Refinement of the Normal Quantile

A benchmark Normal quantile based on recursion, and an appraisal of the Beasley-Springer-Moro, Acklam, and Wichura (AS241) methods

William Shaw, Financial Mathematics Group
King's College London
william.shaw@kcl.ac.uk

V1.2 Working Paper February 20 2007

Summary

An investigation of common techniques for approximating the Normal distribution Quantile function is presented. The high-precision arithmetic and rational approximation packages built in to *Mathematica* are used for this investigation, but we also present a benchmark easily used in other languages, based on the power series representation of the Normal Quantile developed recently by Steinbrecher, and using a simpe non-linear recursive definition of its coefficients to provide a benchmark. Within an arbitrary precision environment this can be used to provide as much precision as is needed, and our working benchmark tool has relative accuracy better than 10^{-19} on $0.5 < u < 0.99$, and better than 10^{-37} on $0.5 < u < 0.98$. A simple version in C-style C++ is provided for benchmarking/education purposes with error order 10^{-15} on $0.005 < u < 0.995$. Using this benchmark with *Mathematica*'s MiniMaxApproximation functions allows rational functions of diverse complexity to be constructed with relatively little effort. In this version of the note we explore the Beasley-Springer-Moro (BSM), Wichura "AS241" (1988) and Acklam algorithms, paying particular attention to the main rational approximations employed within a central region $|u - 0.5| < a$. We find that the BSM rational approximation is sub-optimal with regard to both absolute and relative error on the interval on which it is applied, and that an approximation of the same degree (3,4) as BSM exists whose maximum relative error is less than one-fifth of the BSM approach. This may be applied by a small adjustment to the coefficients. Within the same BSM interval an approximations of degree (4,5) and (5,6) are found with maximum relative errors of $1.5 * 10^{-11}$ and $1.0 * 10^{-13}$ respectivcely. For the Acklam approach, which is degree (5,5) but on a larger region, we find that *Mathematica* generates a (5,5) rational approximation that is extremely close to Acklam's formula and a similar relative error. We can also find a (6,6) variation on Acklam's method with a relative error less than $2.5 * 10^{-11}$ on the central region, and a (5,6) variation on Acklam's method with a relative error less than $1.7 * 10^{-10}$ on the central region. Wichura's AS241 correspondingly uses a (7,7) variation on a different interval to obtain a relative error that is of order machine precision, and we are able to regenerate an expression that is also very close to that of AS241 with a similar error. Within these central regions it is clear that the accuracy just increases with the degree of the rational approximation. Within environments limited to standard machine precision, AS241 is the clear winner on accuracy grounds, though if pure speed is needed, a revised form of BSM can be considered, with rational approximations of the Acklam class or similar degree being a compromise. We also show how the central region for the Acklam formula may be extended, and the error more than halved, by combining a rational approximation with a polynomial of similar average length. In the tail regions we find good behaviour of the AS241 and Acklam formulae, though Moro's approach has issues in the deep tails.

1 Introduction

The purpose of this note is to explore the various approaches to working out the Normal quantile function, for efficient Monte Carlo simulation. We will exploit Mathematica's high-precision arithmetic tools and rational approximations packages in order to try to obtain further optimization of fast methods for approximating the Normal quantile function used within various types of financial simulation. It is very much a "work in progress" and will be updated over the next few months as I learn new ways of doing further optimization. There is no claim that results discussed below are actually absolutely optimal, rather they represent some improvements and other options that I think are worth mentioning as they provide some benefit. Comments and suggestions for improvement are welcome. Furthermore, given that this is a *Mathematica* project, there is, despite the high-precision nature of this environment, a small desire to see the results so far checked in another symbolic environment exists. So I would be pleased to hear from any Maple etc. uses to see if they can verify these results or indeed improve on them. I hope the ideas developed here will be improved on and applied to other quantile construction problems. I will add discussions of

(a) hybrid polynomial-rational methods;
(b) tail series;
(c) other CDFs

in due course. First some reminders.

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. The inverse CDF: $F^{-1}(u) = Q_F(u)$ 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. Readers should see Jaeckel (2002), who provides a detailed discussion of why quantile functions are a much better idea than methods of Box-Muller type. Many financial applications make use of a composite of methods, that I shall refer to as BSM, developed by Beasley and Springer (1977) and Moro in his RISK article (1995). They are summarized by Glasserman (2004) in Section 2.3.2, who also points out that the BSM approach can be made much more accurate by going once around a Newton-Raphson loop. The BSM method has been implemented in various systems supporting financial modelling in C++, including Joshi (2004), who has it on his web site, and is also available in QuantLib. QuantLib also cites an improvement by Acklam, who gives a very detailed discussion of his methods on his web site at:

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

We shall also explore Acklam's approach here. He uses, for example, a higher order rational approximation than BSM in a central region, and we shall look at this issue in particular. Jaeckel (2002) provides an Acklam implementation. The other method we shall consider is Wichura's (1988) AS241 algorithm, which in my view has been wrongly neglected by the mathematical finance community, and provides machine precision. It should also be noted that "Acklam's method" actually comprises two stages - a pair of rational approximations for work not requiring machine precision, together with a Halley refinement (i.e. Newton-Raphson with a quadratic rather than a line) that gets to machine precision.

■ Error Measures

One needs to distinguish between absolute and relative error when comparing algorithms. The BSM discussion focused on absolute error while Acklam focuses on relative error.

