Journal of Statistical Software

*July 2004, Volume 11, Issue 4.* http://www.jstatsoft.org/

# Evaluating the Normal Distribution

**George Marsaglia**
Florida State University

**Abstract**

This article provides a little table-free C function that evaluates the normal distribution with absolute error less than $8 \times 10^{-16}$. A small extension provides relative error near the limit available in double precision: 14 to 16 digits, the limits determined mainly by the computer's ability to evaluate exp(-t) for large t. Results are compared with those provided by calls to erf or erfc functions, the best of which compare favorably, others do not, and all appear to be much more complicated than need be to get either absolute accuracy less than $10^{-15}$ or relative accuracy to the exp()-limited 14–16 digits. Also provided: A short history of the error function erf and its intended use, as well as, in the `'browse files'` attachment, various erf or erfc versions used for comparison.

*Keywords*: normal distribution, Phi, cPhi, error function, erf, erfc, accuracy.

## 1. Introduction

Let $\phi(t)$ be the standard normal density function, $\phi(t) = e^{-\frac{1}{2}t^2}/\sqrt{2\pi}$ and let $\Phi(x)$, Phi$(x)$ and cPhi$(x)$ be the corresponding distribution and complementary distribution functions:

$$\Phi(x) = \text{Phi}(x) = \int_{-\infty}^{x} \phi(t)\, dt, \qquad \text{cPhi}(x) = \int_{x}^{\infty} \phi(t)\, dt = 1 - \Phi(x).$$

I support the common, but unfortunately not universal, notation $\Phi(x)$ for the standard normal distribution when math fonts are available, or Phi(x) and cPhi(x) for $\Phi(x)$ and $1-\Phi(x)$ when referring to programming language functions.

This little C function:

```
 double Phi(double x)
{long double s=x,t=0,b=x,q=x*x,i=1;
 while(s!=t) s=(t=s)+(b*=q/(i+=2));
 return  .5+s*exp(-.5*q-.91893853320467274178L);}
```

produces the following output, with the true value displayed below the value provided by Phi(x):

```
x      Phi(x)              x      Phi(x)               x      Phi(x)
 0.1 .539827837277029      4.5 .999996602326875      -1.1 .1356660609463827
     .5398278372770289 81...    .9999966023268756 99...     .1356660609463826751...
 1.2 .884930329778292      5.6 .999999989282410      -3.3 .0004834241423840
     .8849303297782917 31...    .9999999892824097 41...     .0004834241423837772...
 2.3 .989275889978324      6.7 .999999999989579      -5.5 .0000000189895631
     .9892758899783241 94...    .9999999999895790 23...     .0000000189895624658...
 3.4 .999663070734323      7.8 .999999999999997      -7.7 .0000000000000065
     .9996630707343231 19...    .9999999999999690 46...     .0000000000000068033...
```

A study over the full range for Phi(x) shows a worst error of about $8 \times 10^{-16}$, but that is absolute, not relative error. Concerning relative error, here is a little, but not-quite-as-little C function that uses nine tabled values to provide tail values of the normal distribution, i.e. $\mathrm{cPhi}(x) = 1 - \Phi(x)$, to about the 14–16 digit limit available in double precision for a function that uses exp():

```
double cPhi(double x)
{int i,j=.5*(fabs(x)+1); long double R[9]=
 {1.25331413731550025L, .421369229288054473L, .236652382913560671L,
 .162377660896867462L, .123131963257932296L,.0990285964717319214L,
 .0827662865013691773L,.0710695805388521071L,.0622586659950261958L};
long double pwr=1,a=R[j],z=2*j,b=a*z-1,h=fabs(x)-z,s=a+h*b,t=a,q=h*h;
for(i=2;s!=t;i+=2){a=(a+z*b)/i; b=(b+z*a)/(i+1); pwr*=q; s=(t=s)+pwr*(a+h*b);}
s=s*exp(-.5*x*x-.91893853320467274178L);
if(x>=0) return (double) s; return (double) (1.-s);
}
```

Values for the standard normal distribution are often obtained by means of library functions `erf` or `erfc`, with obligatory changes of scale and argument, such as `.5+.5*erf(x/sqrt(2.))`, or by using functions based on erf or erfc, with adjustments to the many coefficients in the polynomial or rational function approximations used in most erf or erfc implementations.

