Algorithm 885: Computing the Logarithm of the Normal Distribution

JEAN MARIE LINHART
StataCorp LP

We present and compare three C functions to compute the logarithm of the cumulative standard normal distribution. The first is a new algorithm derived from Algorithm 304’s calculation of the standard normal distribution via a series or continued fraction approximation, and it is good to the accuracy of the machine. The second is based on Algorithm 715’s calculation of the standard normal distribution via rational Chebyshev approximation. This is related to, and an improvement on, the algorithm for the logarithm of the normal distribution available in the software package R. The third is a new and simple algorithm that uses the compiler’s implementation of the error function, and complement of the error function, to compute the log of the normal distribution.

Categories and Subject Descriptors: G.1.0 [Numerical Analysis]: General—Numerical algorithms; G.1.2 [Numerical Analysis]: Approximation; G.3 [Probability and Statistics]: — Distribution functions

General Terms: Algorithms

Additional Key Words and Phrases: Normal distribution, logarithm of the standard normal distribution, error function, normal integral

ACM Reference Format:

1. INTRODUCTION

Computing the logarithm of the standard normal distribution (not to be confused with the lognormal distribution) is a problem that has been ignored in the literature. While the statistical program R [R Development CoreTeam 2007] has an implementation, the only other published work we found on the subject was Monahan [1981], which presents a single precision algorithm. The
modern state of the art seems to be to simply take the logarithm after calculating the standard normal distribution. This is accurate in the center of the distribution, but loses a great deal of precision in the tails.

The logarithm of the standard normal distribution is useful in maximum likelihood estimation, in which the logarithm of the likelihood is repeatedly calculated. The need for, and interest in, the logarithm of the standard normal distribution has also been sparked recently by Zeileis and Kleiber [2005], who find inaccuracies in the tails of the cumulative standard normal distribution function in certain implementations. Marsaglia [2004] also points out that problems exist in many implementations of the normal distribution.

Most research on algorithms for the standard normal distribution (also called the standard normal integral) and its brother, the error function, was performed in the 1960s, though interest continues in the literature through the current day. In one of the first algorithms published, Ibbetson [1963] uses a polynomial approximation to calculate the standard normal distribution to seven decimal places. This was improved by Cooper [1968] who gives a fixed-precision algorithm based on the series and continued fraction approach. This algorithm gave full precision of 11 figures on a 12-digit computer, but the precision is limited because the series has a fixed number of terms. This approach was remarked on by Hill [1969] and is used by by Hill and Joyce [1967a] in Algorithm 304, which uses a series and continued fraction approximation. Algorithm 304 along with the modifications of Holmgren [1970] and Adams [1969b], can calculate to any desired precision [Hill and Joyce 1967b; Brophy and Wood 1989]. The methods used in Algorithm 304 are essentially those in Abramowitz and Stegun [1972].

Adams [1969a] gives a rational function approximation for the standard normal distribution good to a relative error of $1 \times 10^{-10}$ to $1 \times 10^{-11}$, and this was later translated by Hill [1973] into a fast and accurate Fortran program. Cody [1969] uses the work of Clenshaw [1962] and Hart et al. [1968] to calculate a full double precision error function via rational Chebyshev approximation at approximately the same time as the related approach of Adams [1969a]. Cody’s algorithm calculates the error function to relative errors between $6 \times 10^{-19}$ and $3 \times 10^{-20}$. The error function can be used to calculate the standard normal distribution by a rescaling and a translation. Cody uses this in later work to compute the standard normal distribution as a part of Algorithm 715 Cody [1993]; this is the algorithm used in R Development CoreTeam [2007]. Schonfelder [1978] presents an algorithm using rational Chebyshev approximation to calculate the error function and standard normal distribution to 30 decimal places.

A survey of the literature on computing the standard normal distribution was given by Martynov [1981], and a more modern review is given by Marsaglia [2004]. This latter gives a table-free C function to compute the normal distribution with error less than $8 \times 10^{-16}$, and along with Zeileis and Kleiber [2005], Marsaglia points out there remain problems with various implementations for the error function and standard normal distribution. Marsaglia [2004] states that many implementations are still based on Hill [1973], Cody [1993] and Hart et al. [1968].