■ References

P. J. Acklam, An algorithm for computing the inverse normal cumulative distribution function, http://home.online.no/~pjacklam/notes/invnorm/

J.D Beasley and S.G. Springer, Algorithm AS111: The percentage points of the Normal Distribution, Applied Statistics, Vol. 26, No. 1., 1977, pp 118-121.

Carlitz, L., The inverse of the error function, Pacific Journal of Mathematics, 13, 2, 459-470, 1963.

P. Glasserman, Monte Carlo Methods in Financial Engineering, Springer, 2004.

P. Jaeckel, Monte Carlo Methods in Finance, Wiley, 2002.

M. Joshi, C++ Design Patterns and Derivatives Pricing, Cambridge, 2004.

G. Marsaglia, M.D. MacLaren, T.A. Bray, A Fast Procedure for Generating Normal Random Variables, Commun. Ass. Comp. Mach., 7, 4-10, 1964.

B. Moro, The full monte, RISK 8 (Feb): 57-58.

G. Steinbrecher, Contribution of full iterative specification of the series for the inverse error function: http://functions.wolfram.com/GammaBetaErf/InverseErf/06/01/0002/ (added March 2002).

Wichura, M.J., Algorithm AS 241: The Percentage Points of the Normal Distribution. Applied Statistics, 37, 477-484, 1988.

2 Benchmark Quantile Functions

The first thing we need to establish is: How do we know when we have a good answer? In estimating the error properties of any model of the Normal Quantile one needs a benchmark, and preferably one that may be used in a diverse set of computational environments. In *Mathematica* we have an easy life. Here is the Normal Quantile function expressed in terms of Mathematica's InverseErf function:

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

This, as it stands, is not efficient enough for fast numerical computation, but it is a very useful benchmark. What else can we do, and how indeed would we check the answers from *Mathematica*'s **InverseErf**? How would be check for accuracy in any other computer language where we do not have any fast/slow/precise/imprecise implementation of the quantile? The following result established recently by G. Steinbrecher is *very* useful. We shall use it here to construct an alternative high precision benchmark.

■ Steinbrecher's recurrence formula and a super-precision Normal Quantile

In 2002 G. Steinbrecher posted the following result to functions.wolfram.com.

$$\text{erf}^{-1}(z) = \sum_{k=0}^{\infty} \frac{c_k \left(\frac{\sqrt{\pi}\, z}{2}\right)^{2k+1}}{2k+1}$$
$$c_0 = 1 \tag{2}$$
$$c_k = \sum_{m=0}^{k-1} \frac{c_m\, c_{k-m-1}}{(m+1)(2m+1)}$$

The observation that such a *simple* recursion for the series exists, and the method of its derivation are new and ingenious. This result was certainly not known to Carlitz, whose 1963 paper looked at series for the formal inverse of erf. It was established by Steinbrecher by doing a series solution of the differential equation for the inverse. (Personal communication from GS.) Let's implement this using *Mathematica*'s approach to efficient recursion.

```
Clear[q];
q[0] = 1;
q[k_] :=
 q[k] = Sum[q[m] * q[k - 1 - m] / (m + 1) / (2 m + 1), {m, 0, k - 1}]
```

The array could be populated by calling value for a length we want to employ, but to avoid large recursions we just fill out this array in order. The following command takes a substantial time, but the results may be stored on disk or indeed held in fast memory.

```
max = 1000;
```

```
Timing[Do[dummy = q[j], {j, 1, max}]]
```

{925.611 Second, Null}

Having done some work to compute this, we shall save it.

```
Save["qdata", q];
```

This may be reloaded as necessary:

```
<< qdata;
```

```
{q[0], q[1], q[2], q[3]}
```

$\left\{1, 1, \frac{7}{6}, \frac{127}{90}\right\}$

```
% // N
```

{1., 1., 1.16667, 1.41111}

```
q[1000] // N
```

3.31126×10^{104}

Next we want to generate a table of ratios for use in an optimized algorithm:

```
algpoly = Table[q[k] / (2 k + 1), {k, 0, max}];
```

```
algpoly[[{1, 2, 3, 4}]]
```

$$\left\{1, \frac{1}{3}, \frac{7}{30}, \frac{127}{630}\right\}$$

```
% // N
```

{1., 0.333333, 0.233333, 0.201587}

```
algratios = RotateLeft[algpoly] / algpoly;
algratios = Drop[algratios, -1];
```

```
algratios[[{1, 2, 3}]] // N
```

{0.333333, 0.7, 0.863946}

```
Length[algratios]
```

1000

```
Last[algratios] // N
```

1.27189

We convert to numerical values in high precision, and use 200 sig figs for internal use. This may seem excessive, but the error in a truncated form of this series can get so small, way below machine precision, that it is necessary in order to get a grip on measuring it.

```
polydata = Table[N[q[k], 200], {k, 0, max}];
```

```
Length[polydata]
```

1001

Here is a crude and not very efficient implementation (don't even think of working out polynomials like this is real applications - this is just to get an idea of how it works!):

```
SuperPolyQuantile[u_, data_] := Module[{v = Sqrt[Pi] (u - 1 / 2),
   len = Length[data], w = Pi * (u - 1 / 2) ^ 2},
  Sqrt[2] * v * Sum[data[[k]] * w^(k - 1) / (2 k - 1), {k, 1, len}]]
```