We will compare accuracies of the two short C functions above with those obtained from various erf and erfc implementations. Absolute error will be displayed as $-\log_{10}|\text{tru-approx}|$, that is, the negative of the exponent in expressing the absolute error as a power of 10. Relative error will be displayed as $-\log_{10}|(\text{tru-approx})/\text{tru}|$.

Results concerning absolute or relative error can be misleading, or mistaken for "digits of accuracy". For example, the above Phi function returns Phi(7)=0.9999999999987196 (format 19.16f), compared to .99999999999872018746..., the true value, and to many users, those ten 9's would be considered significant. For true values near 1, absolute and relative error are much the same. But in some applications, particularly for solutions of diffusion or heat equations, the region of interest may not be the statistician's $0 < x < 5$, say, but rather $10 < x < 14$, where greater accuracy may be provided by means of the complementary normal function $\mathrm{cPhi}(x) = \int_x^\infty \phi(t)\,dt$. For that, use of 1-Phi($x$) will not do as well, so many users may want to have an alternative function such as `cPhi(x)` that provides greater relative accuracy.

Our approach is to have two functions available: Phi($x$) and cPhi($x$). They happen to use the same method, differing only in that Phi($x$) is simpler with no tables, a Taylor series about zero, while cPhi($x$) evaluates a Taylor series about $x = z + h$ for tabled values cPhi($z$), $z = 0, 2, 4, 6, \ldots, 16$. Some implementations that rely on a single function—for example, the function `pnorm(x)` mentioned below—will show a marked difference between `1-pnorm(x)` and `pnorm(-x)`, the latter providing much greater relative accuracy. See that discussion in Section 3 and its impact on interpretation of Figure 4.

Precision for the above cPhi function seems limited by the accuracy of the exp() value in the final step. Section 3 shows that the little Phi and cPhi functions provide results at least as good as—and often better—than those obtained by conversion of erf or erfc from C compiler libraries or their elaborate listings from internet sources.

For those who may want to assess the accuracy of the erf or erfc functions that they have access to, here are values returned by the above cPhi(x) function, using format 20.14e, followed by the true values expressed to that same format (obtained in Maple as .5*erfc(x/sqrt(2.)), using Digits:=30): Also listed are the values returned by two of what appear to be among the best available library or internet erfc functions, .5*erfc(x/sqrt(2.)) from Sun Microsystems (1993) and .5*derfc(x/sqrt(2.)) from Ooura (1998), (exponents removed from last two to save space):

| x | cPhi(x) | true | Sun erfc | Ooura derfc |
|---|---|---|---|---|
| 0.1 | 4.60172162722971e-01 | 4.60172162722971e-01 | 4.60172162722971 | 4.60172162722971 |
| 1.2 | 1.15069670221708e-01 | 1.15069670221708e-01 | 1.15069670221708 | 1.15069670221708 |
| 2.3 | 1.07241100216758e-02 | 1.07241100216758e-02 | 1.07241100216758 | 1.07241100216758 |
| 3.4 | 3.36929265676880e-04 | 3.36929265676881e-04 | 3.36929265676881 | 3.36929265676881 |
| 4.5 | 3.39767312473006e-06 | 3.39767312473006e-06 | 3.39767312473006 | 3.39767312473006 |
| 5.6 | 1.07175902583109e-08 | 1.07175902583109e-08 | 1.07175902583109 | 1.07175902583109 |
| 6.7 | 1.04209769879652e-11 | 1.04209769879652e-11 | 1.04209769879652 | 1.04209769879652 |
| 7.8 | 3.09535877195868e-15 | 3.09535877195870e-15 | 3.09535877195867 | 3.09535877195867 |
| 8.9 | 2.79233437493963e-19 | 2.79233437493966e-19 | 2.79233437493966 | 2.79233437493966 |
| 10.0 | 7.61985302416052e-24 | 7.61985302416053e-24 | 7.61985302416050 | 7.61985302416047 |
| 11.1 | 6.27219439321695e-29 | 6.27219439321703e-29 | 6.27219439321689 | 6.27219439321687 |
| 12.2 | 1.55411978638958e-34 | 1.55411978638959e-34 | 1.55411978638957 | 1.55411978638957 |
| 13.3 | 1.15734162836903e-40 | 1.15734162836904e-40 | 1.15734162836902 | 1.15734162836902 |
| 14.4 | 2.58717592540226e-47 | 2.58717592540226e-47 | 2.58717592540229 | 2.58717592540228 |
| 15.5 | 1.73446079179383e-54 | 1.73446079179387e-54 | 1.73446079179381 | 1.73446079179382 |
| 16.6 | 3.48454651995040e-62 | 3.48454651995041e-62 | 3.48454651995027 | 3.48454651995030 |

