LGS Simulation Lecture 1
Scope and Key Algorithms

Introduction

The most fundamental problem is to calculate the expectation of a set of cash flows under various assumptions about probability distributions. Such expectations are essentially integrals of cash-flows over explicit or very implicit density functions. Sometimes these can be done analytically (get a formula) or semi-analytically (numerical integration of a formula). But there are numerous problems that may be tackled, or have to be tackled by simulation. Here is a reasonably good set of approaches:

1. Simulation of a known exact univariate terminal distribution (TD) to price an instrument based on a single asset;

2. Simulation of a known exact multivariate terminal distribution to price an instrument based on two or more assets;

3. Simulation of a multivariate terminal distribution based on a combination of marginals and a *copula* to price an instrument based on two or more assets;

4. Any of 1-3 using space-filling algorithms rather than random numbers;

5. Modulation of any of the above by some method for the introduction of skewness/kurtosis, or general change of TD to assess distribution risk;

6. Use of any of the above in a coarse asset-price path - e.g. we make jumps between widely separated events using an effective "terminal" distribution at each event;

7. Detailed simulation of the solution of the raw SDE by Euler, Milstein... methods, i.e., we build a fine path to simulation the random walk in detail.

I will try to take an optimal route through these approaches given the time available. The thing I want to say up front is that you should only go to 7 if the terminal distribution is non-integrable analytically and/or a pig to simulate. It is a common beginners' error to simulate an entire detailed path to do a European option, or even a coarse Asian tail within the standard Black-Scholes framework.

Note also that an understanding of simulating the Normal or log-Normal distribution is critical, and is relevant to ALL of the above cases, and not just under idealized assumptions. If we have an SDE with Brownian motion we still need Normal samples to numerically solve SDEs. And we might still use a Gaussian copula even with rather exotic marginal distributions.

Books and other resources

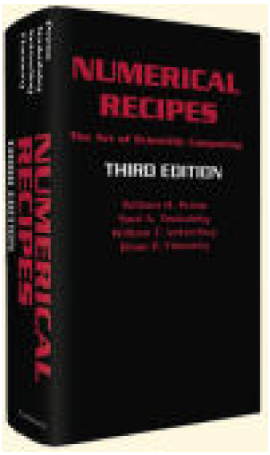
There are several that I can recommend.

You or your department/bank will probably have several copies of this lying around:

Numerical Recipes in C++ (or indeed any other language for the mathematical discussion). Press et al, Cambridge University Press, 2002. Chapter 7 discusses simulation aspects.

But there is now a Third Edition, which has in fact standardized on C++!

http://www.nr.com/



I will refer you to this for discussion of a lot of non-quantile based sampling. The books I have used most beyond that are:

P. Jaeckel, Monte Carlo Methods in Finance (See also his web site at www.jaeckel.org.)

P. Glasserman, "Monte Carlo Methods in Financial Engineering", Springer, 2005;

P.E. Kloden and E. Platen, "Numerical Solution of Stochastic Differential Equations", Springer, 2000.

You should also see Luc Devroye's classic text

Non-Uniform Random Variate Generation, Springer 1986.

This is out of print and Prof Devroye is so cross with Springer about not reprinting he has put it up on his web site. His web site has moved a bit because his comments tend to annoy people but it is brilliant. I looked a few days ago and it is currently at

http://cg.scs.carleton.ca/~luc/rnbookindex.html

but you should also look at Luc's other stuff, especially the links on random number generation. Note that the book is now a little dated in places but it remains a classic. I and many others have encouraged him to update it.

For copula theory in particular a lot of simulation work is discussed in

Copula Methods in Finance, Cherubini et al 2004, Wiley.

Your Prof's recent papers/preprints

So this is an area where I have found myself writing a bit on recently. The following papers maight be of interest. Though the the first two focus a lot on making sense of the univariate and multivariate Student distribution they also explain a lot about the link between copula theory, Quantiles, multivariate distributions, and preferred sampling methods, and have some worked examples.

Shaw, W.T., 2006, Sampling Student's T distribution – use of the inverse cumulative distribution function. Journal of Computational Finance, Vol 9 Issue 4, pp 37-73, Summer 2006 (on web)

W.T. Shaw and K.T.A. Lee, *Bivariate Student t distributions with variable marginal degrees of freedom and independence*, Journal of Multivariate Analysis (2007), doi:10.1016/j.jmva.2007.08.006.

G. Steinbrecher and W.T. Shaw, 2008, *Quantile Mechanics*, EJAM (on web)

W.T. Shaw, 2007, *Dependency without Copulas or Ellipticity*, European Journal of Finance.

The main papers and new ArXiv preprints are linked from the course web page. All of the others can be found through the links from my home page or papers page

http://www.mth.kcl.ac.uk/~shaww/web_page/papers/WebChronoPapers.htm

let me know if you have problems getting anything.

Special Methods vs Quantile Functions

In elementary probability we often learn how to sample a distribution using a Quantile function (more about this in a moment). Then when we first do simulation we find out that this is a bit hard for key distributions (e.g Normal), so we invent special methods, like Box-Muller and Polar-Marsaglia. Then after a while we learn that these special methods are a bit klunky when you try and use them with copula theory or sampling based on space-filling methods, so we migrate back to Quantiles. So I will focus a lot on Quantile methods, but will remind you of a key special method.

Reminder on Quantile Functions

Given a distribution function $F_X(x)$, a simple means of simulation is to set

$$X = F_X^{-1}(U) \tag{1}$$

where U is a sample from the uniform distribution on [0, 1].

Making U is compsci probem (see NumRec Section 7.1 and Glasserman) and not something I am interested in, but you MUST take care not to use STUPID methods that only produce a "few" independent numbers. Use the more advanced routines in Glasserman and NR.

The inverse CDF: $F_X^{-1} = Q_{F_X}$ is the Quantile Function associated with the distribution. We do not have to do this, witness the use of Box-Muller, Polar-Marsaglia methods for the Normal case, and its extension to Student by Bailey. But it is very useful if we can, particularly if we are working with algorithms based on hypercube-filling quasi-Monte-Carlo methods, or in particular copula methods.

Quantile Functions and Copulas

For the case of copula-based simulation where there is a real underlying bivariate distribution, we make a sample (X_1, X_2) from a given bivariate (in general multivariate) distribution (we have not seen how to do that yet, but I need the principle now!), then form a sample from the associated copula:

$$\{U_1, U_2\} = \left\{F_{X_1}(X_1), F_{X_2}(X_2)\right\} \tag{2}$$

Then to get samples with marginals with CDFs G_i:

$$\{Y_1, Y_2\} = \left\{G_1^{-1}[U_1], G_2^{-1}[U_2]\right\} = \left\{Q_{G_1}[U_1], Q_{G_2}[U_2]\right\} \tag{3}$$

It is always helpful to know quantile functions. But note also we do not need the U_i, at least when we are dealing with samples from a real distribution, for it would suffice to understand the *composite* mappings in the following:

$$\{Y_1, Y_2\} = \left\{G_1^{-1}\left[F_{X_1}(X_1)\right], G_2^{-1}\left[F_{X_2}(X_2)\right]\right\} \tag{4}$$

This is one of several motivations for considering composite maps of the form $y = G^{-1}[F(x)]$, where F, G are CDFs. I call this a distributional transmutation map.