Here is a variaton that stops at a fixed point:

```
SuperPolyQuantileShort[u_, data_, end_] :=
 Module[{v = Sqrt[Pi] (u - 1 / 2),
   len = Length[data], w = Pi * (u - 1 / 2) ^ 2},
  Sqrt[2] * v * Sum[data[[k]] * w^(k - 1) / (2 k - 1), {k, 1, end}]]
```

```
SuperPolyQuantile[0.975, polydata]
```

1.95996

■ Optimal evaluation to a tolerance

This is a faster scheme that keeps going until a tolerance is reached. Note how *simple* this algorithm is once the ratios have been precomputed. This algorithm is modelled on Marsaglia's for the forward Normal CDF.

```
SbNormalQuantile[u_, ratios_, tol_] :=
 Module[{b = Sqrt[Pi] * (u - 1 / 2), q , i = 0, s, t = 0, end = tol},
  s = b; q = b * b;
  While[s - t > 10 ^ (-end),
   (t = s; b *= q * ratios[[i += 1]]; s = t + b)];
  Sqrt[2] *
   s]
```

In the following plot we evaluate out to 0.99 stopping the iteration onces the last term added is less than $10^{-10.13}$. This gets us down to order 10^{-9} relative accuracy on $0.5 \le u \le 0.99$ with this *one* function.

```
N[SbNormalQuantile[975 / 1000, algratios, 16], 16]
```

1.959963984540053

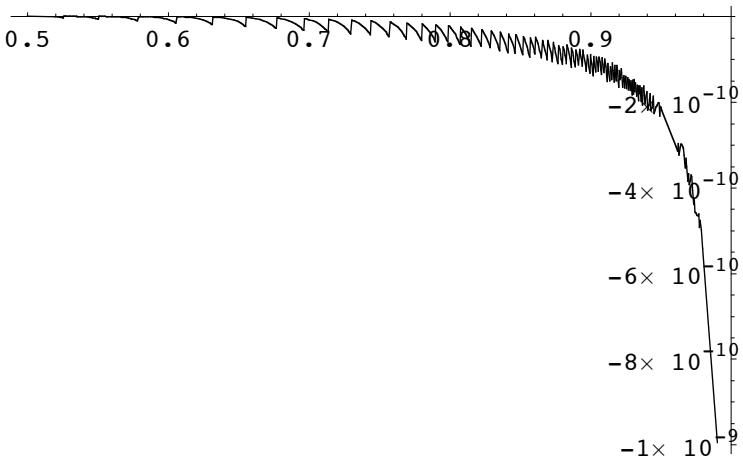
```
N[QuantileN[975 / 1000], 16]
```

1.959963984540054

```
N[QuantileN[998 / 1000], 16]
```

2.878161739095484

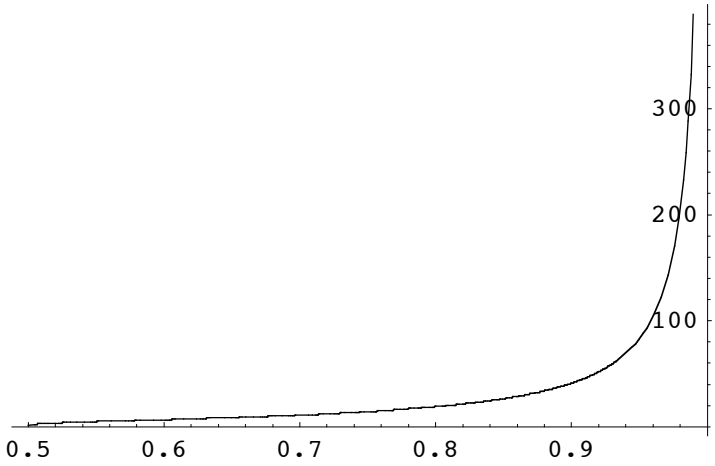
```
Plot[SbNormalQuantile[u, algratios, 10.13] / QuantileN[u] - 1,
  {u, 0.5, 0.99}, PlotRange → All];
```



For assessment of amount of work done we will have a variation from which we can extract the number of terms taken.

```
SbNormalQuantileReport[u_, ratios_, tol_] :=
 Module[{b = Sqrt[Pi] * (u - 1 / 2), q , i = 0, s, t = 0},
  s = b; q = b * b;
  While[s - t > 10 ^ (-tol),
   (t = s; b *= q * ratios[[i += 1]]; s = t + b)];
  {Sqrt[2] * s, i}]
```

```
Plot[SbNormalQuantileReport[u, algratios, 10.13][[2]],
  {u, 0.5, 0.99}, PlotRange → All];
```



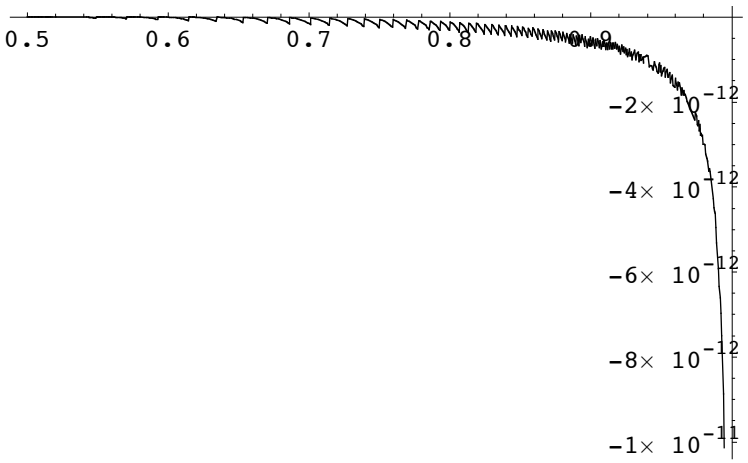

```
iterreport =
  Table[SbNormalQuantileReport[u, algratios, 10.13][[2]],
   {u, 0.5, 0.99, 0.0001}];
```

```
Mean[iterreport] // N
```