## 2. Evaluating cPhi($x$) $= \int_x^\infty \phi(t)\,dt$

With the normal density $\phi(x) = e^{-\frac{1}{2}x^2}/\sqrt{2\pi}$, define the function R($x$) by means of

$$\mathrm{R}(x)\phi(x) = \int_x^\infty \phi(t)\,dt, \text{ that is, } \mathrm{R}(x) = \mathrm{cPhi}(x)/\phi(x).$$

The ratio $\mathrm{R}(x) = \mathrm{cPhi}(x)/\phi(x), x \geq 0$ is a well behaved, convex function. The terms in its Taylor series may be easily developed by a 2-lag recursion. Furthermore, its odd derivatives all have the same sign, as do its even. It starts at $\mathrm{R}(0) = \sqrt{\pi/2}$ then drops steadily toward zero. The graph of $y = \mathrm{R}(x)$ looks much like that of $y = 2/(x+\sqrt{x^2+8/\pi})$, so much so that they are difficult to separate in a plot of this size, where both are plotted for $0 \leq x < 15$:

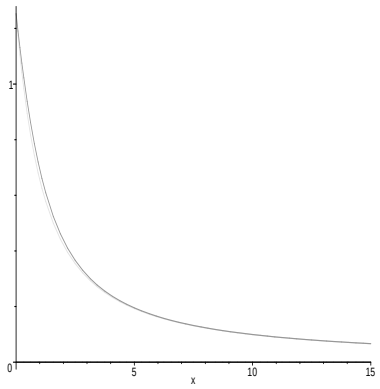



Figure 1: $\mathrm{R}(x)$ and $2/(x + \sqrt{x^2+8/\pi})$

To get the Taylor expansion of R, differentiate $\mathrm{R}(x)\phi(x) = \int_x^\infty \phi(t)\,dt$ on both sides to get

$$\mathrm{R}'(x)\phi(x) - x\mathrm{R}(x)\phi(x) = -\phi(x), \text{ that is, } \mathrm{R}' = x\mathrm{R} - 1.$$

Then

$$\mathrm{R}'' = x\mathrm{R}' + \mathrm{R}; \qquad \mathrm{R}''' = x\mathrm{R}'' + 2\mathrm{R}'; \qquad \mathrm{R}'''' = x\mathrm{R}''' + 3\mathrm{R}'';$$

and in general, if $\mathrm{R}^{[k]}$ means the $k$'th derivative of R, then for $k \geq 1$: $\mathrm{R}^{[k+1]} = x\mathrm{R}^{[k]} + k\mathrm{R}^{[k-1]}$.

Because $\mathrm{R}(x)$ has an easily-developed Taylor expansion, one can use a few exact values, say at 0,2,4,8,10,12,14,16, then use the Taylor expansion to get R at intermediate points. >From $\mathrm{R}(x)$ one easily obtains $\mathrm{cPhi}(x)$ by multiplication: $\mathrm{cPhi}(x) = \mathrm{R}(x)\phi(x)$. This leads to an easy way to evaluate $\mathrm{R}(z+h)$, given the exact value $a = \mathrm{R}(z)$, by summing the series two terms at a time until the new increment makes no change in the sum `s`. Using C expressions:

```
        t=a; b=z*a-1; pwr=1; s=a+h*b;
        for(i=2;s!=t;i+=2)
{a=(a+z*b)/i; b=(b+z*a)/(i+1); pwr=pwr*h*h; s=(t=s)+pwr*(a+h*b);}
```

With merely the values of $\mathrm{R}(z)$ for $z = 0, 2, 4, 6, 8, 10, 12, 14, 16$, we can use the Taylor series to evaluate $\mathrm{R}(z+h)$, with $|h| \leq 1$. This provides the basis for the C function $\mathrm{cPhi}(x)$ above.