Note that not all copulas come from identifiable bi- or multi-variate distributions. Sometimes one just postulates them together with a sampling rule. This causes all sorts of arguments. You might like to Google: Mikosch Copulas "Tales and Facts" to see an entertaining discussion. Otherwise see "Copula Methods in Finance". Actually things are not as clear cut as is suggested in some of these discussions - see the paper by Shaw and Lee on the multivariate Student distribution for lots of ways to make a Student T copula. There really is no such thing as "*The* T Copula". That paper also explains how to simulate a number of T copulas in practice.

However you make a copula, it is the glue that introduces dependency. The marginals are treated separately and most easily by a Quantile function.

The Normal Quantile Function

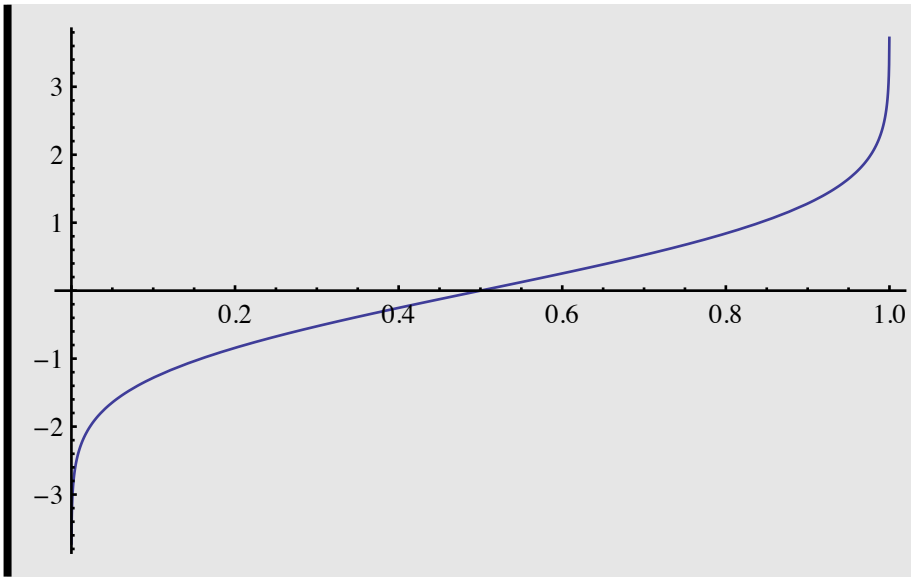
Recall the following definition that expresses the Normal CDF in terms of the error function "erf": in *Mathematica* we can see it

```
Ncdf[x_] := (1 + Erf[x/Sqrt[2]])/2
```

This can formally be inverted with

```
QuantileN[u_] := Sqrt[2] InverseErf[2 u - 1]
```

```
Plot[QuantileN[u], {u, 0.0001, 0.9999}]
```



This function is actually quite expensive to work out computationally so people have been lead to other ways of sampling the Nomal. The two most commonplace options are (a) converting samples from a square (b) approximating this function. Let's look at (b) first as it gives a practical fast method of using the easy Quantile function directly.

The Beasley-Springer/Moro approximation

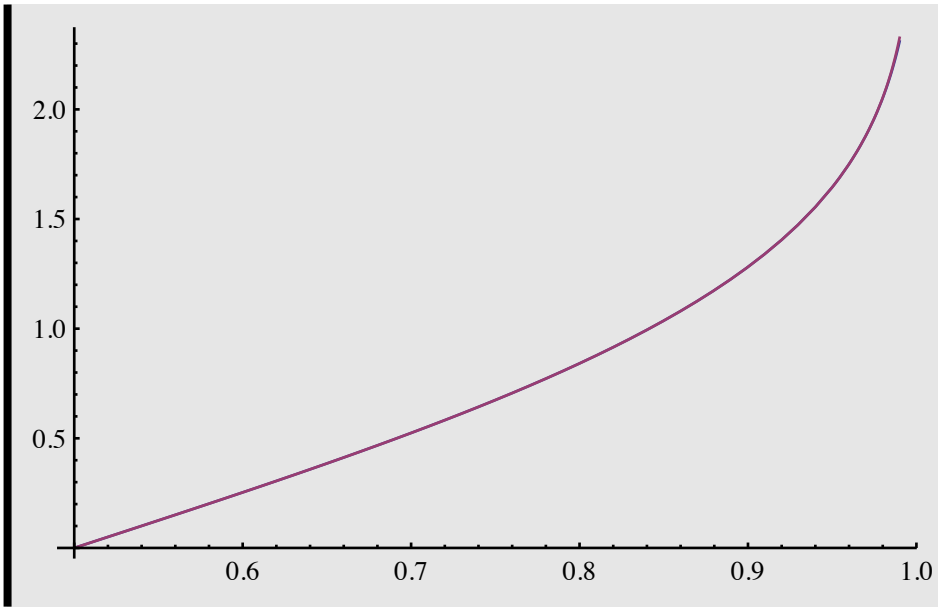
This has two bits, consisting of a piece to deal with $0.5 \le u \le 0.92$, defined originally by Beasley-Springer, and a tail construction developed by Moro. (see RISK, The full Monte, B. Moro, Risk 8, Feb, p. 57-58.). Symmetry takes case of the other half-region.

```
a = {2.50662823884, -18.61500062529, 41.39119773534,
   -25.44106049637};
b = {-8.47351093090, 23.08336743743, -21.06224101826,
   3.13082909833};
```