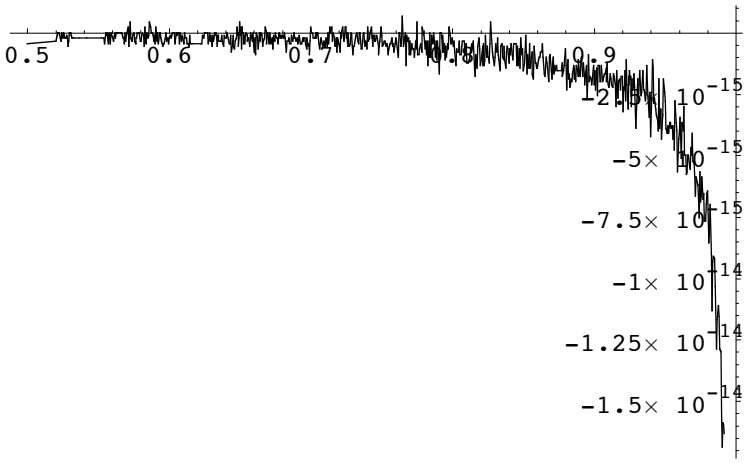
30.1236

So this is not in fact taking, on average, a great deal of calculation to get errors of order 10^{-9} or better on a large sub-interval - my guess is around three times the effort of Acklam's central approximation. Although this is doing more work than the traditional rational approximations, note that we have not worked very hard on optimization, nor on advanced methods for summing the series, and that this series works in a complex disk - not just on a real interval. Furthermore we can just dial up the precision when we need to. To see the real power of the method, we note that with the terms generated so far we can get a relative error better than 10^{-11} on $0.5 \le u \le 0.995$ with this single function, or better than $2 * 10^{-14}$ on $0.5 \le u \le 0.992$!

```
Plot[SbNormalQuantile[u, algratios, 12.4] / QuantileN[u] - 1,
  {u, 0.5, 0.995}, PlotRange → All];
```

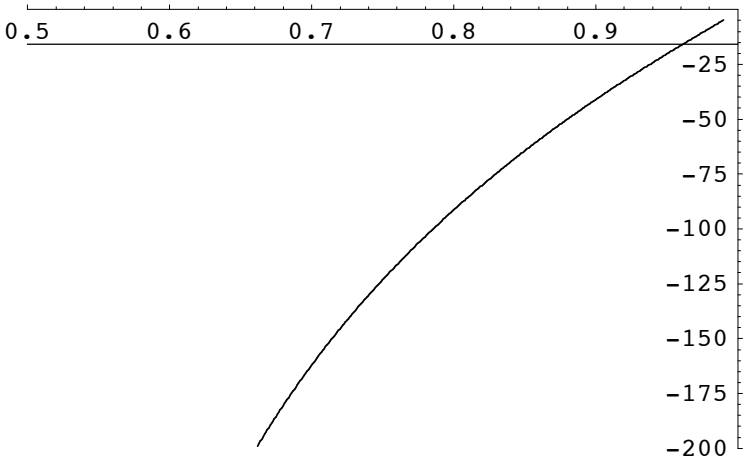



```
Plot[SbNormalQuantile[u, algratios, 15] / QuantileN[u] - 1,
  {u, 0.5, 0.992}, PlotRange → All];
```



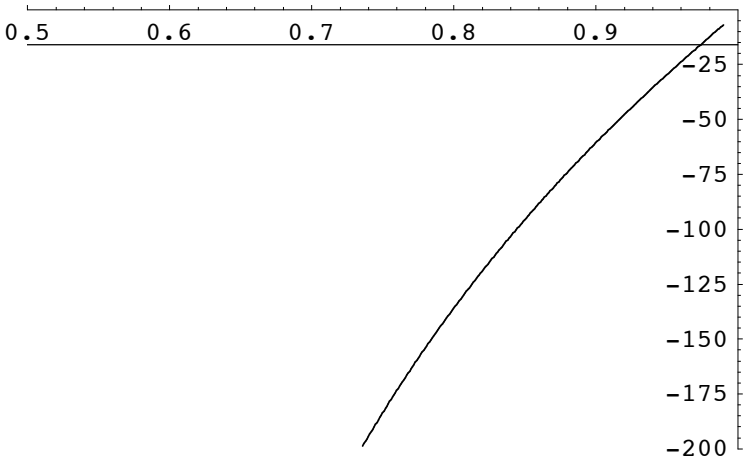
Let's see how much precision we can get. Going back to the full series, here is the comparison with the internal quantile over a larger region than the Acklam range. In this first graph we plot the log to the base 10 of the absolute error over the range for which we can work it out. Note that this graph is showing the difference as below 10^{-200} for u less than 0.65! The superimposed line is at -16, i.e. the a rough estimate of the machine precision limit.

```
edata = Table[{u, SuperPolyQuantileShort[u, polydata, 200] -
     N[QuantileN[u], 200]},
   {u, 1 / 2 + 1 / 1000, 99 / 100, 1 / 1000}];
rawlogs = Map[{#[[1]], Log[10, Abs[#[[2]]]]} &, edata];
plotdata = Select[rawlogs, NumberQ[#[[2]]] &];
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.5, 1}, {-200, 0}},
  Epilog → Line[{{0.5, -16}, {1, -16}}]];
```