It turns out that even the Taylor expansion for $\mathrm{R}(z+h)$ about $z = 0$ provides a simple and accurate way to evaluate $\mathrm{cPhi}(x) = 1 - \Phi(x)$ with absolute error $< 10^{-15}$, for $-7 < x < 7$, a range likely to cover most $x$'s encountered in probability and statistics. I pointed this out in response to a newsgroup query seeking methods for evaluating the normal distribution, displaying a table much like the one above for $\Phi(.1), \Phi(1.2), \Phi(2.3), \ldots, \Phi(6.7), \Phi(7.8)$, (see Newsgroup postings 2002, 2004). Several suggestions led to simplifications, particularly by Bill Daly, who pointed out that the Taylor series about zero provides even-order terms whose sum is $1/\phi(0)$, and odd-order terms with a simple recursion.

Rather than use simplifications of the Taylor expansion about zero for $\mathrm{R}(x)$, we might start with a different function, say $B(x) = \int_0^x \phi(t)\,dt/\phi(x)$, so that $B(x)\phi(x) = \int_0^x \phi(t)\,dt$. As with $\mathrm{R}(x)$ above, the recursion $B^{[k+1]}(x) = xB^{[k]}(x) + kB^{[k-1]}(x)$ readily follows, and in particular

for the Taylor series about zero:
$B(x) = x + x^3/3 + x^5/(3 \cdot 5) + x^7/(3 \cdot 5 \cdot 7) + \cdots$ and thus

$$\Phi(x) = \frac{1}{2} + \phi(x)\left(x + \frac{x^3}{3} + \frac{x^5}{3 \cdot 5} + \frac{x^7}{3 \cdot 5 \cdot 7} + \frac{x^9}{3 \cdot 5 \cdot 7 \cdot 9} + \cdots\right).$$

It is surprising that summing this series until the new term seems to vanish, as in our little C function:

```
double Phi(double x)
{long double s=x,t=0,b=x,q=x*x,i=1;
while(s!=t) s=(t=s)+(b*=q/(i+=2));
return  .5+s*exp(-.5*q-.91893853320467274178L);   }
```

provides greatest absolute error of about $8 \times 10^{-16}$, making it a method of choice for many applications in probability and statistics. The compact form resulted from another suggestion by Bill Daly in the newsgroups postings .

For this tiny Phi function, Figure 5 shows -$\log_{10}(|\text{tru} - \text{approx}|)$, the 'digits of accuracy' for absolute error, along with those for two popular but far more elaborate methods for evaluating the normal probability distribution, (descriptions below).

For applications requiring relative accuracy to the limit available in double precision, cPhi(x) above may be used. For either requirement: absolute- or relative-accuracy, Phi or cPhi provide values for the normal distribution function by simple and easily understood methods, avoiding the need for special function libraries with ponderous listings for erf or erfc, and the necessary argument change from x to x/sqrt(2).

The cPhi function expects, and returns, an ordinary double, but internal calculations may be made more accurate by use of C's long double, which for many implementations invokes an 80-bit floating point processor. One rarely needs values of the normal CDF $\Phi(x)$ or its complement cPhi($x$) accurate to as many as 15 digits, or for values $x$ beyond 6 or so, but an easily implemented Taylor series for $\mathrm{R}(x) = \text{cPhi}(x)/\phi(x)$ makes such accuracy available with only a few stored values of $\mathrm{R}(x)$ and few lines of code.