Three algorithms are chosen with a full double-precision calculation of the standard normal distribution to modify into a logarithm of the standard normal distribution. The first is essentially Algorithm 304 (Hill and Joyce [1967a] with the modifications of Holmgren [1970] and Adams [1969b]), and it can calculate the standard normal distribution to any desired precision. The second algorithm is based on Algorithm 715 [Cody 1993], and it computes the standard normal distribution via rational Chebyshev polynomial approximations to the error function. Rational Chebyshev approximations have fixed precision, but they are fast because they have a fixed number of calculations. An implementation of Algorithm 715 is included in the software package R. The code in R includes an option to calculate the log of the normal distribution; the logarithm code provided here based on Algorithm 715 has similarities to it, but it has better behavior in the tails than the R implementation. The third algorithm calculates the logarithm of the standard normal distribution via implementations of the error function and complement of the error function that are included with modern compilers (e.g., gcc; icc also includes the error and complementary error functions). These implementations should be fast, but without knowing which algorithm is used, they must be tested for accuracy.

2. ALGORITHMS

The logarithm of the standard normal distribution, \( \log(\Phi(x)) \), is given by:

\[
\log(\Phi(x)) = \log \left( \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} \, dt \right). \tag{1}
\]

As \( x \to -\infty \), \( \Phi(x) \to 0 \), and as \( x \to \infty \), \( \Phi(x) \to 1 \). Using double-precision arithmetic, if \( x > 8.5 \), \( \Phi(x) \) is calculated to be identically 1, and if \( x < -38 \), \( \Phi(x) \) is calculated to be identically 0. Consequently, we seek an implementation of the logarithm of the standard normal distribution to improve accuracy and detail in the tails.

In the first algorithm, we present an implementation in C of a modification of Algorithm 304 to compute the logarithm of the standard normal distribution. We call this implementation \texttt{lnnorm()}.

This algorithm for computing the logarithm of the standard normal distribution closely follows Algorithm 304 with the remarks of Adams [1969b] and Holmgren [1970]. In the center of the distribution, one can simply take the logarithm of the series representation for \( \Phi(x) \). If \( x > 0 \) this is given by:

\[
\Phi(x) = \frac{1}{2} + \frac{1}{2} \text{erf} \left( \frac{x}{\sqrt{2}} \right) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \sum_{n=0}^{\infty} \frac{x^{2n+1}}{1 \cdot 3 \cdot 5 \ldots (2n + 1)}. \tag{2}
\]

Likewise, if \( x < 0 \), use \(|x|\) and subtract. Here \( \text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_{0}^{z} e^{-t^2} \, dt \) is the error function of \( z \).

The first modification from Algorithm 304 is to use the series for \(|x| \leq 2\) and the continued fraction outside this interval, instead of using the series for \(-2.32 \leq x \leq 3.5\).
In the tails where $|x| > 2$, a continued fraction representation for the complement of the standard normal distribution is used:

$$Q(x) = 1 - \Phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \left[ \frac{1}{x} \left\{ 1 + \frac{-1}{x^2 + 3} \frac{-6}{x^2 + 7} \frac{-20}{x^2 + 11} \right. \right. $$

$$\left. \left. \left. \frac{-42}{x^2 + 15} \cdots \frac{-6(n-1) - 4(n-2)^2 + (n-2)}{x^2 + 3 + 4(n-1)} \right\} \right]\right]. \quad (3)$$

This implements the comment of Adams [1969b] to increase the speed of the algorithm by modifying the continued fraction. This continued fraction is equivalent to the one given in Abramowitz and Stegun [1972] and used by Hart et al. [1968] for testing.

This algorithm can scale to any desired precision (as noted by Hill [1973] and Brophy and Wood [1989]) by taking as many terms of the series or continued fraction as needed to achieve that precision.

For $x < 0$, $\Phi(x) = Q(-x)$, but for $x > 0$, one must compute $\Phi(x) = 1 - Q(x)$. In calculating the logarithm, we will avoid making this subtraction, which would result in a loss of precision. Instead, to calculate $\log(\Phi(x)) = \log(1 - Q(x))$, we use either a compiler implementation for the function $\log1p(y) = \log(1 + y)$ or, if $\log1p(y)$ is not available, a Taylor series for $\log(1 + y)$ is used to compute the result. We only use $\log1p$ if $|Q(x)| < 0.1$, where either the compiler implementation or the Taylor series for $\log(1 + y)$ is fast and accurate.