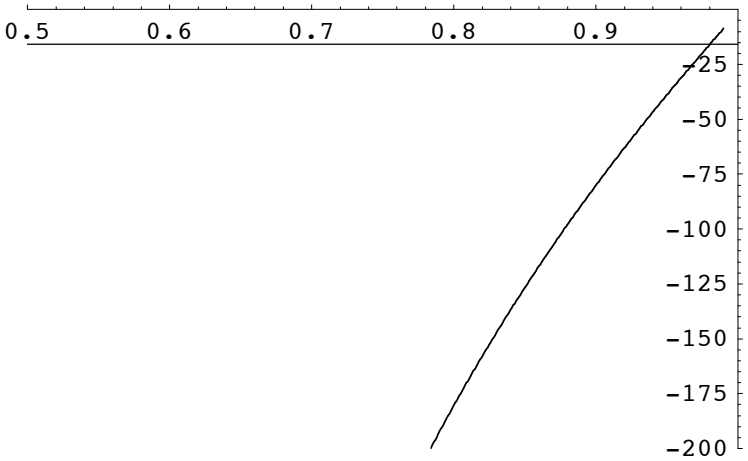


Here is the polynomial taken to 300 terms:

```
edata = Table[{u, SuperPolyQuantileShort[u, polydata, 300] -
     N[QuantileN[u], 200]},
   {u, 1 / 2 + 1 / 1000, 99 / 100, 1 / 1000}];
rawlogs = Map[{#[[1]], Log[10, Abs[#[[2]]]]} &, edata];
plotdata = Select[rawlogs, NumberQ[#[[2]]] &];
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.5, 1}, {-200, 0}},
  Epilog → Line[{{0.5, -16}, {1, -16}}]];
```

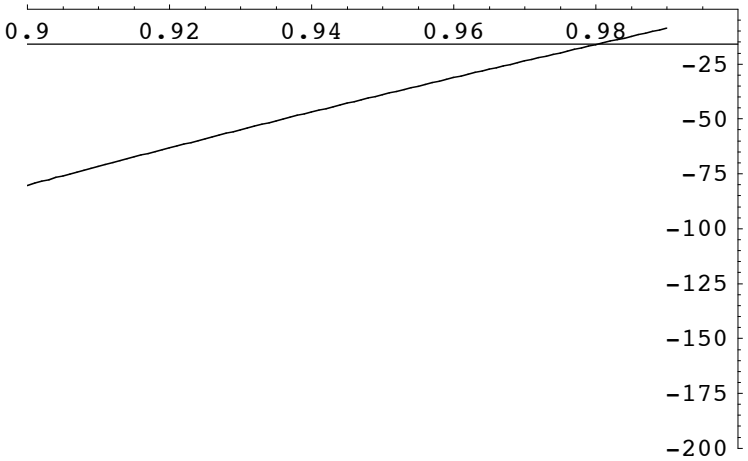


```
edata = Table[{u, SuperPolyQuantileShort[u, polydata, 400] -
     N[QuantileN[u], 200]},
   {u, 1 / 2 + 1 / 1000, 99 / 100, 1 / 1000}];
rawlogs = Map[{#[[1]], Log[10, Abs[#[[2]]]]} &, edata];
plotdata = Select[rawlogs, NumberQ[#[[2]]] &];
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.5, 1}, {-200, 0}},
  Epilog → Line[{{0.5, -16}, {1, -16}}]];
```

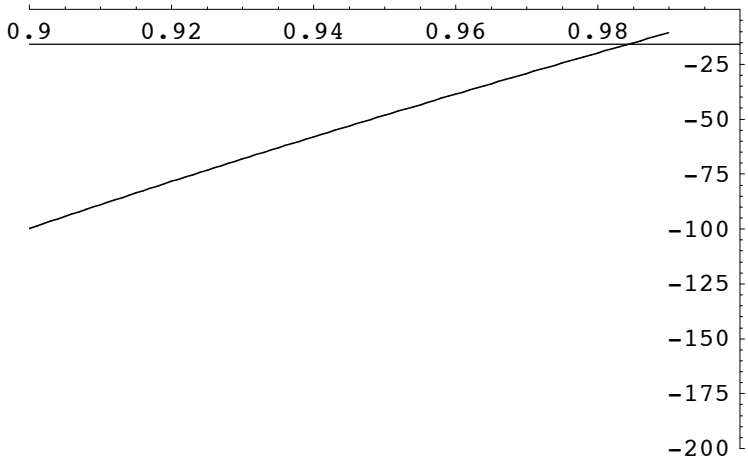


Now we need to zoom in and make sure our polynomial is good enough to rather better than machine precision deep into the tail.

```
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.9, 1}, {-200, 0}},
  Epilog → Line[{{0.5, -16}, {1, -16}}]];
```