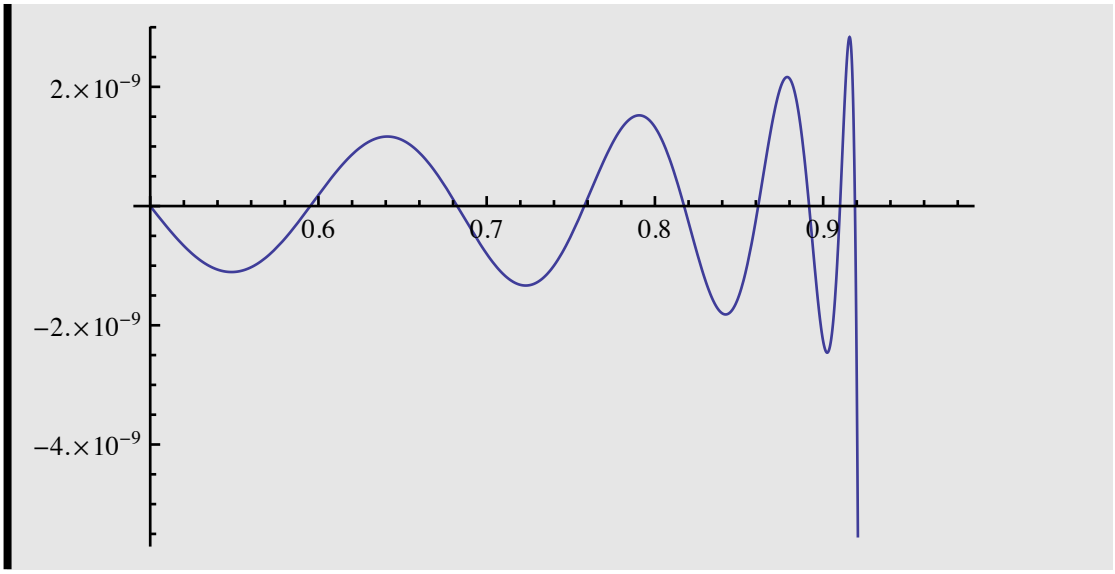
Note that if you look in Glassermann's otherwise excellent book he has a discussion in Section 2.3.2. The algorithm is specified in terms of constants in both equation 2.27 and Fig. 2.12. These formulae are inconsistent and the constants need to be used in the nested polynomial form implied by Fig 2.12. and not equation 2.27.

```
BSN[u_] := Module[{v = u - 1 / 2, r = (u - 1 / 2)^2},
  v * (a[[1]] + r * (a[[2]] + r * (a[[3]] + r * (a[[4]])))) /
    (1 + r * (b[[1]] + r * (b[[2]] + r * (b[[3]] + r * b[[4]])))) ]
```

```
Plot[{BSN[u], QuantileN[u]}, {u, 1/2, 0.99}]
```



```
Plot[BSN[u] - QuantileN[u], {u, 1/2, 0.98}]
```



So this is pretty good and the reason for the restriction to 0.92 is obvious (though it would have probably been created to work in this interval)

```
Table[BSN[u] - QuantileN[u] , {u, 1 / 2, 0.92, 0.0001}];
```

```
Max[Abs[%]]
```

```
3.00778 × 10^-9
```

■ Dealing with the Tail Issue

Moro's main contribution in his RISK article was to nail down an accurate series for the tail using a polynomial in

$$r = \log(-\log(1-u)) \tag{5}$$

```
c = {0.3374754822726147, 0.9761690190917186, 0.1607979714918209,
   0.0276438810333863, 0.0038405729373609, 0.0003951896511919,
   0.0000321767881768, 0.0000002888167364, 0.0000003960315187};
```

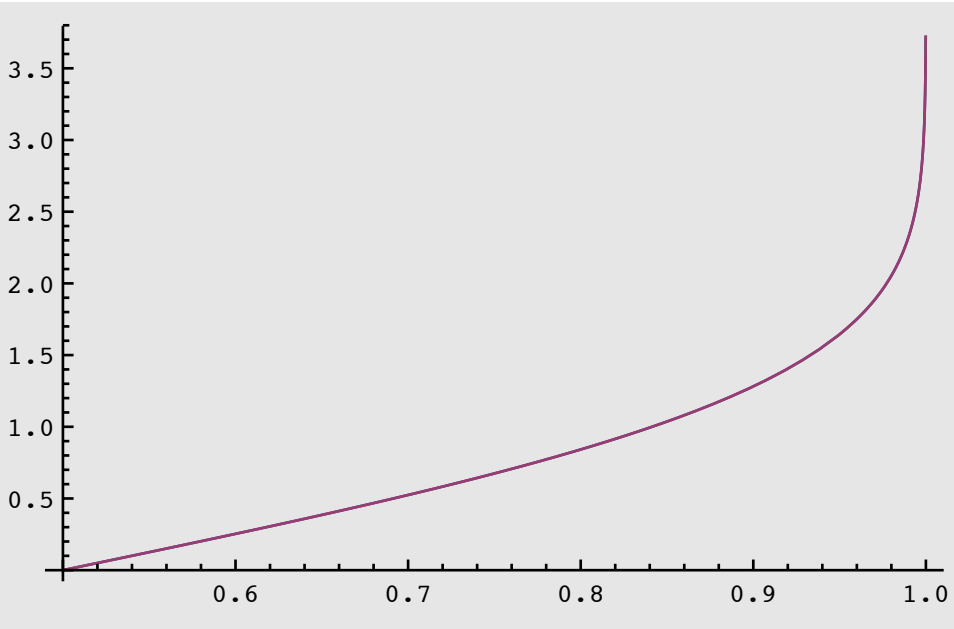
```
MoroTail[u_] := Module[{r = Log[-Log[1 - u]]},
  c[[1]] +
   r *
    (c[[2]] +
      r *
       (c[[3]] +
         r *
          (c[[4]] +
            r * (c[[5]] + r * (c[[6]] +
                   r * (c[[7]] + r * (c[[8]] + r * c[[9]])))))))]
```

```
MoroTail[0.975]
```

```
1.95996
```

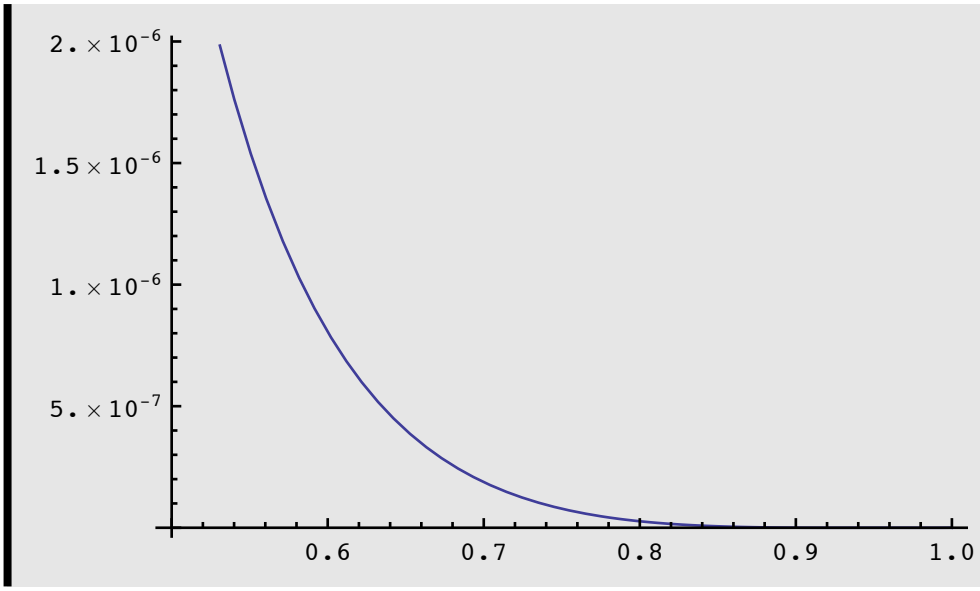
Superficially this looks good everywhere!

```
Plot[{MoroTail[u], QuantileN[u]}, {u, 0.5, 0.9999}]
```