Extreme values of $x$ can wreak their own version of havoc in algorithms; we need to carefully work out the limiting behavior. As $x \to \infty$, $\log(\Phi(x)) \to 0$. Since we use double-precision arithmetic, we depart from Algorithm 304 by restricting the algorithm for $x > 38$ to return 0. Although $x \approx 38$ is not extreme, we find that $\log(\Phi(38))$ already returns 0.

On the other side, $x \to -\infty$, caution is required as extreme values give unique results. When $x < -2$, $\Phi(x) = Q(-x)$, and the continued fraction approximation of Algorithm 304 is used when $-1 \times 10^9 < x < -2$.

When $x \leq -1 \times 10^9$, an asymptotic approximation is used. To obtain it, observe that a first approximation to $Q(x)$ is:

$$Q(x) \approx \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \left( \frac{1}{x} \right). \quad (4)$$

Hence, a first approximation to the logarithm is:

$$\log(\Phi(x)) = \log(Q(-x)) \approx -x^2/2 + \log(1/|x|) - \log(\sqrt{2\pi}) \quad (5)$$

For $x \leq -1 \times 10^9$, the approximation:

$$\log(\Phi(x)) \approx -x^2/2 \quad (6)$$

is exact for double-precision arithmetic, and it is used to calculate $\log(\Phi(x))$, in another departure from Algorithm 304.
In the second and third algorithms, called \texttt{lnanorm()} and \texttt{lnenorm()} respectively, we present C functions to calculate the standard normal distribution via the error function. We have:

\begin{align*}
\Phi(x) &= \frac{1}{2} + \frac{1}{2} \text{erf} \left( \frac{x}{\sqrt{2}} \right) \\
Q(x) &= 1 - \Phi(x) = \frac{1}{2} \text{erfc} \left( \frac{x}{\sqrt{2}} \right). 
\end{align*}

The second algorithm, \texttt{lnanorm()}, based on Algorithm 715, uses rational Chebyshev approximation to find rational polynomial approximations for \texttt{erf()} and \texttt{erfc()} and then uses these formulas to calculate the standard normal distribution. See Clenshaw [1962] or Press et al. [1992] for a reference on rational Chebyshev approximation. Cody [1969] gives coefficients to calculate \texttt{erf}(x); Cody [1993] gives coefficients for \texttt{erf} \left( \frac{x}{\sqrt{2}} \right).

This algorithm has one clear weakness. The rational Chebyshev approximation has fixed precision. It is also over a fixed domain, an issue that comes to light when investigating the tails of \texttt{lnanorm()}. For \( x < -37.519 \) this algorithm sets the standard normal distribution to 0, and for \( x > 8.572 \) the standard normal distribution is set to 1. When calculating \( \log(\Phi(x)) \), we can use the greater of these limits, \( |x| \leq 37.519 \), to calculate; however, if \( |x| > 37.519 \), the returned result will be either \(-\infty\) or 0 depending on the sign of \( x \). This is especially galling in the negative tail, where \texttt{lnnorm()}, based on the continued fraction of Algorithm 304 and the subsequent asymptotic approximation, is able to calculate values to the limits of the machine. To remedy this and get improved values for \( x < -37.519 \), we call through to \texttt{lnnorm()} to remedy this problem.

The third algorithm, \texttt{lnenorm()}, uses, if available, the formulæ above and the compiler supplied functions \texttt{erf()} and \texttt{erfc()} to compute the standard normal distribution. The code we used is part of the GNU C library, and it is a relative of the algorithm by Cody [1969]. It has similar weaknesses to \texttt{lnanorm()} in the negative tail, and we use the same solution of calling through to \texttt{lnanorm()} to remedy this problem.