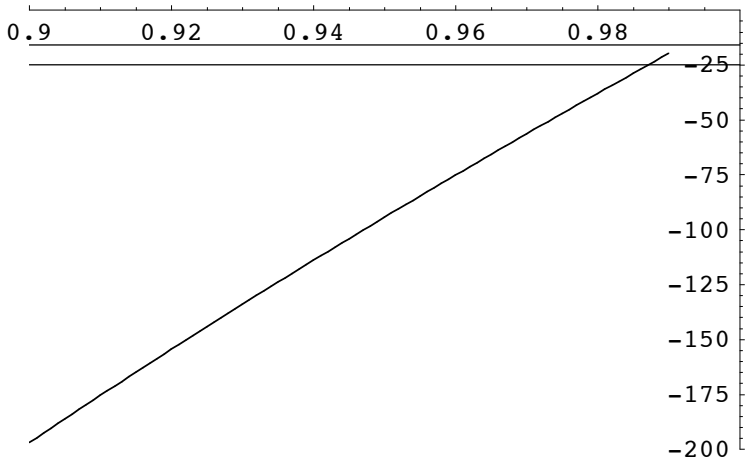


```
edata = Table[{u, SuperPolyQuantileShort[u, polydata, 500] -
     N[QuantileN[u], 200]},
   {u, 1 / 2 + 1 / 1000, 99 / 100, 1 / 1000}];
rawlogs = Map[{#[[1]], Log[10, Abs[#[[2]]]]} &, edata];
plotdata = Select[rawlogs, NumberQ[#[[2]]] &];
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.9, 1}, {-200, 0}},
  Epilog → Line[{{0.5, -16}, {1, -16}}]];
```



So we are now better than machine precision up to about 0.98. Finally by going to 1000 terms the error at $u = 0.99$ is almost down to 10^{-20}.

```
edata = Table[{u, SuperPolyQuantileShort[u, polydata, 1000] -
     N[QuantileN[u], 200]},
   {u, 1 / 2 + 1 / 1000, 99 / 100, 1 / 1000}];
rawlogs = Map[{#[[1]], Log[10, Abs[#[[2]]]]} &, edata];
plotdata = Select[rawlogs, NumberQ[#[[2]]] &];
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.9, 1}, {-200, 0}},
  Epilog → {Line[{{0.5, -16}, {1, -16}}],
    Line[{{0.5, -25}, {1, -25}}]}];
```




```
N[Last[plotdata][[2]], 3]
```

```
-19.8
```

Note that at $u = 0.98$, which is above the end point of all three popular approximation schemes, the error is less than 10^{-37}, so we can be confident about using our polynomial as a benchmarking tool for schemes that go to, and well below, machine precision. For u at about 0.9 or below the error is about 10^{-200} or less! This may all seem like overkill but given that Wichura's scheme goes to machine precision we have to do much better in order to have any kind of plausible benchmark.

We now go to the popular schemes and implement them.

■ A first implementation in C++ (C style)

The following is my very first attempt at this in C++. You can switch the doubles to long doubles if you like - on my system it makes only a marginal improvement. I also use the public domain part of the Numerical Recipes vector routines for array storage. Readers can swap in whatever their preference is. Note that this is a stand alone program with comments and I/O. The actual routine is very short, largely immune to data transcription errors, and will work with complex numbers if suitably modified. (n.b. Suggestions for improvement welcome)

```
//recursivequantile.cpp
//Normal quantile using
//Steinbrecher's non-linear recursion
//This test program works on 0.004<u<0.996
//with max relative error<3e-15
//using Bloodshed Dev C++
//(1.1e-15 with double→long double everywhere)
//William Shaw
//Version 0.2-provisional version
//for education and benchmarking only
//February 2007
//william.shaw@kcl.ac.uk

#include<cmath>
#include<iostream>
#include<fstream>
#include "nrutil_nr.h"//public domain Numerical Recipes vectors
using namespace std;

double RecursiveQuantile(double u,NRVec<double>&rat)
{int P=rat.size();
  int i;
  double b,s,t,q;
  const double rootpi=1.772453850905516027298167483341;
  const double roottwo=1.414213562373095048801688724209;
  b=rootpi*(u-0.5);
  i=-1;
  t=0;
  s=b;
  q=b*b;
  while(fabs(b)>1.0e-17&&i≤P) s=(t=s)+(b*=q*rat[i+=1]);
  if (i==P) {cout <<"recursion limit reached \n";
      cout <<"switch to tail series \n ";
      cout <<"last term = " <<b <<"\n";}
  return roottwo*s;
  }
```

```
int main()
{
  NRVec<double>qarr(2701);
  NRVec<double>ratios(2700);
  double u;
  double quantile;
  char name[5];
  int k,m;
  cout <<"Steinbrecher's recursive quantile \n ";
  cout <<"Initializing series..... \n";
  //Initialize the array of coefficients and ratios
  //using Steinbrecher recursion:
  //q[k]=Sum[q[m]*q[k-1-m]/(m+1)/(2m+1),{m,0,k-1}] qarr[0]=1.0;
  qarr[1]=1.0;
  for (k=2;k≤2700;k++)
  {qarr[k]=0.0;
      for (m=0;m<k;m++) {qarr[k]+=qarr[m]*qarr[k-1-m]/(m+1)/(2*m+1);}};
  for (k=1;k≤2700;k++)
  {qarr[k]=qarr[k]/(2*k+1);
      ratios[k-1]=qarr[k]/qarr[k-1];}
  cout <<"Series ratios computed to order 2700 \n";
  cout <<qarr[2700] <<"\n";
  cout <<ratios[2699]<<"\n";
  cout <<"Working on test array \n";
  ofstream out("quantiles.txt");
  for (k=4;k≤996;k++) {u=k/1000.;
      quantile=RecursiveQuantile(u,ratios);
      out.precision(25);
      out <<u <<","<<quantile <<"\n";}
      cout <<"Output written to quantiles.txt \n";
  cout <<"Hit any key to quit \n";
  cin>>name;
  return(0);}
```