Since cPhi($x$) comes from the product of $\mathrm{R}(x)$ and the normal density $\phi(x)$, one would expect that the error could be no better for cPhi($x$) than for $\phi(x)$. As Figure 2 shows, the above C code evaluates $\mathrm{R}(x)$ more accurately than it can evaluate exp(-.5*x*x-.91893853320467274178L) (the constant is $\frac{1}{2}\ln(2\pi)$ ), and thus the digits-of-accuracy measure for the product $\mathrm{R}(x)\phi(x)$ is pretty much that of the exp() part. Occasionally, there is constructive cancellation and the accuracy of cPhi($x$) is better than that for evaluating the normal density. There, 'digits of accuracy' means $-\log_{10}(|\text{tru} - \text{approx}|/\text{tru})$. Both cPhi($x$) and $\phi(x)$ average 14.7 digits of accuracy; both lose around two digits of accuracy as x increases from 0 to 16. Sometimes cPhi is a little better, sometimes not. Since cPhi($x$) comes from $\mathrm{R}(z + h)$ with $z = 0, 2, 4, \ldots .16$, we can expect greater errors near 1,3,5,...15, when $h$ is near $\pm 1$. This only shows up near 9,11,13 and 15 in Figure 2; otherwise, errors in exp() seem to dominate.

## 3. The error function erf

In 1871, J.W. Glaisher published an article on definite integrals in which he comments that while there is scarcely a function that cannot be put in the form of a definite integral, for the evaluation of those that cannot be put in the form of a tolerable series we are limited to combinations of algebraic, circular, logarithmic and exponential—the elementary or primary functions. See Glaisher (1871). He writes:

> "The chief point of importance, therefore, is the choice of the elementary functions; and this is a work of some difficulty. One function however, viz. the integral $\int_x^\infty e^{-x^2}\,dx$, well known for its use in physics, is so obviously suitable for the purpose, that, with the exception of receiving a name and a fixed notation, it may almost be said to have already become primary. ... As it is necessary that the function should have a name, and as I do not know that any has been suggested, I propose to call it the *Error-function*, on account of its earliest and still most important use being in connexion with the theory of Probability, and notably with the theory of Errors, and to write
>
> $$\int_x^\infty e^{-x^2}\,dx = Erf\, x. \quad "$$

Glaisher goes on to demonstrate use of *Erf* in the evaluation of a variety of definite integrals. We still use 'error function' and Erf, but Erf has become erf, with a change of limits and a normalizing factor: $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt$, while Glaisher's original *Erf* has become $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}\,dt$. The normalizing factor $\frac{2}{\sqrt{\pi}}$ that makes $\mathrm{erfc}(0) = 1$ was not used in early editions of the famous "A Course in Modern Analysis" by Whittaker and Watson. Both were students and later colleagues of Glaisher, as were other eminences from Cambridge mathematics/physics: Maxwell, Thomson (Lord Kelvin), Rayleigh, Littlewood, Jeans, Whitehead and Russell. Glaisher had a long and distinguished career at Cambridge and was editor of *The Quarterly Journal of Mathematics* for fifty years, from 1878 until his death in 1928.

It is unfortunate that changes from Glaisher's original *Erf*: the switch of limits, names and the standardizing factor, did not apply to what Glaisher acknowledged was its most important application: the normal distribution function, and thus $\frac{1}{\sqrt{2\pi}}\int e^{-\frac{1}{2}t^2}\,dt$ did not become the basic integral form. So those of us interested in its most important application are stuck with conversions:

$$\Phi(x) = \mathrm{Phi}(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{1}{2}t^2}\,dt = \frac{1}{2}+\frac{1}{2}\mathrm{erf}(x/\sqrt{2}), \qquad 1-\Phi(x) = \mathrm{cPhi}(x) = \frac{1}{2}\mathrm{erfc}(x/\sqrt{2}).$$

A search of the internet will show many applications of what we now call erf or erfc to problems of the type that seemed of more interest to Glaisher and his famous colleagues: integral solutions of differential equations. These include the telegrapher's equation, studied by Lord Kelvin in connection with the Atlantic cable, and Kelvin's estimate of the age of the earth (25 million years), based on the solution of a heat equation for a molten sphere (it was far off because of then unknown contributions from radioactive decay). More recent internet mentions of the use of erf or erfc for solving differential equations include short-circuit power dissipation in electrical engineering, current as a function of time in a switching diode, thermal

spreading of impedance in electrical components, diffusion of a unidirectional magnetic field, recovery times of junction diodes and the Mars Orbiter Laser Altimeter.