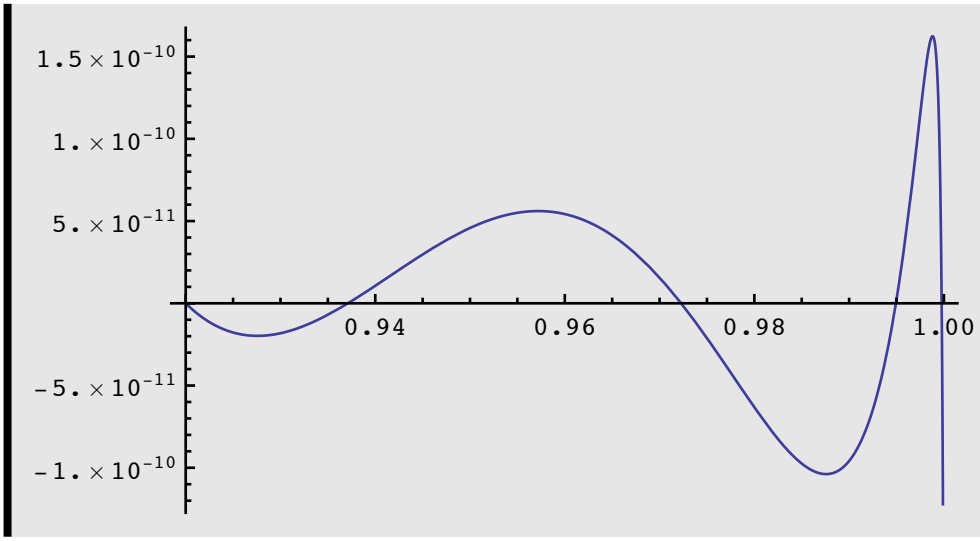


But things do go off!

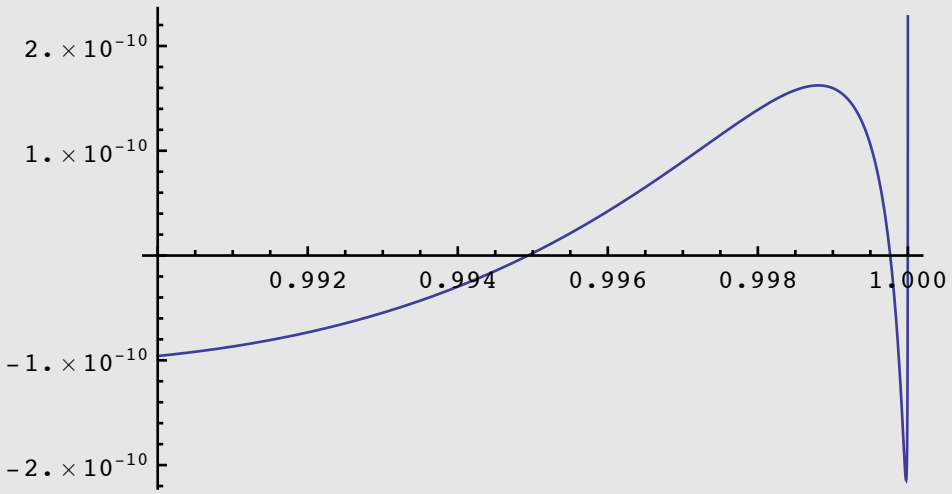
```
Plot[MoroTail[u] - QuantileN[u], {u, 0.5, 0.9999}]
```



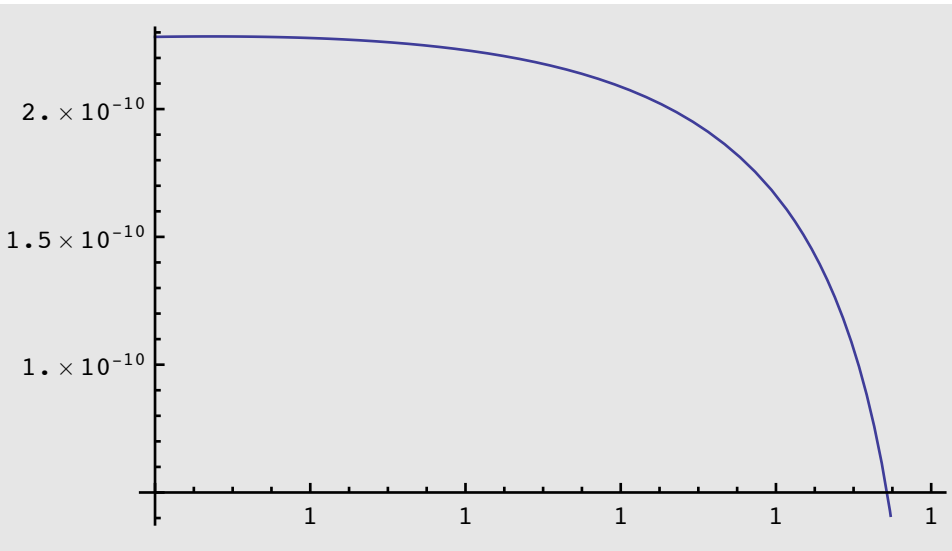

```
Plot[MoroTail[u] - QuantileN[u], {u, 0.92, 0.9999}]
```



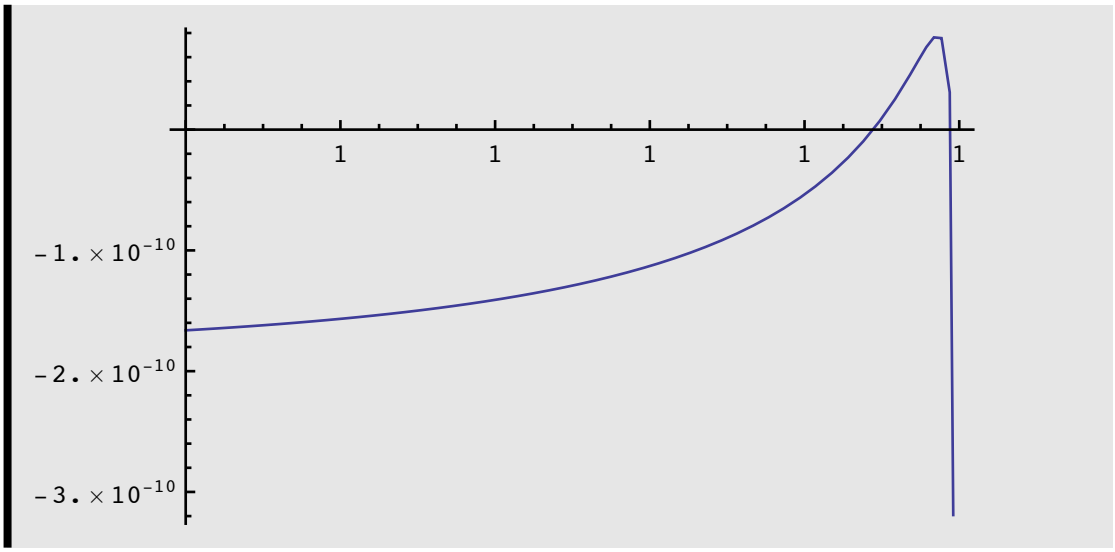
```
Plot[MoroTail[u] - QuantileN[u], {u, 0.99, 0.9999999}]
```




```
Plot[MoroTail[u] - QuantileN[u], {u, 0.9999999, 0.9999999999}]
```



```
Plot[MoroTail[u] - QuantileN[u],
 {u, 0.9999999999, 0.9999999999999}]
```



Joshi has a C++ implementation of this in his web download. It might well be a fun compfin project to try to improve on this without extending comp time too much. Glassermann remarks on going once around Newton-Raphson iteration to refine the method. Clearly we are just doing a combination of taking logs and working out polynomials and ratios of them - there are no special problems and Joshi's C++ implementation is good and straightforward. Here is an extract from his Normals.cpp file with just the relevant bits. We will do an introductory C lecture presently, so do not worry about the details of this for now.

