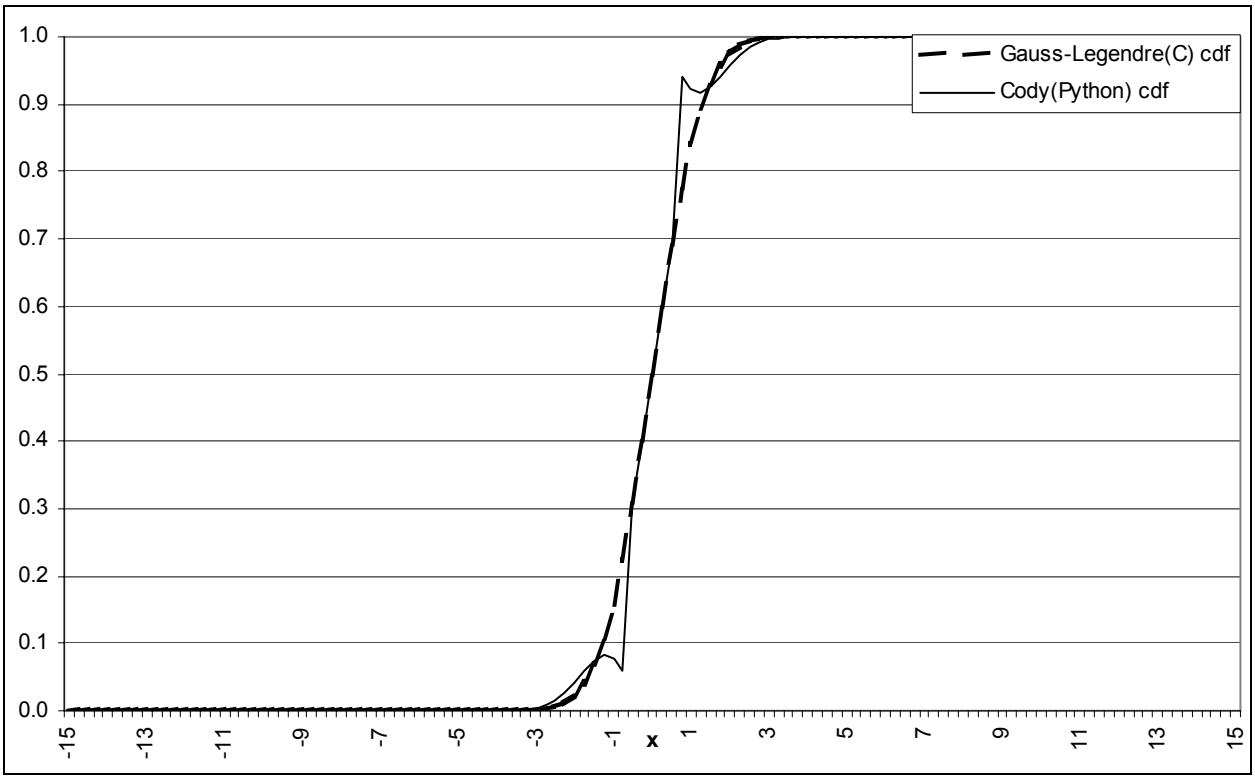

FIGURE 1: Gauss-Legendre quadrature vs. Rational fraction approximations before adjustment

Using the algorithm described above with Python[1] (appendix A.3) gives accurate enough results only for $x \in [-0.5, 0.75]$. Figure 1 compares the cdf given by Cody's algorithm and the Gauss-Legendre quadrature that we examine below, it is clear that Cody's algorithm cannot be of use if we seek accuracy. If we alter Cody's algorithm as follows we obtain a better precision without affecting its speed:

$$\operatorname{erf}(x) = xR_1(x) \qquad ;\forall 0 < x \le 1.5 \tag{8}$$

$$\operatorname{erfc}(x) = \exp(-x^2)\left(0.5R_1(x^2) + 0.2R_2(x^2) + 0.3R_3(x^2)\right) \qquad ;\forall 0.46875 \le x \le 4.0 \tag{9}$$

The results of the adjusted algorithm match the overall shape of the cumulative distribution function however, like with Matlab, it reaches zero (resp. 1) at $x \simeq -8.2931$ (resp. $x \simeq 8.2931$), which creates a problem if we need to divide by (resp. log) the probability. We will see next that the adjusted Cody algorithm while fast is less precise than the Composite Gauss-Legendre quadrature (figures 2 and 3).