## 4. Implementations and variations of erf and erfc

Although Glaisher advocated erf as a means to represent integrals that could not be put in the form of a tolerable series or combinations of algebraic, circular, logarithmic or exponential functions, Section 2 shows that $\int_0^x \phi(t)\,dt$, $\int_x^\infty \phi(t)\,dt$ and hence erf, can be represented by means of a quite tolerable Taylor series—times an exponential function. This invites comparison with—and possibly replacement for—some of the many available erf implementations, or variations of them tailored for evaluating the normal CDF $\Phi$.

Some of Newsgroup postings (2002) or Newsgroup postings (2004) point to Fortran or C versions of erf based on numerical approximations from Cody (1993), Hart *et al.* (1968) and Hill (1973). They are labelled 'Cody', 'Hart' and 'AS66' below. An erfc developed at Sun Microsystems (1993) ('Sun' in figures below) was also cited. The Sun erfc is available under GNU on linux systems. All of these are rather elaborate programs, often handling the range in several pieces.

The best C version of erfc that I have found is derfc.c by Ooura (1998). It uses a rational function times $e^{-x^2}$, with explicit coefficients (no tables) for the ratio of polynomials of degree 12 and 13. It too provides 14-16 digit accuracy; a plot of its 'digits of accuracy' vs. x in $0 < x < 16$ looks very much like that for cPhi(x) and for the Sun erfc, (Figure 3).

A listing at `http://tigger.smu.edu.sg/software/mnp-stuff/stat.ubc.ca/pnorms2.c` provides a C function called pnorm(x) based on Algorithm 715 of Cody (1993). That pnorm() provides absolute accuracy comparable to that of the little Phi(x) function above, but one encounters difficulty in using pnorm($x$) for large $x$. If used directly, $x$ values beyond 8.2 return zero, so it is not suitable for providing the complementary normal integral by means of 1-pnorm($x$). However, pnorm($-x$) works well. For example, the true value of cPhi(10) is 7.619853024160526066597334325145e-24. Invoking the above pnorm version of Cody's rational approximation,

1-pnorm(10.)=0.0000000000000000e+00, pnorm(-10)=7.6198530241605269e-24.

Several implementations seem to be based on the polynomial approximations Hart *et al.* (1968). Compiled and printed results show that, like the pnorm function based on Cody, relative accuracy is a nice 12-15 digits for early positive x, but trail off to 1 or so around x=8, (Figure 4), for values derived as $1 - \Phi(x)$ rather than $\Phi(-x)$

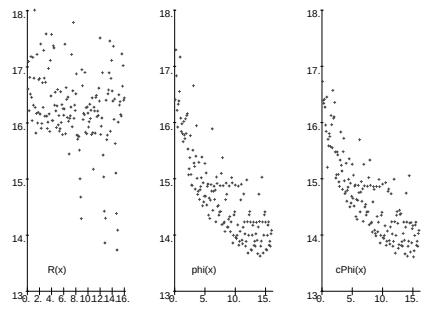
Some of the most widely used methods for erf or erfc are based on Algorithm AS66 by Hill (1973). A plot of the relative error for a double precision C version shows digits of accuracy around 11 at x=0, dropping to 8 at x=12.

Figure 3 shows relative errors for cPhi, derfc and AS66. That last one pales in comparison. If interpreted as $\Phi(-x)$, the Cody version has that same limiting relative accuracy exhibited by cPhi, derfc and Sun.