You should note his efficient representation of polynomials, which is good programming practice and something you should always follow.

```
#include <cmath>

// the InverseCumulativeNormal function via the Beasley-Springer/Moro approximation

double InverseCumulativeNormal(double u)

{

    static double a[4]={  2.50662823884,
                        -18.61500062529,
                         41.39119773534,
                        -25.44106049637};


    static double b[4]={-8.47351093090,
                         23.08336743743,
                        -21.06224101826,
                          3.13082909833};



    static double c[9]={0.3374754822726147,
                         0.9761690190917186,
```

```
                        0.1607979714918209,
                        0.0276438810333863,
                        0.0038405729373609,
                        0.0003951896511919,
                        0.0000321767881768,
                        0.0000002888167364,
                        0.0000003960315187};


    double x=u-0.5;
    double r;
    if (fabs(x)<0.42) // Beasley-Springer
    {
        double y=x*x;

r=x*(((a[3]*y+a[2])*y+a[1])*y+a[0])/((((b[3]*y+b[2])*y+b[1])*y+b[0])*y+1.0);
    }
    else // Moro
    {
        r=u;
        if (x>0.0)
            r=1.0-u;
            r=log(-log(r));

r=c[0]+r*(c[1]+r*(c[2]+r*(c[3]+r*(c[4]+r*(c[5]+r*(c[6]+r*(c[7]+r*c[8])))))));


        if (x<0.0)
            r=-r;
    }
    return r;
}
 *
 * Copyright (c) 2002
 * Mark Joshi
 *
 * Permission to use, copy, modify, distribute and sell this
```

```
 * software for any purpose is hereby
 * granted without fee, provided that the above copyright notice
 * appear in all copies and that both that copyright notice and
 * this permission notice appear in supporting documentation.
 * Mark Joshi makes no representations about the
 * suitability of this software for any purpose. It is provided
 * "as is" without express or implied warranty.
*/
```

Doing without Quantiles: Box-Muller; Polar-Marsaglia etc.

Until relatively recently, before people started playing with copulas and pseudo-random numbers, it was not considered necessary to worry about the Quantile functions, and any method for generation Gaussian deviates was used, except the inverse normal CDF! Such methods are used in the Mathematica packages for simulation, for example, and you can see the details of the maths behind it in Numerical Recipes Section 7.2 (Section 7.3.4 in the Third Edition). In summary you start off with TWO uniform deviates $(x_1,\ x_2)$, i.e. you sample from a square. Then there are two routes. The easiest conceptually is to make the change of variables

$$y_1 = \cos(2\pi x_2)\sqrt{-2\log(x_1)} \tag{6}$$

$$y_2 = \sin(2\pi x_2)\sqrt{-2\log(x_1)} \tag{7}$$

If you work out the Jacobian of the transformation (exercise - do it you have not done so before) you find that the y_i are independent Normal variables. So you get two at once which is a payback for starting from a square. The polar version of this is a bit more efficient. Instead of picking uniform deviates in the unit square, we instead pick v_1 and v_2 as the Cartesian coordinates of a random point inside the unit circle around the origin. Then the sum of their squares, R^2, is in fact a uniform deviate, which can be used for x_1, while the angle that $(v_1,\ v_2)$ makes with the horizontal axis can serve as the random angle $2\pi x_2$. You save some trig calls at the price of sample form the unit circle by rejection from a larger square. You still get two back of course.

You can find C++ to work these out in Numerical Recipes. Let's take a look in *Mathematica*,

Here is the first version without doing any optimization:

In[1]:=
```
npbm[mu_, sigma_] :=
 mu + sigma Sqrt[-2 Log[Random[]]] Cos[2 Pi Random[]]
```

Here is the second version using (a) Rejection from a big square, (b) avoiding the trig call, (c) outputting a pair, (d) using the Mathematica compiler.

```
normpair = Compile[{mu, sigma},
   Module[{va = 0.0, vb = 0.0, rad = 2.0, den},
            While[rad >= 1.00,
                    (va = 2.0 * Random[] - 1.0;
      vb = 2.0 * Random[] - 1.0;
                    rad = va * va + vb * vb)];
            den = Sqrt[-2.0 * Log[rad] / rad];
            {mu + sigma * va * den, mu + sigma * vb * den}]];
```

```
Timing[data = Flatten[Table[normpair[0, 1], {i, 1, 10 000}]];]
```

```
{0.042085, Null}
```

```
Mean[data]
```

```
-0.0117209
```

```
Mean[data^2] - Mean[data]^2
```

```
1.02347
```

■ Antithetic combinations

You force any of these things to be symmetric by outputting the deviates of the opposite sign at the same time. Get two for one and zero mean!

```
normpairat = Compile[{mu, sigma},
   Module[{va = 0.0, vb = 0.0, rad = 2.0, den},
            While[rad >= 1.00,
                    (va = 2.0 * Random[] - 1.0;
      vb = 2.0 * Random[] - 1.0;
                    rad = va * va + vb * vb)];
            den = Sqrt[-2.0 * Log[rad] / rad];
            {mu + sigma * va * den, mu + sigma * vb * den,
     mu - sigma * va * den, mu - sigma * vb * den}]];
```

```
normpairat[0, 1]
```

```
{0.53645, 1.30197, -0.53645, -1.30197}
```

```
Timing[data = Flatten[Table[normpairat[0, 1], {i, 1, 10 000}]];]
```

```
{0.04879, Null}
```

```
Mean[data]
```

-4.996×10^{-19}

```
Mean[data^2] - Mean[data]^2
```

```
0.990798
```

N.B.!!! Bailey's Variation for the Student Distribution - not many people know this...

It was only in 1994 that Bailey showed that the T distribution could be sampled by a minor modification to these methods. Here is the efficient polar algorithm:

1. Sample two uniform deviates u and v from [0,1] and let $U = 2\,u - 1,\ V = 2\,v - 1$ (same as before)

2. Let $W = U^2 + V^2$; if $W > 1$ return to step 1 and resample (same as before!);

3. Let $T = U\sqrt{n\left(W^{-2/n} - 1\right)/W}$ and that is one sample from the T with n degrees of freedom.

The last step has the limit $T = U\sqrt{(-2\log W)/W}$ as $n \to \infty$ recovering the Normal case.

Note that as with the transmutation idea you can use the SAME underlying uniform samples and hence assess distributional risk with less Monte Carlo noise. Note that the "other" output is no longer an independent sample so you get half as many samples this way. I will return to the Student t later.