2.2. COMPOSITE GAUSS-LEGENDRE QUADRATURE

The Gauss quadrature, consists in approximating the integral $I = \int_a^b w(x) f(x) dx$ by the summation $\sum_{k=1}^{K} w_i f(x_i)$ where: $f : \mathbb{R} \to \mathbb{R}$, $w(x)$ a weight function, the $x_i \in [a,b]$ are roots of Legendre polynomials of order *K* that are orthogonal to $w(x)$, and the w_i's are the roots' weights. This approach has been used with various weight functions, the simplest and yet very accurate one is attributed to Legendre, see Press et al. (1992) for more details regarding the methodology and other versions of the Gauss quadrature. The integration method known as Gauss-Legendre quadrature is Gauss's quadrature with $w(x) = 1; \forall x \in [-1,1]$, unlike Simpson's method and Newton-Cotes formulae, that use arbitrary and equally spaced points, the Gaussian quadrature determines precise points in $[a,b]$ symmetrically around zero, but not necessarly equally spaced, therefore it is not appropriate for tabulated data.

Press et al. (1992), offer additional explanations and a computer routine that helps finding Legendre polynomials roots and their respective weights, the polynomials are defined by:

$$P_0(x) = 1; \qquad P_1(x) = x; \qquad (k+1)P_{k+1} = (2k+1)xP_k - kP_{k-1}; \qquad \forall k \ge 1 \tag{10}$$

[1] The Python code available in appendix requires Numeric-22.0, the Numerical Extension to Python, written by Paul F. Dubois.

The following theorem announces a property of Gauss-Legendre quadrature, relating the order of the quadrature and the shape of the integrand to the precision of the approximation.

> **THEOREM:** The Gauss-Legendre quadrature is exact if $f(x)$ is a polynomial of degree less than or equal to $2n-1$, where n is Legendre polynomial number of roots.

Recall that the Gauss-Legendre quadrature applies only over the interval $[-1,1]$, in Harris & Stoker (1998, p. 571), for any chosen interval $[a,b]$, the integral $\int_a^b f(x)dx$ is rewritten with the substitution $x = \frac{b-a}{2}z + \frac{b+a}{2}$ to give:

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}z + \frac{b+a}{2}\right)dz \tag{11}$$

With *k* being the order of Lagrange polynomial (i.e. number of points), the error expected from the Gauss-Legendre quadrature after the change of variable is (Chapra & Canale, 1998):

$$E_k = \frac{2^{2k+1}(k!)^4}{(2k+1)\left[(2k)!\right]^3} f^{(2k)}(\xi) \qquad ; -1 \le \xi \le 1 \tag{12}$$

Limiting ourselves to the fifth order of Legendre polynomial, we approximate the normal cdf, using the code in Python (appendix A.2); the error in (12) becomes very small and is approximately equal to $\frac{f^{(10)}(\xi)}{1.23\text{E+}09}; \forall \xi \in [-1,1]$. While the algorithm gives accurate results it is conspicuously slow[2], this requires using a faster programming language like ANSI-C (appendix A.1). The advantage of the composite Gauss-Legendre quadrature is that it is precise and does not converge to zero or one over a large interval; in our test, we use the [-15.0,15.0] interval with 0.25 step size. The analysis of the error on the Gauss-Legendre quadarature shows that the maximum error compared to results from Matlab and Excel is 1.0E-4 (figure 3), and that it is always less than the absolute error between Cody's adjusted algorithm and Matlab.

[2] On a Pentium III 450 Mhz.