This program (or my modifications to it) can be downloaded from

http://www.mth.kcl.ac.uk/~shaww/web_page/papers/recursivequantile.cpp

3 The Beasley-Springer-Moro method

All algorithms exploit the symmetry of the quantile and focus on one half of the unit interval. Many algorithms recognize that it is then best to split the upper region $1/2 \le u < 1$ into two or even three sub-intervals. BS worked with a split $0.5 < u \le 0.92$ and $0.92 < u < 1$. Moro's RISK article modified the tail formula and we shall work with that combination. In the inner region their rational approximation is of type (3, 4) in $(u - 0.5)^2$. From Glassermann's book, or copying Joshi's C++ code, the Beasley-Springer-Moro formula is

```
BSMQuantile[u_] :=
 Module[{v, r,
   a = {2.50662823884, -18.61500062529,
     41.39119773534, -25.44106049637},
   b = {-8.47351093090, 23.08336743743,
     -21.06224101826, 3.13082909833},
   c = {0.3374754822726147, 0.9761690190917186,
     0.1607979714918209, 0.0276438810333863,
     0.0038405729373609, 0.0003951896511919,
     0.0000321767881768,
     0.0000002888167364, 0.0000003960315187}},
  If[0.08 ≤ u ≤ 0.92,
   (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]])))),
   (If[u > 0.92, r = Log[-Log[1 - u]], r = Log[-Log[u]]];
    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]] + r * c[[9]])))))))))]
 ]
```

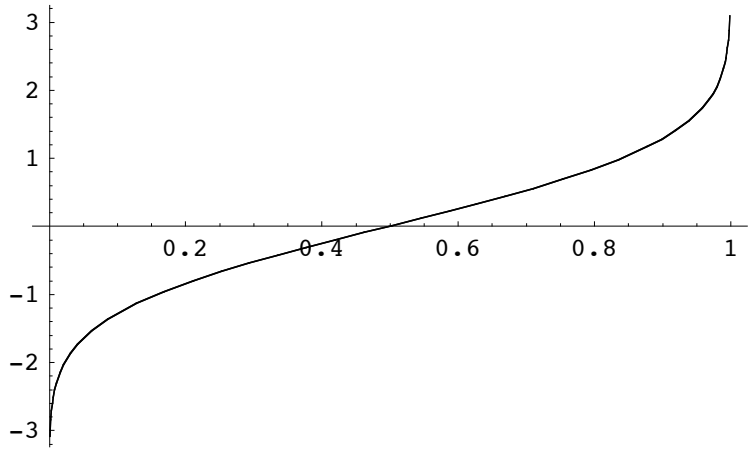
Let's make a quick evaluation check:

```
Map[BSMQuantile, {0.025, 0.5, 0.975}]
```

{−1.95996, 0., 1.95996}

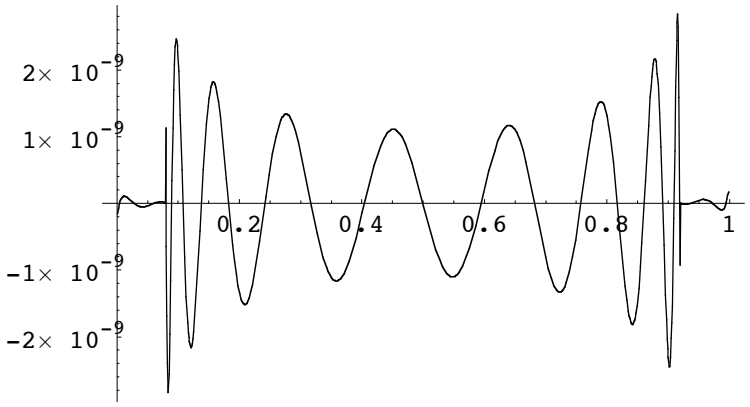
Let's plot it together with the built-in function:

```
Plot[{BSMQuantile[u], QuantileN[u] }, {u, 0.001, 0.999}];
```



You cannot see the difference over most of this plot, wo we really need to plot the difference and will do so over the interval over which this approximation is applied in the BSM framework. First here is the absolute error.

```
Plot[BSMQuantile[u] - QuantileN[u] ,
  {u, 0.001, 0.999}, PlotRange → All];
```