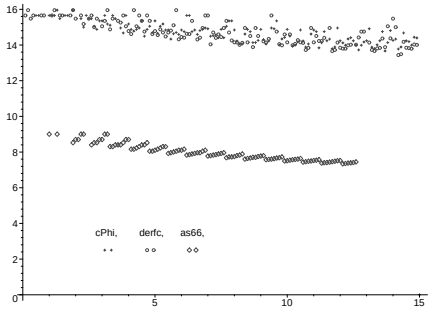
Relative errors using $1 - \Phi(x)$ are shown in Figure 4, for versions based on Hart, Cody and a Fortran version of AS66, which returns zero for $x > 12.6$. Note that the legend there uses cHart and cCody to indicate that 1-Hart and 1-Cody were used. They have the property ascribed above: $\Phi(-x)$ yields better relative accuracy than $1 - \Phi(x)$, where the particular function used to evaluate $\Phi$ is based on Hart or Cody. Using Hart($-x$) would not show the

rapid decline to zero beyond $x = 5$, but would taper off to about 9 digits at $x = 12$. Similarly, if the Cody implementation is used to get $\Phi(-x)$, rather than $1 - \Phi(x)$ as shown, the relative error plot compares favorably with the excellent results for `cPhi and derfc` in Figure 3, or the above table for `cPhi, Sun` and Ooura's `derfc`. There was no difference between $1 - \Phi(x)$ and $\Phi(-x)$ for the AS66 implementation of the normal distribution function. Both are poor.

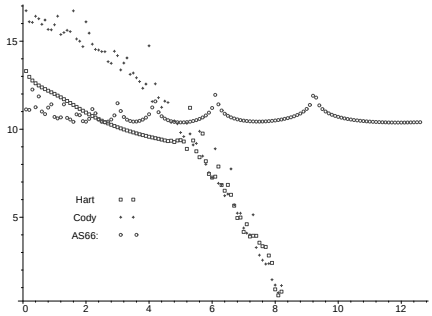
Plots comparing absolute and relative errors for Phi, cPhi and other sources:



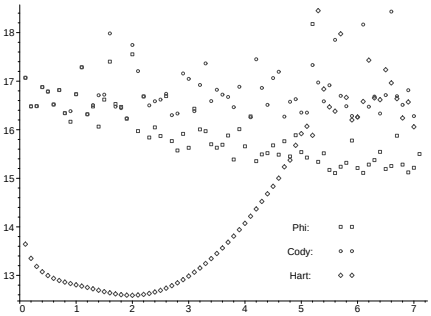

2. Relative error, R($x$), $\phi(x)$ and R($x$)$\phi(x)$




3. Relative error, cPhi(x), derfc(x) and AS66(x).




4. Relative error, cHart, cCody and AS66.




5. Absolute error, Phi(x), Hart and Cody.

**Figures 2,3,4,5 showing errors for Phi, cPhi and various implementations of erf and erfc.**

The plots in Figures 2,3,4,5 were patched from .ps versions of earlier, no longer available sources. Details of the plots as they appear here may be easily magnified by most pdf viewers, such as GSview or Adobe reader. (In GSview, just right-click the mouse.)

Some tentative conclusions can be drawn from the above discussion and plots:
For applications in probability and statistics, where absolute accuracy may be more important, for example when determining that $\Phi(5.3)$ is .999999989282410 compared to the true value of .99999989282409741..., the absolute accuracy provided by the tiny Phi(x) provides an attractive alternative to sources based on erf, comparable or better than all but Cody, which averages about 1/3 more digit of absolute accuracy but requires an elaborate, mysterious program to squeeze out that extra 1/3 of a digit.

For applications requiring relative accuracy, as in the extreme tails, there seems no better method than the short cPhi function listed above. It is better than many, and provides the limiting accuracy of the some of the best: Ooura, Sun or (properly applied), Cody. All of these rely on a multiple of $\exp(-cx^2)$ and suffer from the same limit on accuracy as cPhi: evaluation of exp(-t) for large t.

And of course they require that irritating transformation `.5*erfc(x/sqrt(2.))`.

## 5. Summary

If you want values $\Phi(x)$ of the standard normal distribution for use in probability and statistics, you are likely to find no better method than this little C function:

```
double Phi(double x)
{long double s=x,t=0,b=x,q=x*x,i=1;
while(s!=t) s=(t=s)+(b*=q/(i+=2));
return  .5+s*exp(-.5*q-.91893853320467274178L);}
```

which will provide probabilities with an absolute error less than $8 \times 10^{-16}$. (You may want to add a line that will return 0 for $x < -8$ or 1 for $x > 8$, since an error of $10^{-16}$ can make a true result near 0 negative, or near 1, exceed 1.)