Acklam's formula

We make a quantile function that constructs N[0,1] numbers. You can find a big notebook on my website containing some work I have done in early 2007 looking at quantiles for the normal. Go there if you want to see some detail. By early next term I hope to post a new paper that uses differential equations to define quantile functions. To summarize, despite my fondness for Wichura's algorithm AS241 I will use a compiled version of Peter Acklam's code to convert random numbers on the unit interval to standard normal sample.:

You sould see his web page at:

http://home.online.no/~pjacklam/notes/invnorm/

for links to other implementations, including C/C++, Fortran, Java, VB, Matlab.

```
1 - 0.97575
```

```
0.02425
```

```
Acklam = Compile[{{u, _Real}},
   Module[
    {a = Reverse[{-39.69683028665376, 220.9460984245205,
        -275.9285104469687, 138.3577518672690,
        -30.66479806614716, 2.506628277459239}],
b = Reverse[{-54.47609879822406, 161.5858368580409,
        -155.6989798598866, 66.80131188771972,
        -13.28068155288572}],
     c = Reverse[{-0.007784894002430293, -0.3223964580411365,
        -2.400758277161838, -2.549732539343734,
        4.374664141464968, 2.938163982698783}],
     d = Reverse[{0.007784695709041462, 0.3224671290700398,
        2.445134137142996, 3.754408661907416}]},
    Which[0.02425 ≤ u ≤ 0.97575,
     Module[{v = u - 1 / 2, r = (u - 1 / 2)^2},
      v *
       (a[[1]] +
          r *
           (a[[2]] +
             r * (a[[3]] + r * (a[[4]] + r * (a[[5]] + r * a[[6]])))))) /
        (1 +
          r *
           (b[[1]] +
             r * (b[[2]] + r * (b[[3]] + r * (b[[4]] + r * b[[5]])))))],
     u > 0.97575,
     Module[{q = Sqrt[-2 * Log[1 - u]]},
      -(c[[1]] +
          q *
           (c[[2]] +
             q * (c[[3]] + q * (c[[4]] + q * (c[[5]] + q * c[[6]])))))) /
       (1 + q * (d[[1]] + q * (d[[2]] + q * (d[[3]] + q * d[[4]]))))],
     True,
     Module[{q = Sqrt[-2 * Log[u]]},
      (c[[1]] +
         q *
          (c[[2]] +
            q * (c[[3]] + q * (c[[4]] + q * (c[[5]] + q * c[[6]])))))) /
       (1 + q * (d[[1]] + q * (d[[2]] + q * (d[[3]] + q * d[[4]]))))]]];
```

```
QuantileN[u_] := Sqrt[2] InverseErf[2 u - 1]
```

```
QuantileN[0.975]
```

```
1.95996
```

```
SeedRandom[100];
Timing[dataone = Table[QuantileN[Random[]], {10 000}];]
```

```
{2.1286, Null}
```

```
SeedRandom[100];
Timing[datatwo = Table[Acklam[Random[]], {10 000}];]
```

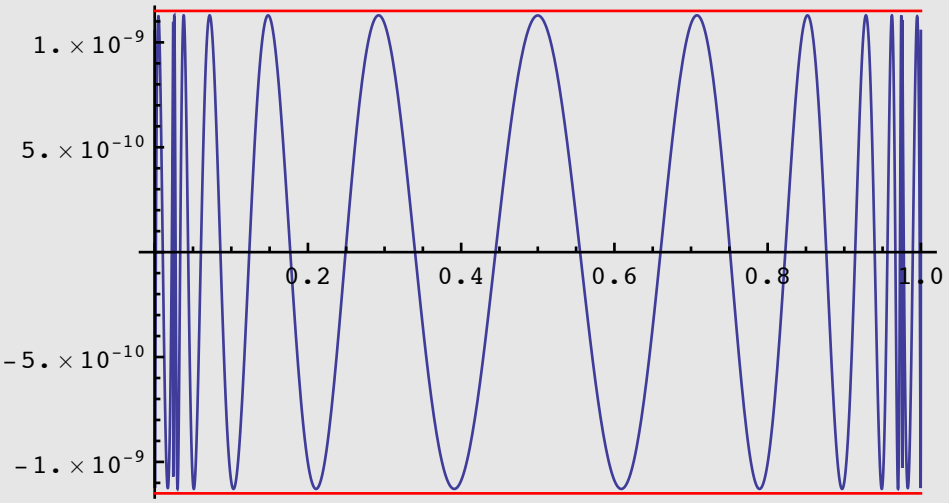
```
{0.043215, Null}
```

```
Max[Abs[Flatten[datatwo / dataone] - 1]]
```

1.12884×10^{-9}

So that is good and fast. Note that this is the first level of Acklam's method. Level two goes once round a high-order Newton-Raphson type scheme to get the error down to machine precision (at least with "double precision).

```
Plot[Acklam[u] / QuantileN[u] - 1, {u, 0.000001, 0.999999},
 PlotPoints → 300,
 Epilog → {RGBColor[1, 0, 0],
   Line[{{0, 1.15 × 10^-9}, {1, 1.15 × 10^-9}}],
    Line[{{0, -1.15 10^-9}, {1, -1.15 10^-9}}]}]
```



Wichura's (1988) Normal Quantile Function

Here it is in *Mathematica*, based on the algorithm in Wichura's 1988 paper on "Algorithm AS241" The Percentage Points of the Normal Distribution. Applied Statistics, 37, 477-484, 1988. This is a three stage algorithm. For $0.075 \leq u \leq 0.925$, a rational approximation of type (7, 7) is employed, with a coordinate reflection. For

