[6.35] Cumulative normal distribution and inverse

The cumulative normal (or Gaussian) distribution is defined as

$$\Pr\{X \le x\} = p(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{(t-m)^2}{2\sigma^2}}\, dt \qquad [1]$$

where *m* is the mean and σ^2 is the variance, so σ is the standard deviation. In other words, this integral is the area under the probability function to the left of *x*. This is also sometimes called the 'lower tail' of the distribution. When the mean is 0 and the variance is 1, this is called the *standard* cumulative normal distribution.

You can find the cumulative normal distribution by integrating equation [1] with the *nint()* function. The function below, *cndint(),* does this.

```
cndint(x,m,s)
Func
©(x,mean,std dev) find CND with integration
©29oct00/dburkett@infinet.com
1/(s*√(2*π))*nint(e^(-.5*((t-m)/s)^2),t,-∞,x)
Endfunc
```

For x = 0, mean = 0 and standard deviation = 1, *cndint()* returns the probability p = 0.49999999952549 in about 7.5 seconds on a HW2 92+ with AMS 2.05. The answer should be p = 0.5. For x = 6, mean = 0 and standard deviation = 1, *cndint()* returns p = 0.99999999853883 in about 10.5 seconds. This answer is in error by about -5E-10. We can write functions that are much faster and somewhat more accurate.

It turns out that the cumulative normal distribution (called CND from now on) can be found efficiently with the error function erf(x). First, this identity relates the standard CND to the general CND:

$$\Pr\{X \le x\} = P\left(\frac{x-m}{\sigma}\right) \qquad [2]$$

where P(x) is the standard CND. This converts a distribution with mean *m* and standard deviation σ to the standard CND. Next, this identity relates P(x) to erf(x):

$$\text{erf}(x) = 2 \cdot P(x\sqrt{2}) - 1 \qquad [3]$$

which can be solved for

$$p(x) = \frac{\text{erf}\left(\frac{x}{\sqrt{2}}\right)+1}{2} \qquad [4]$$

This can be combined with [2] to yield a simple function to find the general CND, like this:

```
cnd(x,m,s)
func
©(x,m,s) find cum norm dist at x, mean m, std dev s
©Find cumulative distribution at x, with mean m and standard deviation s
©dburkett@infinet.com 8jun00

(erf(((x-m)/s)/1.4142135623731)+1)/2

Endfunc
```

This function calls *erf(),* described in tip [6.34], which must be in the same folder as *cnd().* That folder must be the current folder.

As an example, find the standard cumulative normal distribution for x = 1:

```
cnd(1,0,1)
```
returns 0.841345

To find the cumulative normal distribution for at x = 12, for a distribution with a mean of 10 and a standard deviation of 2.5, use

```
cnd(12,10,2.5)
```
which returns 0.788145

For some problems you need the 'upper tail' distribution, which is the area to the *right* of the sample, or the probability that the random variable is greater than some *x*. This can be found with *cnd(),* as expected: just use 1 - *cnd().* For example, to find the upper tail normal distribution for x = 2, mean = 0 and standard deviation = 1, use

```
1-cnd(2,0,1)
```
which returns 0.0227501319482

For some other problems, you might need to find the probability that the random variable is between two limits x1 and x2, where x2 > x1. In this case, use

```
cnd(x2,m,s) - cnd(x1,m,s)
```

where *m* is the mean and *s* is the standard deviation. For example, to find the probability that the random variable is between 1.6 and 2.2, for a distribution with mean = 2 and standard deviation = 0.5, use

```
cnd(2.2,2,.5) - cnd(1.6,2,.5)
```
which returns 0.443566

For real problems, the mean and standard deviation will probably have more digits, and you can save a little typing by entering this example like this, instead:

```
cnd(2.2,m,s)-cnd(1.6,m,s)|m=2 and s=.5
```

The CND function is available in the free TI *Statistics and List Editor* (SLE) flash application. However, if you don't need all the additional functions that the SLE includes, you may prefer to use these two simple functions, instead of filling up your flash memory with the 300K SLE. In addition, the function shown here is more accurate than that included in the SLE application. For the upper-tail distribution with x = 2, mean = 0 and standard deviation = 2,the SLE returns a result of .022750062014, which is in error by about 8E-8.

It is occasionally necessary to find the inverse of distribution function, that is, to find the value of the random variable *x*, given a probability, a mean and a standard deviation. There is no closed-form solution to this problem. One possiblity is to use the built-in *nSolve()* function with *cnd().* For example, suppose you know that the distribution has a mean of 7, a standard deviation of 1. What is the value of the random variable *x* for which the probability is 0.2? Use *nSolve()* like this:

```
nSolve(cnd(x,7,1)=0.2,x)
```

which returns x = 6.158379 in about 19 seconds on a HW2 92+.. You can (and should) check this solution with

```
cnd(6.158379,7,1)
```
which returns 0.2, as hoped.