3. PACKAGE DESCRIPTION

The package is written in C, using IEEE double-precision arithmetic. There are three possible implementations for \( \log(\Phi(x)) \). The first is called \texttt{lnnorm()}, based on Algorithm 304. The second, \texttt{lnanorm()}, is based on Algorithm 715, and the third, \texttt{lnenorm()}, is based on the compiler supplied error function and complementary error function, if these are available. Implementation of a minor modification of Algorithm 304 for \( \Phi(x) \) is included for comparison purposes; this function is named \texttt{norm()}. Likewise the normal for Algorithm 715 is called \texttt{anorm()}, and the normal calculated from the error function is \texttt{enorm()}. A driver program \texttt{main} is provided; it generates the tables and timings in this article.
A global header file, norminc.h is included. In this file, various preprocessor directives may be redefined to customize the code for a particular setup. In particular, HAS_ERF may be undefined if the error function and complementary error function are not included with a compiler, and HAS_LOG1P may be changed if the compiler does not include a log1p() function. As an alternative, we include a simple Taylor series to calculate this value for small \( x \). Six definitions are used for double-precision arithmetic, and they may be modified for greater or lesser precision. The first three are LNORM_MAX, LNORM_MIN, and LNANORM_MIN to determine the behavior of the algorithm. The final three are constants: \( \sqrt{\pi} \), \( \sqrt{2\pi} \), and \( \sqrt{2} \).

4. TESTING AND TIMING

We use \texttt{norm()} and \texttt{lnnorm()} as reference routines to test against the two other routines for the standard normal distribution and the log of the standard normal distribution.

We expect, and find, relative errors of the logarithms to be comparable to the relative errors of the standard normals. The accuracy of \texttt{enorm(x)} and \texttt{lnenorm(x)} depends on the implementation of the error function. Absolute errors are also included, but they are of less interest. The glibc error function included with the gcc4 compiler is accurate to machine precision; it is similar to the algorithm for \texttt{anorm}().

Testing is performed on an Intel Pentium D series (dual-core with 64-bit extensions (EM64T)) running Fedora Core 7 and compiled using gcc4.

We investigate the error on \([-5, 5]\) to get a feel for what happens in the center of the distribution; then we investigate the error on \([-37.519, 37.519]\), the full range where the three algorithms differ. The tables present the mean and maximum absolute and relative errors for 1001 evenly spaced and 1001 random points on these intervals compared to values generated by \texttt{norm(x)} and \texttt{lnnorm(x)}.

### 1001 Evenly Spaced Points on Interval from -5.0000 to 5.0000

<table>
<thead>
<tr>
<th>Difference</th>
<th>anorm</th>
<th>enorm</th>
<th>lnnorm</th>
<th>lnenorm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum absolute</td>
<td>2.2898e-16</td>
<td>2.2204e-16</td>
<td>9.3299e-15</td>
<td>8.8518e-15</td>
</tr>
<tr>
<td>Mean absolute</td>
<td>3.4159e-17</td>
<td>3.1812e-17</td>
<td>4.4768e-16</td>
<td>5.2428e-16</td>
</tr>
<tr>
<td>Mean relative</td>
<td>4.3037e-16</td>
<td>5.9409e-16</td>
<td>4.9798e-16</td>
<td>6.767e-16</td>
</tr>
</tbody>
</table>

### 1001 Evenly Spaced Points on Interval from -37.5190 to 37.5190

<table>
<thead>
<tr>
<th>Difference</th>
<th>anorm</th>
<th>enorm</th>
<th>lnnorm</th>
<th>lnenorm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum absolute</td>
<td>2.2204e-16</td>
<td>2.2204e-16</td>
<td>1.1369e-13</td>
<td>2.2737e-13</td>
</tr>
<tr>
<td>Maximum relative</td>
<td>5.6897e-14</td>
<td>1.9644e-13</td>
<td>5.6053e-14</td>
<td>2.0823e-13</td>
</tr>
<tr>
<td>Mean relative</td>
<td>4.6741e-15</td>
<td>1.7794e-14</td>
<td>4.7691e-15</td>
<td>1.75e-14</td>
</tr>
</tbody>
</table>
Algorithm 885: Computing the Logarithm of the Normal Distribution

### 1001 Random Points on Interval from -5.0000 to 5.0000

<table>
<thead>
<tr>
<th>Difference</th>
<th>anorm</th>
<th>enorm</th>
<th>lnanorm</th>
<th>lnenorm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum relative</td>
<td>8.2059e-15</td>
<td>7.8213e-15</td>
<td>1.1529e-14</td>
<td>1.0844e-14</td>
</tr>
<tr>
<td>Mean absolute</td>
<td>3.0568e-17</td>
<td>2.8687e-17</td>
<td>4.5628e-16</td>
<td>5.3581e-16</td>
</tr>
<tr>
<td>Mean relative</td>
<td>3.9894e-16</td>
<td>5.4155e-16</td>
<td>5.0785e-16</td>
<td>6.9251e-16</td>
</tr>
</tbody>
</table>