$u > 0.925$, variables are changed to $r = \sqrt{-\text{Log}[1-u]}$ and then a choice is made between two further rational approximations of type (7, 7) depending on whether $r \leq 5$ or $r > 5$. C source for this may be found at: http://mpa.itc.it/markus/grass63progman/as241_8c-source.html

```
WichuraQuantile[u_] :=
 Module[{a = {3.3871328727963666080, 133.14166789178437745,
     1971.5909503065514427, 13731.693765509461125,
     45921.953931549871457, 67265.770927008700853,
     33430.575583588128105, 2509.0809287301226727},
   b = {42.313330701600911252, 687.18700749205790830,
     5394.1960214247511077, 21213.794301586595867,
     39307.895800092710610, 28729.085735721942674,
     5226.4952788528545610},
   c = {1.42343711074968357734, 4.63033784615654529590,
     5.76949722146069140550, 3.64784832476320460504,
     1.27045825245236838258, 0.24178072517745061177O,
     0.022723844989269184583 3, 0.000774545014278341407640},
   d = {2.05319162663775882187, 1.67638483018380384940,
     0.68976733498510000455O, 0.14810397642748007459O,
     0.015198666563616457196 6, 0.000547593808499534494600,
     1.05075007164441684324 * 10^(-9)},
   e = {6.65790464350110377720, 5.46378491116411436990,
     1.78482653991729133580, 0.29656057182850489123O,
     0.026532189526576123093 O, 0.001242660947387078438 60,
     0.000027115555687434875 7 815,
     2.01033439929228813265 * 10^(-7)},
   f = {0.599832206555887937690, 0.136929880922735805310,
     0.014875361290850614852 5,
     0.000786869131145613259100,
     1.84631831751005468180 * 10^(-5),
     1.42151175831644588870 * 10^(-7),
     2.04426310338993978564 * 10^(-15)}
  },
  If[0.075 ≤ u ≤ 0.925,
      Module[{v = u - 1 / 2, r},
       r = 180625 / 10^6 - v * v;

    v *
     (a[[1]] +
        r *
         (a[[2]] +
           r *
            (a[[3]] + r * (a[[4]] +
                  r * (a[[5]] + r * (a[[6]] +
                        r * (a[[7]] + r * a[[8]])))))))) /
      (1 +
        r *
         (b[[1]] +
           r *
            (b[[2]] + r * (b[[3]] +
                  r * (b[[4]] + r * (b[[5]] +
```

```
                        r * (b[[6]] + r * b[[7]])))))))],
  If[u < 1 / 2, r = u, r = 1 - u ]; r = Sqrt[-Log[r]];
  If[r <= 5,
   (r = r - 16 / 10;
    Sign[u - 1 / 2]
     (c[[1]] +
        r *
         (c[[2]] +
           r * (c[[3]] + r * (c[[4]] +
                  r * (c[[5]] + r * (c[[6]] +
                         r * (c[[7]] + r * c[[8]])))))))) /
      (1 +
        r *
         (d[[1]] +
           r * (d[[2]] + r * (d[[3]] +
                  r * (d[[4]] + r * (d[[5]] +
                         r * (d[[6]] + r * d[[7]])))))))),
   (r = r - 5; Sign[u - 1 / 2]
     (e[[1]] +
        r *
         (e[[2]] +
           r * (e[[3]] + r * (e[[4]] +
                  r * (e[[5]] + r * (e[[6]] +
                         r * (e[[7]] + r * e[[8]])))))))) /
      (1 +
        r *
         (f[[1]] +
           r * (f[[2]] + r * (f[[3]] +
                  r * (f[[4]] + r * (f[[5]] +
                         r * (f[[6]] + r * f[[7]])))))))
   )]
 ]]
```

Evaluation

```
WichuraQuantile[0.975]
```

```
1.95996
```

Plot

```
Plot[WichuraQuantile[u], {u, 0.00001, 0.99999}];
```

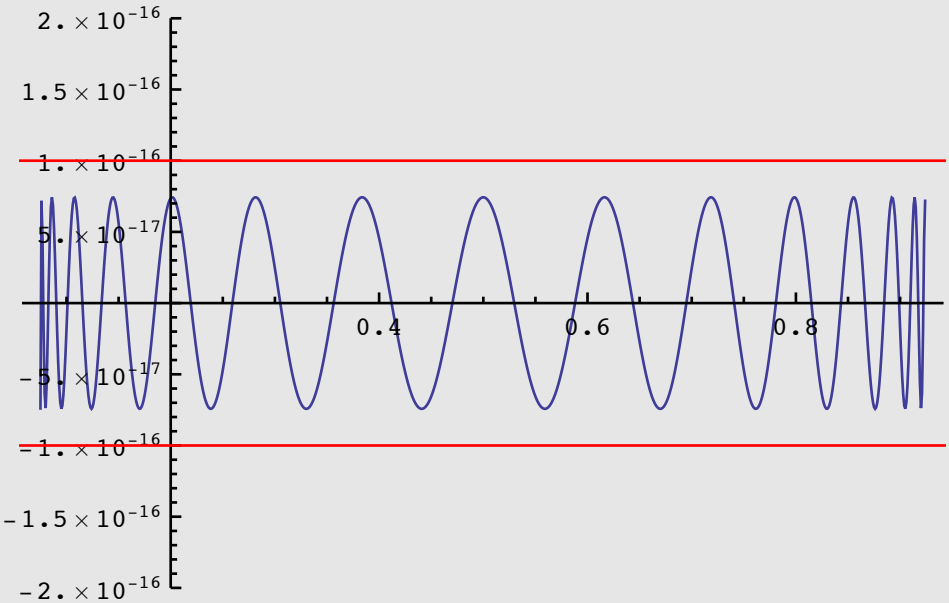
Relative Error against *Mathematica*'s internal Erf-based Quantile

So we just test the claim that the error is of order machine precision. The red lines are drawn at 10^{-16} and the relative error is less than this down to less than $u = 10^{-22}$.

```
QuantileN[u_] := Sqrt[2] InverseErf[2 u - 1]
```

```
edata = Table[{u, WichuraQuantile[u] / N[QuantileN[u], 200] - 1},
   {u, 10^(-6), 1 - 10^(-6), 1 / 1000}];
```

```
ListPlot[edata, PlotJoined → True,
 Epilog → {RGBColor[1, 0, 0], Line[{{0, 10^-16}, {1, 10^-16}}],
   Line[{{0, - 10^-16}, {1, -10^-16}}]},
 PlotRange → {-2 10^(-16), 2 × 10^-16}]
```



You cannot make plots like this in C/C++ unless you have invoked precision beyound double.

Modern Rejection: Ratio of Uniforms

For fast simulation WITHOUT regard to the quantile structure, you should see the rejection techniques discussed in

A.

NR III Section 7.3

B.

A.J. Kinderman and J,F, Monahan, 1977, Computer Generation of Random Variables Using the Ratio of Uniform Deviates, ACM Transactions on Mathematical Software, Vol. 2, pp 257-260.

C.

J.L.Leva, 1992, A Fast Normal Random Number Generator, ACM Transactions on Mathematical Software, Vol. 18, 4, , pp 449-453.

These are fast but have a difficult interaction with deterministic space-filling methods and copula methods. The latter relies very explicitly on having quantiles for the construction of marginals.

Student T Distribution Quantile Functions - the closed form cases

What happens if we replace the Normal by the fatter-tailed Student for example? Strangely the Quantile function for the Student had not received much detailed attention till I had a play with it a couple of years ago. The details are given in a paper I published in the Journal of Computational Finance in 2006. I have given a power series for the middle region and a tail series as well and these can be used in much the same was as the BSM model. I now know how to tidy this up and will deal with it next lecture. Given that there is a parameter n you cannot write down one nice rational approximation for example. There are however some amusing special values of n when you can actually directly invert the CDF exactly. Some of you will know about the Cauchy distribution, which is the $n = 1$ Student distribution.

We can give special methods for the Student T distribution CDF when the degrees of freedom n is an even integer. A paper by E. Platen and a colleague in Applied Mathematical Finance 2006 emphasized role of T_4 case as being maximum likelihood estimate of distribution of world index returns. So $n = 4$ is of great practical importance. It turns out that solving the problem when n is even integer corresponds to solving a sparse polynomial of degree $n - 1$. Hence $n = 2, 4$ can be solved exactly.