The execution time can be considerably reduced by providing *nSolve()* with a good initial guess for the value of *x*. If the initial guess is good enough, we don't need to use *nSolve()* at all, and the result is returned very quickly.

For the standard CND, with mean = 0 and standard deviation = 1, Odeh and Evans give this estimating function for x_p for a given probability *p*:

$$x_p = f(t) = t + \frac{p_4t^4+p_3t^3+p_2t^2+p_1t+p_0}{q_4t^4+q_3t^3+q_2t^2+q_1t+q_0} \qquad \text{and} \qquad t = \sqrt{-2\ln(p)} \qquad \text{and } 10^{-20} < p < 0.5 \qquad [5]$$

and

p_0 = -0.32223 24310 88	q_0 = 0.09934 84626 060
p_1 = -1.0	q_1 = 0.58858 15704 95
p_2 = -0.34224 20885 47	q_2 = 0.53110 34623 66
p_3 = -0.02042 31202 45	q_3 = 0.10353 77528 50
p_4 = -0.45364 22101 48 E-4	q_4 = 0.38560 70063 4 E-2

For the general case in with mean *m* and standard deviation *s*, we have

$$x_p = s \cdot f(t) + m \qquad [6]$$

using the identity in equation [2]. Finally, we use the symmetry of the normal distribution about the mean to account for $p \geq 0.5$, like this:

$$x_p = [\text{sgn}(p - 0.5)][s \cdot f(t)] + m \qquad [7]$$

Here, sgn(x) is a 'sign' function such that

$$\text{sgn}(x) = \left\{ \begin{array}{l} -1 \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{array} \right\} \qquad [8]$$

We cannot use the built-in *sign()* function, because it returns the expression 'sign(0)' for x=0, and we want 1. This is fixed with a *when()* function:

```
sgn(x) = when(x≠0,sign(x),1)
```

The function shown below, *cndif(),* implements the algorithm just described to find the inverse of the cumulative distribution function.

```
cndif(p,m,s)
func
©(probability,mean,std dev) inverse CND
©Inverse cumulative normal distribution
©Returns x with probability that X≤x
©Fast version.
©29oct00/dburkett@infinet.com

local r,t

p→r
if p>.5:1-r→r
√(-2*ln(r))→t

when(p≠.5,sign(p-.5),1)*s*(t+(polyeval({-.453642210148E-4,-.204231210245E-1,-.34
2242088547,-1,-.322232431088},t)/polyeval({.38560700634E-2,.10353775285,.5311034
62366,.588581570495,.99348462606E-1},t)))+m
```

```
Endfunc
```

For example, find *x* with a probability of 0.25, for a distribution with mean = 20 and standard deviation = 0.15:

```
cndif(.25,20,.15)                    returns x = 19.898826537593
```

We can check this answer with *cnd():*

```
cnd(19.898826537593,20,.15)          returns p = 0.25000000025933
```

cndif() is fast, since it simply uses two *polyeval()* functions to calculate the estimating rational polynomial. The reference for Odeh and Evans' estimating function claims about 7 significant digits of accuracy. If you need more accuracy than this, you can use this function, *cndi():*

```
cndi(p,m,s)
func
©(probability,mean,std dev) inverse CND
©Inverse cumulative normal distribution
©Returns x with p=probability that X≤x
©Accurate, slow version.
©29oct00/dburkett@infinet.com

local r,t

p→r
if p>.5:1-r→r
√(-2*ln(r))→t

when(p<.999999713348,nsolve(cnd(x,m,s)=p,x=when(p≠.5,sign(p-.5),1)*s*(t+(polyeva
l({-.453642210148ᴇ-4,-.204231210245ᴇ-1,-.342242088547,-1,-.322232431088},t)/poly
eval({.38560700634ᴇ-2,.10353775285,.531103462366,.588581570495,.99348462606ᴇ-1},
t)))+m),when(p≠.5,sign(p-.5),1)*s*(t+(polyeval({-.453642210148ᴇ-4,-.204231210245
ᴇ-1,-.342242088547,-1,-.322232431088},t)/polyeval({.38560700634ᴇ-2,.10353775285,
.531103462366,.588581570495,.99348462606ᴇ-1},t)))+m)

Endfunc
```

cndi() uses *nSolve()* to find *x*, with an initial guess found by Odeh & Evans' expression. However, if p > 0.999999713348, *cndi()* just uses Odeh & Evan's expression. This value of *p* corresponds to a standard deviation of 5. Above this limit, *nSolve()* does not improve the accuracy, and takes much longer.

cndi() typically executes in a few seconds. While not extremely impressive, this is faster than simply using *nSolve()* with no initial guess.

These are the references I used to develop the functions:

Handbook of Mathematical Functions, Abramowitz and Stegun, Dover, 1965. The standard normal probability distribution, and the transformation to the general normal distribution, are described on p931. The relation to the error function is shown on p934.

Statistical Computing, Kennedy and Gentle, Marcel Dekker, 1980. Odeh and Evans' approximation for the inverse cumulative normal distribution is shown on p95.