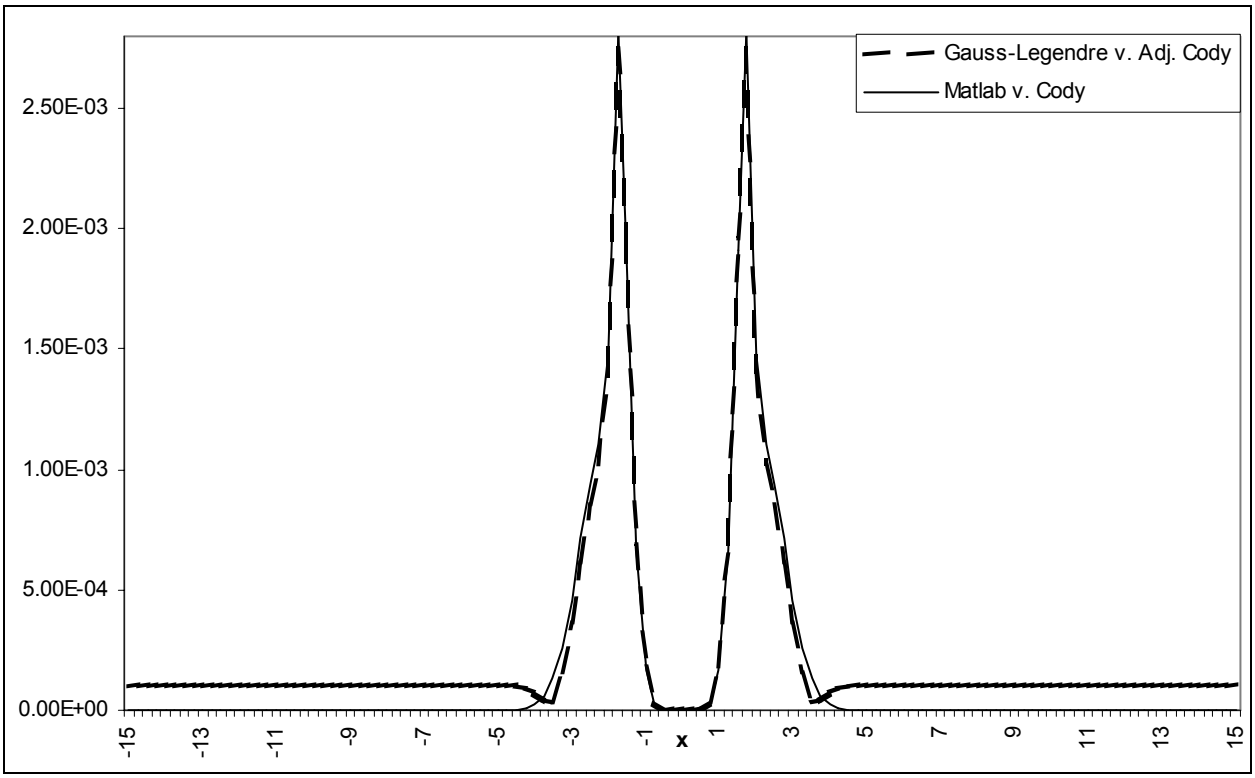

FIGURE 2: Absolute error between Gauss-Legendre quadrature, Matlab, and Rational fractions approximation after adjustment