If you want relative errors (digits of precision) near the limit available in double precision for extreme tails of the distribution, such as for solving differential equations, you might evaluate $1-\Phi(x) = \mathrm{cPhi}(x)$ as `.5*erfc(x/sqrt(2.))` by invoking the complementary error function erfc available from some compilers, or try one of many available on the web, with all the attendant mysteries over content and the nuisance of the two changes of scale. You will find that most of them have long listings, with constants involved in various polynomial, rational function or continued fraction approximations and that some fall short of the 14–16 digit accuracy limit on $\exp(-t)$ for large $t$. Or you might just paste this little, but not quite as little, cPhi function above your main C program:

```
double cPhi(double x)
{long double R[9]=
   {1.25331413731550025L, .421369229288054473L, .236652382913560671L,
    .162377660896867462L, .123131963257932296L,.0990285964717319214L,
    .0827662865013691773L,.0710695805388521071L,.0622586659950261958L};
int i,j=.5*(fabs(x)+1);
long double pwr=1,a=R[j],z=2*j,b=a*z-1,h=fabs(x)-z,s=a+h*b,t=a,q=h*h;
for(i=2;s!=t;i+=2){a=(a+z*b)/i; b=(b+z*a)/(i+1); pwr*=q; s=(t=s)+pwr*(a+h*b);}
s=s*exp(-.5*x*x-.91893853320467274178L);
if(x>=0) return (double) s; return (double) (1.-s);}
```

and have values for the complementary normal distribution

$$\mathrm{cPhi}(x) = \int_x^\infty \phi(t)\,dt = 1 - \Phi(x)$$

to the 14–16 digits available from the exp() function, as well as have an understanding of the mathematics behind the method.

## 6. Attachments

The 'browse files' section contains source code for the brief Phi and cPhi functions as well that for the `Hart, Cody, AS66, derfc and Sun` variations of erf or erfc used for comparison, particularly in the tables and in Figures 2,3,4,5. Additional sources for erf or normal CDF functions, Brown *et al.* (1997) and Kahaner *et al.* (1989), were suggested by a referee.

## References

Brown BW, Lovato J, Russell K (1997). *DCDFLIB: Library of Fortran Routines for Cumulative Distribution Functions, Inverses, and Other Parameters.* Department of Biomathematics, University of Houston.

Cody WJ (1993). "Algorithm 715." *ACM Transactions On Mathematical Software*, **19**, 22–32.

Glaisher JW (1871). "On a Class of Definite Integrals." *Philosophical Magazine*, **XXXII**, 294–301.

Hart JF, Cheney EW, Lawson CL, Maehly HJ, Mesztenyi CK, Rice JR, Thacher HG, Witzgall C (1968). *Computer Approximations.* Wiley, New York.

Hill ID (1973). "Algorithm AS66." *Applied Statistics*, **22**(3).

Kahaner D, Moler C, Nash S (1989). *Numerical Methods and Software.* Prentice Hall, Englewood Cliffs, NJ.

Newsgroup postings (2002). `sci.math, Aug 20,2002, thread of 25, Search under "Integrating a Bell Curve"`.

Newsgroup postings (2004). `comp.lang.c, Feb 16,2004, thread of 29, Search under "erf function in C"`.

Ooura T (1998). "C function derfc." E-mail: ooura@kurims.kyoto-u.ac.jp , URL http://momonga.t.u-tokyo.ac.jp/~ooura/gamerf.html.

Sun Microsystems (1993). `erfc.c listing at` http://www.netlib.org/fdlibm/.

**Affiliation:**

George Marsaglia
Professor Emeritus, Statistics
Florida State University
Home address: 1616 Golf Terrace Drive
Tallahassee FL 32301, United States of America
E-mail: geo@stat.fsu.edu

*Journal of Statistical Software*
July 2004, Volume 11, Issue 4.
http://www.jstatsoft.org/

*Submitted:* 2004-06-05
*Accepted:* 2004-07-18