### 1001 Random Points on Interval from -37.5190 to 37.5190

<table>
<thead>
<tr>
<th>Difference</th>
<th>anorm</th>
<th>enorm</th>
<th>lnanorm</th>
<th>lnenorm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum absolute</td>
<td>2.2204e-16</td>
<td>2.2204e-16</td>
<td>1.1369e-13</td>
<td>2.2737e-13</td>
</tr>
<tr>
<td>Maximum relative</td>
<td>5.6274e-14</td>
<td>2.0273e-13</td>
<td>5.7096e-14</td>
<td>1.8526e-13</td>
</tr>
<tr>
<td>Mean absolute</td>
<td>4.9602e-18</td>
<td>4.3459e-18</td>
<td>6.6618e-15</td>
<td>1.8359e-14</td>
</tr>
<tr>
<td>Mean relative</td>
<td>4.7698e-15</td>
<td>1.7247e-14</td>
<td>4.7343e-15</td>
<td>1.6987e-14</td>
</tr>
</tbody>
</table>

The relative errors for $\ln\text{anorm}(x)$ and $\ln\text{enorm}(x)$ are similar to those for $\text{anorm}(x)$ and $\text{enorm}(x)$, indicating that accuracy for the logarithms is similar to that for the normal standard distribution functions. Values of $\text{norm}(x)$, $\text{anorm}(x)$, and $\text{enorm}(x)$ can be compared to tabulated values in Abramowitz and Stegun [1972], which are available $0 \leq x \leq 5$ to 15 decimal places. Hand comparing values to the table indicates that the last place (15th decimal place) is sometimes in error with each of the three, and there is no pattern except that $\text{anorm}(x)$ and $\text{enorm}(x)$ usually agree.

This gives no basis for preferring one implementation over another. Agreement over the interval $[-5, 5]$ is tighter, with a maximum relative error on the order of magnitude of $1 \times 10^{-14}$ for the logarithms; this indicates most of the discrepancy is in the tails of the distribution. We suspect that $\ln\text{anorm}(x)$ and $\ln\text{enorm}(x)$ lose a small amount of accuracy in the tails. We can verify that $\ln\text{norm}(x)$ gives a correct result at $x = -1.0 \times 10^{-8}$ and $x = -1.0 \times 10^{-9}$, by using the asymptotic approximations (5) and (6) to compare with the results from the continued fraction. The asymptotic approximations used are accurate to IEEE double precision.

Accuracy can also be assessed using tabulated values of the logarithm of the standard normal distribution. We use two tables with values from the log of the standard normal distribution for testing. The first is in Abramowitz and Stegun [1972] and was taken from Pearson and Hartley [1954]. Table 26.2 of Abramowitz and Stegun [1972] contains $-\log_{10} Q(x)$ for $5 \leq x \leq 500$, where

$$-\log_{10} Q(x) = -\log \Phi(-x)/\log(10).$$

We obtain $-\ln\text{norm}(-500)/\ln(10) \approx 54289.9082995821 \approx 54289.90830$, the value given in the table. This table is replicated exactly by all three algorithms. Results in the table have from six to ten significant figures with five decimal places. This table has values outside of the range of $[-37.519, 37.519]$, so it is of interest even with the reduced precision of the answers.

---

1. This table was compiled by Julia Bell. The author of this article is amused to note that women have had a long history with the logarithm of the standard normal distribution.
2. This table was not published in the 1966 edition.
The *Index of Mathematical Tables* Fletcher and Rosenhead [1962] pointed to a 16-decimal-place log of the normal distribution in Shep pard [1939]. This table has values of \(-\log(\Phi(-x))\) for \(0 \leq x \leq 10\) only. Table IV from Sheppard was typed into a text file, and then used to compare to \(\text{lnnorm}()\), \(\text{lnanorm}()\), and \(\text{lnenorm}()\). Using \(\text{lnnorm}()\) as an example, \(\text{lnnorm}(-x) = -L(x)\), where \(L(x)\) is the value given in Table IV.