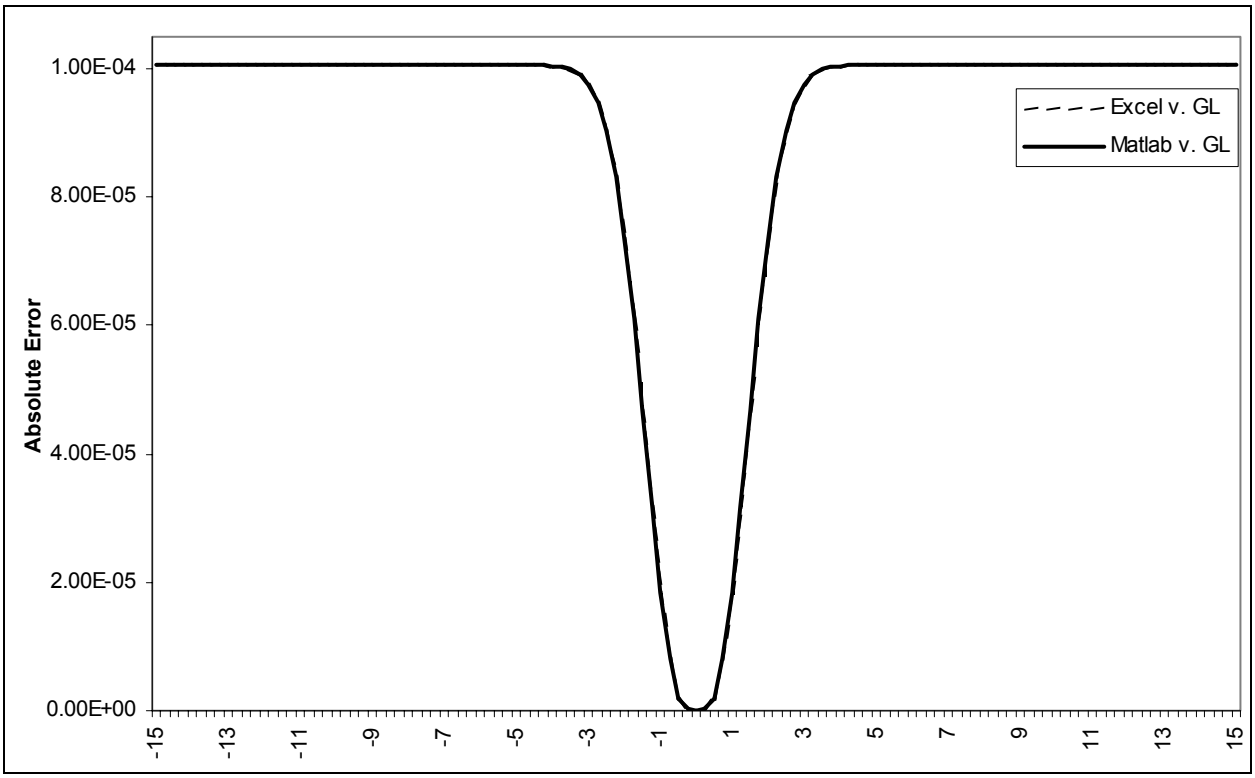

FIGURE 3: Absolute error of the Gauss-Legendre quadarture compared to Excel and Matlab

3. Inverse of Standard Normal Cumulative Distribution

With $F(.)$ being the normal cumulative distribution function (cdf), given a probability p, we seek to approximate the point x such that:

$$F(x) = p \Leftrightarrow x = F^{-1}(p) \qquad ; \forall (p, x) \in [0,1] \times \mathbb{R} \tag{13}$$

However, the inverse function of $F(.)$ is not possible to obtain in a closed form, various approximation methods were suggested. We test the algorithm developed by Odeh & Evans (1974) and described in Kennedy & Gentle (1980, p. 93-95), the algorithm is based on the approximation of rational fractions derived from Taylor series. Compared to results given by Matlab and Microsoft Excel in table 1, the algorithm written in Python is seven-decimal-place accurate, which is an appropriate approximation considering that $x \in \mathbb{R}$.

Table 1: Evaluation of the results of the inverse cdf approximation

p	MS-Excel	Matlab	Approx.
0	#NUM!	-Inf	-1.00E+20
0.1	-1.28155079437419	-1.2815515655446	-1.281551560901960
0.2	-0.84162138591637	-0.8416212335729	-0.841621221601852
0.3	-0.52440100262174	-0.5244005127080	-0.524400523866581
0.4	-0.25334657038911	-0.2533471031358	-0.253347106157011
0.5	0.00000000000000	0.0000000000000	0.000000000000000
0.6	0.25334657038911	0.2533471031358	0.253347106157011
0.7	0.52440100262174	0.5244005127080	0.524400523866581
0.8	0.84162138591637	0.8416212335729	0.841621221601852
0.9	1.28155079437419	1.2815515655446	1.281551560901960
1	#NUM!	+Inf	1.00E+20

4. Conclusion

Various research problems arise when a model cannot be solved in a closed form, in addition to simulating the model instead of solving it, researchers can analyze the problem numerically. In this short note, we provide explanations and computer code to compute the cumulative standard normal distribution function and its inverse; we retain results form the 5th order composite Gauss-Legendre quadrature in ANSI-C and the rational fraction approximations in Python as precise and fast numerical approximations. The composite Gauss-Legendre quadrature may be improved upon by using more points for better precision, and fewer subintervals for more speed. Additional points and their weight can be computed (Press et al., 1992 and Vetterling et al., 1992) and found in various references (Judd, 1998).

REFERENCES

Chapra, S. C. & R. P. Canale (1998): *Numerical Methods for Engineers: With Programming and Software Applications*, 3rd ed., McGraw-Hill, Boston.

Cody, W. J. (1969): "Rational Chebyshev Approximations for the Error function, *Mathematical Computation* 23, 631-638.

Geweke, J. (1995): "Monte Carlo Simulation and Numerical Integration", *Federal Reserve Bank of Minneapolis, Research Department Staff Report* 192.

Greene, W. H. (2000): *Econometric Analysis*, 4th ed., Prentice Hall, New Jersey.

Harris, J. W. & H. Stocker (1998): *Handbook of Mathematics and Computational Science*, Springer-Verlag, New York.

Judd, K. L. (1998): *Numerical Methods in Economics*, MIT Press, Cambridge.

Kennedy, W. J. Jr. & J. E. Gentle (1980): *Statistical Computing*, Marcel Dekker, New York.

Kreyszig, E. (1993): *Advanced Engineering Mathematics*, 7th ed., John Wiley & Sons, New York.

Odeh, R. E. & J. O. Evans (1974): "Algorithm AS 70: Percentage Points of the Normal Distribution", *Applied Statistics* 23, 96-97.