For the even case: First we have an iterative specification of the CDF

```
Clear[a]
```

```
a[0, n_] := Gamma[(n + 1) / 2] / Gamma[n / 2] / Sqrt[n Pi];
a[k_, n_] := a[k, n] = (n - 2 k) / n / (2 k + 1) a[k - 1, n];

TCDF[n_, x_] := 1 / 2 +
    x * Sum[a[p, n] x^(2 p), {p, 0, n / 2 - 1}] /
      (1 + x^2 / n)^(1 / 2 (n - 1));
```

```
{Simplify[TCDF[2, x]], Simplify[D[TCDF[2, x], x]]}
```

$$\left\{\frac{1}{2}\left(\frac{x}{\sqrt{x^2+2}}+1\right), \frac{1}{\left(x^2+2\right)^{3/2}}\right\}$$

```
{Simplify[TCDF[4, x]], Simplify[D[TCDF[4, x], x]]}
```

$$\left\{\frac{x^3+6x+\left(x^2+4\right)^{3/2}}{2\left(x^2+4\right)^{3/2}},\ \frac{12}{\left(x^2+4\right)^{5/2}}\right\}$$

Solve TCDF[n, x] = u , a polynomial problem! Let $a = 4u(1-u)$ and $p = n + x^2$

```
P[n_, x_] := x * Sum[a[p, n] x^(2 p), {p, 0, n / 2 - 1}];
R[n_, p_] := Expand[PowerExpand[
    4 * n^(n - 1) (P[n, x]) ^2 + p^(n - 1) (a - 1) /. x → Sqrt[p - n]]] ;
TraditionalForm[Table[{2 k, R[2 k, p] == 0}, {k, 1, 7}]]
```

$$\begin{pmatrix}
2 & a\,p-2=0\\
4 & a\,p^3-12\,p-16=0\\
6 & a\,p^5-135\,p^2-\frac{1215\,p}{4}-\frac{2187}{2}=0\\
8 & a\,p^7-2240\,p^3-7168\,p^2-35\,840\,p-204\,800=0\\
10 & a\,p^9-\frac{196\,875\,p^4}{4}-\frac{1\,640\,625\,p^3}{8}-\frac{10\,546\,875\,p^2}{8}-\frac{615\,234\,375\,p}{64}-\frac{2\,392\,578\,125}{32}=0\\
12 & a\,p^{11}-1\,347\,192\,p^5-6\,928\,416\,p^4-54\,561\,276\,p^3-484\,989\,120\,p^2-4\,583\,147\,184\,p-44\,9\\
14 & a\,p^{13}-\frac{353\,299\,947\,p^6}{8}-\frac{17\,311\,697\,403\,p^5}{64}-\frac{40\,393\,960\,607\,p^4}{16}-\frac{848\,273\,172\,747\,p^3}{32}-\frac{18\,893\,357\,029\,365\,p^2}{64}-\frac{872}{}
\end{pmatrix}$$

and so on. We get interesting family of sparse polynomials. Almost half the coefficients are missing. Two exact solutions $n = 2, 4$, and $n = 6, 8, 10, \ldots$ easy by Newton-Raphson! So combining with Cauchy we can do $n = 1, 2, 4$ exactly. Here are the formulae for the Quantile functions:

■ n=1

```
Q[u_, 1] := Tan[Pi (u - 1 / 2)]
```

```
TraditionalForm[Q[u, 1]]
```

$$\tan\left(\pi\left(u-\frac{1}{2}\right)\right)$$

■ n=2

```
Q[u_, 2] := (2 u - 1) / Sqrt[2 u (1 - u)]
```

```
TraditionalForm[Q[u, 2]]
```

$$\frac{2u-1}{\sqrt{2}\ \sqrt{(1-u)\,u}}$$

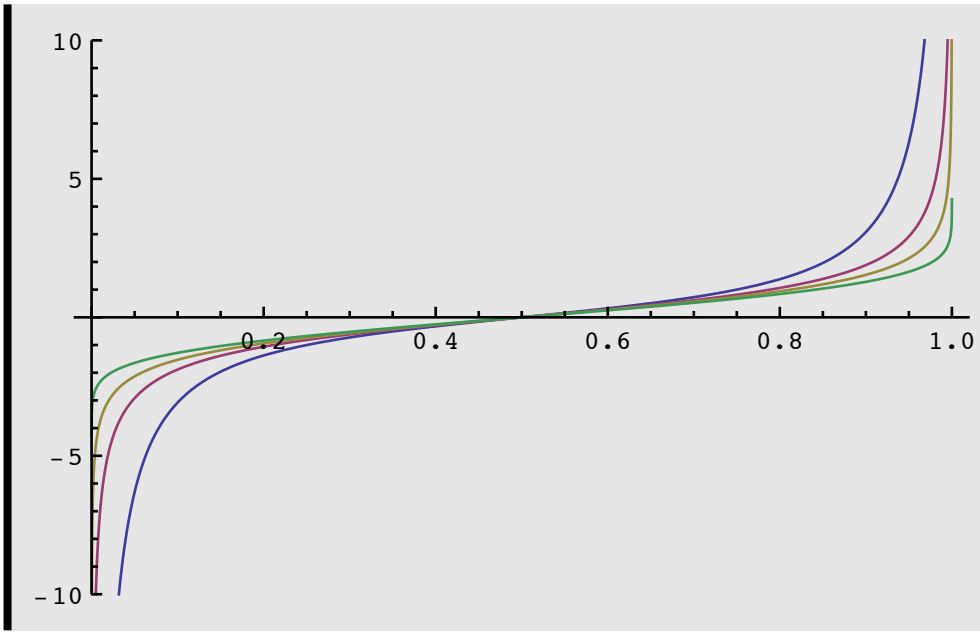
■ **n=4**

```
Q[u_, 4] := Module[{a = 4 u (1 - u), p},
  p = 4 / Sqrt[a] Cos[1 / 3 ArcCos[Sqrt[a]]];
  Sign[u - 1 / 2] Sqrt[p - 4]]
```

$$\operatorname{sgn}\left(u-\frac{1}{2}\right)\sqrt{\left(\left(2\cos\left(\frac{1}{3}\cos^{-1}\left(2\sqrt{(1-u)\,u}\right)\right)\right)\Big/\left(\sqrt{(1-u)\,u}\right)-4\right)} \tag{8}$$

Here they are together with the normal, clearly demonstrating the increasingly fat-tailed nature. n=4 has infinite kurtosis, n=2 has infinite variance.

```
Plot[{Q[u, 1], Q[u, 2], Q[u, 4], QuantileN[u]},
 {u, 0.00001, 0.99999}, PlotRange → {-10, 10}]
```



These are on Wikipedia now!

Doing it better? Other Distributions?

In the case of the normal people have invested a lot of effort into one-off composite rational approximations. This distribution has no free parameters so once the effort has been made that is it. Or it was until GPUs came along, where examples may be processed in blocks and the block will take as long as the slowest branch. So Formula like Acklam's and AS241 can be dominated by computation time for the longer branch. Also, what do we do about all those other distributions with parameters?

So this is where we divert to some recent papers by your prof, on the theory of "Quantile Mechanics".....

What we do is to forget all about the messy business of functionally inverting a CDF. Instead we go back to basics and represent these things in terms of the solutions to differential equations. Off to the EJAM paper.