We split the error calculations against the tabulated values into two sections because the larger results for \(x \geq 5\) have higher precision values in the table.

Comparison with Tabulated Values from 0.0000 to 5.0000

<table>
<thead>
<tr>
<th>Error</th>
<th>(\text{lnnorm})</th>
<th>(\text{lnanorm})</th>
<th>(\text{lnenorm})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum absolute</td>
<td>9.77\text{e-15}</td>
<td>3.5527\text{e-15}</td>
<td>3.5527\text{e-15}</td>
</tr>
<tr>
<td>Maximum relative</td>
<td>2.5825\text{e-15}</td>
<td>2.9296\text{e-16}</td>
<td>2.9756\text{e-16}</td>
</tr>
<tr>
<td>Mean absolute</td>
<td>1.1821\text{e-15}</td>
<td>6.3783\text{e-16}</td>
<td>7.2056\text{e-16}</td>
</tr>
<tr>
<td>Mean relative</td>
<td>2.8316\text{e-16}</td>
<td>1.0523\text{e-16}</td>
<td>1.1419\text{e-16}</td>
</tr>
</tbody>
</table>

Comparison with Tabulated Values from 5.0000 to 10.0000

<table>
<thead>
<tr>
<th>Error</th>
<th>(\text{lnnorm})</th>
<th>(\text{lnanorm})</th>
<th>(\text{lnenorm})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum absolute</td>
<td>1.4211\text{e-14}</td>
<td>7.1054\text{e-15}</td>
<td>2.1316\text{e-14}</td>
</tr>
<tr>
<td>Maximum relative</td>
<td>3.7901\text{e-16}</td>
<td>2.1268\text{e-16}</td>
<td>4.688\text{e-16}</td>
</tr>
<tr>
<td>Mean absolute</td>
<td>3.0303\text{e-15}</td>
<td>2.8213\text{e-15}</td>
<td>4.528\text{e-15}</td>
</tr>
<tr>
<td>Mean relative</td>
<td>9.5221\text{e-17}</td>
<td>8.1215\text{e-17}</td>
<td>1.3096\text{e-16}</td>
</tr>
</tbody>
</table>

Each algorithm produces results accurate to double precision. \(\text{lnanorm}(x)\) performs marginally better than the other two on this interval; however, there is no substantive basis for preferring one implementation over another based on these results.

Timings were performed using the interval \([-37.519, 37.519]\) and a for loop. Overhead is initially included in the timings and the function calls; the overhead is later subtracted out. The overhead includes the for loop and the calculation of a random \(x\). One million calls to \(\text{lnnorm}()\) were completed in approximately 0.64 seconds, to \(\text{lnanorm}()\) in approximately 0.39 seconds, and to \(\text{lnenorm}()\) in approximately 0.33 seconds. Using compiler optimizations (the \(-O\) flag to gcc), the timings were 0.43, 0.32, and 0.28 seconds, respectively. As expected, \(\text{lnanorm}()\) and \(\text{lnenorm}()\) are faster, about one and one-half times to twice as fast as \(\text{lnnorm}()\), with \(\text{lnenorm}()\) edging out \(\text{lnanorm}()\).

5. CONCLUSION

All three logarithms of the standard normal distribution have acceptable double-precision accuracy, with \(\text{lnanorm}()\) edging out the other two. For speed, \(\text{lnenorm}()\), which uses the error function and complementary error function provided by the compiler, is marginally faster than \(\text{lnanorm}()\), and both are about twice as fast as \(\text{lnnorm}()\). For simplicity, speed, and a strictly double-precision implementation, \(\text{lnenorm}()\) is preferred, provided that a built-in error function is provided and its accuracy can be established. For a strictly double-precision implementation that is portable, \(\text{lnanorm}\) is preferred over
the other two for having the best accuracy and nearly the best speed. For a portable implementation with precision that will scale with the precision of the computer, $\ln\text{norm}(\cdot)$ is preferred. An implementation of the continued fraction part of $\ln\text{norm}(\cdot)$ is required regardless of which implementation is chosen to get accuracy out into the negative tail of the distribution; $\ln\text{anorm}(\cdot)$ and $\ln\text{enorm}(\cdot)$ both call through to $\ln\text{norm}(\cdot)$ when $x < -37.519$.

REFERENCES


Received November 2007; revised January 2008; accepted January 2008