Press, W. H., S. A. Teukolsky, W. T. Vetterlig, & B. P. Flannery (1992): *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed., Cambridge University Press, Cambridge.

Vetterling, W. T., S. A. Teukolsky, W. H. Press, & B. P. Flannery (1992): *Numerical Recipes: Examples Book (C)*, 2nd ed., Cambridge University Press, Cambridge.

APPENDIX

A1. ANSI-C CODE FOR THE CDF (CDF.C)

```
/*****************************************************************/
/* Purpose: Computes cdf of standard normal dist. using a composite  */
/*          fifth-order Gauss-Legendre quadrature                    */
/* Code by: Chokri Dridi (December, 2002)                            */
/*****************************************************************/

#include <math.h>
#include <stdio.h>

long double GL(long double, long double); /* integration over closed interval
*/
long double cdf (long double); /* cdf function */
long double f(long double); /* function to integrate */
double p=0.;

int main ()
{
   /* the main function to get the cdf is cdf() */
   /* the main() block is used just to generate values for testing */
   long double x=-15;
   while (x<=15.){
                  p=cdf(x);
                  printf("%.17e\n",p);
                  x=x+.25;
                 }
      return 0;
}

/* cdf function */
long double cdf(long double x){
                               if(x>=0.){
                                           return (1.+GL(0,x/sqrt(2.)))/2.;
                                         }
                               else {
                                      return (1.-GL(0,-x/sqrt(2.)))/2.;
                                    }
                              }

/* Integration on a closed interval */
long double GL(long double a, long double b)
{
   long double y1=0, y2=0, y3=0, y4=0, y5=0;

   long double x1=-sqrt(245.+14.*sqrt(70.))/21., x2=-sqrt(245. -
14.*sqrt(70.))/21.;
   long double x3=0, x4=-x2, x5=-x1;
   long double w1=(322.-13.*sqrt(70.))/900., w2=(322.+13.*sqrt(70.))/900.;
   long double w3=128./225.,w4=w2,w5=w1;
   int n=4800;
   long double i=0, s=0, h=(b-a)/n;
```

```
   for (i=0;i<=n;i++){
                      y1=h*x1/2.+(h+2.*(a+i*h))/2.;
                      y2=h*x2/2.+(h+2.*(a+i*h))/2.;
                      y3=h*x3/2.+(h+2.*(a+i*h))/2.;
                      y4=h*x4/2.+(h+2.*(a+i*h))/2.;
                      y5=h*x5/2.+(h+2.*(a+i*h))/2.;
                      s=s+h*(w1*f(y1)+w2*f(y2)+w3*f(y3)+w4*f(y4)+w5*f(y5))/2.;
                      }
      return s;
}

/* Function f, to integrate */
long double f(long double x){
                               return (2./sqrt(22./7.))*exp(-pow(x,2.));
                                }
```

A2. PYTHON CODE FOR THE CDF USING GAUSS-LEGENDRE QUADRATURE (CDF-GL.PY)

```
"""
Purpose: Computes cdf of standard normal dist. using a composite
         fifth-order Gauss-Legendre quadrature
Code by: Chokri Dridi (December, 2002)
"""

from Numeric import *

" cdf function "
def cdf(x):
    if x>=0.:
        return (1.+GL(0,x/sqrt(2.)))/2.
    else:
        return (1.-GL(0,-x/sqrt(2.)))/2.

" Integration on a closed interval "
def GL(a,b):
    y1=0.
    y2=0.
    y3=0.
    y4=0.
    y5=0.

    x1=-sqrt(245.+14.*sqrt(70.))/21.
    x2=-sqrt(245.-14.*sqrt(70.))/21.
    x3=0.
    x4=-x2
    x5=-x1

    w1=(322.-13.*sqrt(70.))/900.
    w2=(322.+13.*sqrt(70.))/900.
    w3=128./225.
    w4=w2
    w5=w1

    n=4800
    s=0.
```

```
    h=(b-a)/n

    for i in range(0,n,1):
        y1=h*x1/2.+(h+2.*(a+i*h))/2.
        y2=h*x2/2.+(h+2.*(a+i*h))/2.
        y3=h*x3/2.+(h+2.*(a+i*h))/2.
        y4=h*x4/2.+(h+2.*(a+i*h))/2.
        y5=h*x5/2.+(h+2.*(a+i*h))/2.
        s=s+h*(w1*f(y1)+w2*f(y2)+w3*f(y3)+w4*f(y4)+w5*f(y5))/2.;
    return s;

" Function f, to integrate "
def f(x):
    return (2./sqrt(22./7.))*exp(-x**2.);
```

A3. PYTHON CODE FOR THE CDF USING RATIONAL FRACTIONS APPROXIMATION (CDF.PY)

```
"""
Purpose: Algorithm to compute cdf of a Gaussian distribution
         The cdf is 0 for all x < -8.29314441 and is 1 for all x > 8.29314441
         Before adjustment, the accuracy of this algorithm is acceptable only
for points -0.5<=x<=0.75
Coded by: Chokri Dridi (December, 2002)
Based on: Kennedy & Gentle(1980): Statistical Computing, Marcel Dekker, p.
90-92
"""

from Numeric import *

def cdf(x):
    if x > 0.:
        y=x
    else:
        y=-x

    if y >= 0. and y <= 1.5:
        p=(1.+erf(y/sqrt(2.)))/2.
    if y > 1.5:
        p=1.-erfc(y/sqrt(2.))/2.
    if x > 0.:
        return p
    else:
        return 1.-p

def erf(x):
    " for 0<x<=0.5 "
    return x*R1(x)

def erfc(x):
    " for 0.46875<=x<=4. "
    if x > 0.46875 and x < 4.:
        return exp(-x**2.)*(0.5*R1(x**2.)+0.2*R2(x**2.)+0.3*R3(x**2.))
    if x >= 4.:
        " for x>=4. "
```

```
        return (exp(-x**2.)/x)*(1./sqrt(22./7.)+R3(x**-2.)/(x**2.))

def R1(x):
    N=0.
    D=0.
    p=[2.4266795523053175e2,2.1979261618294152e1,6.9963834886191355,-
3.5609843701815385e-2]
    q=[2.1505887586986120e2,9.1164905404514901e1,1.5082797630407787e1,1.]
    for i in range(0,3,1):
        N=N+p[i]*x**(2.*i)
        D=D+q[i]*x**(2.*i)
    return N/D

def R2(x):
    N=0.
    D=0.
    p=[3.004592610201616005e2,4.519189537118729422e2,3.3932081673434368 70e2,
       1.529892850469404039e2,4.316222722205673530e1,7.211758250883093659,
       5.641955174789739711e-1,-1.368648573827167067e-7]
    q=[3.004592609569832933e2,7.909509253278980272e2,
       9.313540948506096211e2,6.389802644656311665e2,

2.775854447439876434e2,7.700015293522947295e1,1.278272731962942351e1,1.]
    for i in range(0,7,1):
        N=N+p[i]*x**(-2.*i)
        D=D+q[i]*x**(-2.*i)
    return N/D

def R3(x):
    N=0.
    D=0.
    p=[-2.99610707703542174e-3,-4.94730910623250734e-2,
       -2.26956593539686930e-1,-2.78661308609647788e-1,-2.23192459734184686e-
2]
    q=[1.06209230528467918e-2,1.91308926107829841e-
1,1.05167510706793207,1.98733201817135256,1.]
    for i in range(0,4,1):
        N=N+p[i]*x**(-2.*i)
        D=D+q[i]*x**(-2.*i)
    return N/D
```

A4. PYTHON CODE FOR THE INVERSE CDF (INVCDF.PY)

```
"""
Purpose: Algorithm to compute inverse cdf of a Gaussian distribution
         for values of p; 1.0E-20<p<1
Coded by: Chokr Dridi (November, 2002)
Based on: Kennedy & Gentle(1980): Statistical Computing, Marcel Dekker, p.
93-95
"""

from Numeric import *

def invcdf(p):
```

```
    if p>0.5:
        return -inv(p)
    else:
        return inv(p)

def inv(p):
    xp=0.
    lim = 1.e-20
    p0 = -0.322232431088
    p1 = -1.0
    p2 = -0.342242088547
    p3 = -0.0204231210245
    p4 = -0.453642210148e-4
    q0 = 0.0993484626060
    q1 = 0.588581570495
    q2 = 0.531103462366
    q3 = 0.103537752850
    q4 = 0.38560700634e-2
    if p < lim or p == 1.:
        return -1./lim
    if p == 0.5:
        return 0.
    if p > 0.5:
        p=1.-p
    y = sqrt(log(1./p**2.))
    xp= y+((((y*p4+p3)*y+p2)*y+p1)*y+p0)/((((y*q4+q3)*y+q2)*y+q1)*y+q0)
    if p < 0.5:
        xp = -xp
    return xp
```