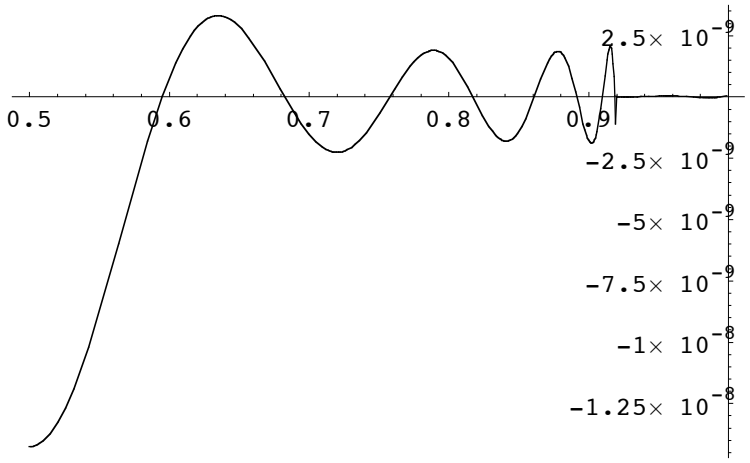
This graph is consistent with the claim that the absolute error is no more than 3×10^{-9}. Let's tabulate the error to find the maximum over the central region - we only need to worry about the upper half:

```
Max[Table[Abs[BSMQuantile[x] - QuantileN[x]],
  {x, 0.5, 0.92, 0.00001}]]
```

3.00778×10^{-9}

So the claim is almost true in this interval. Here is the relative error plot for comparison with Acklam's analysis. Acklam, rather more usefullym, works with relative errors. :

```
Plot[BSMQuantile[u] / QuantileN[u] - 1 ,
  {u, 1 / 2, 0.999}, PlotRange → All];
```



So the relative error is not so good, and peaks in the middle at:

```
BSMQuantile[0.50000000001] / QuantileN[0.50000000001] - 1
```

-1.42785×10^{-8}

The shape of both these error plots is also rather worrying and suggests that the BSM central series is sub-optimal. In a minimax approximation one would expect to see either the relative or absolute error to be managed in a more uniform way, with one of these error graphs showing oscillations between essentially level and

parallel lines. With the growing or decaying errors shown in the plots, we might expect to be able to squeeze down the maximum errors. Before attempting to do this, we shall look at Acklam's result, which is altogether better in this respect, and demonstrates the effect we are looking for.

4 Acklam's approximation

```
1 - 0.97575
```

0.02425

Acklam worked with a split $0.5 < u \leq 0.97575$ and $0.97575 < u < 1$. The following data has been pasted electronically from his web site, noting that he stores the coefficients in reverse order to that used by Glasserman. I have coded this ever so slightly differently from BSM.

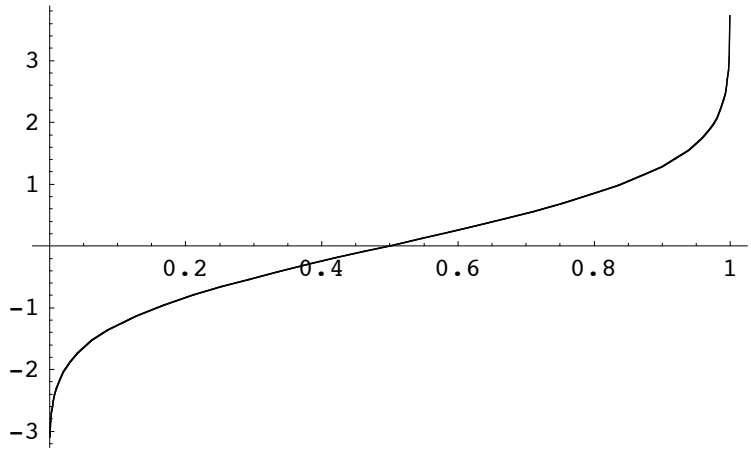
```
Acklam[u_] :=
 Module[
  {a = Reverse[{-39.69683028665376, 220.9460984245205,
      -275.9285104469687, 138.3577518672690,
      -30.66479806614716, 2.506628277459239}],
b = Reverse[{-54.47609879822406, 161.5858368580409,
      -155.6989798598866, 66.80131188771972,
      -13.28068155288572}],
   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[0.02435 ≤ 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]]))))]]]
```

Let's plot the two together:

```
Acklam[0.975]
```

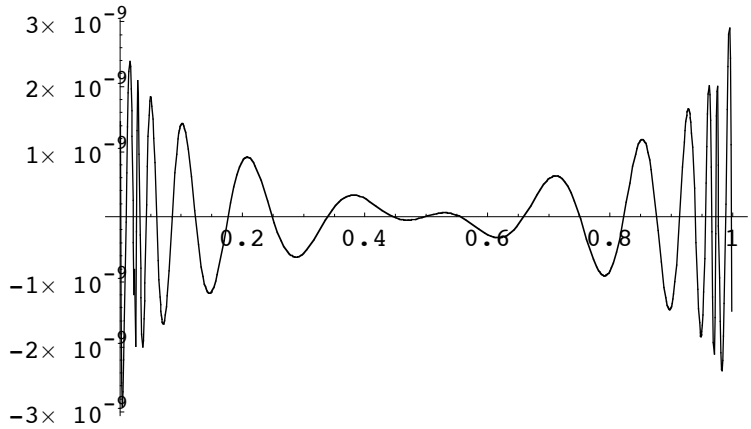
1.95996

```
Plot[{Acklam[u], QuantileN[u] }, {u, 0.001, 0.9999}];
```



Again we really need to plot the difference:

```
Plot[Acklam[u] - QuantileN[u] ,
  {u, 0.001, 0.999}, PlotRange → All];
```



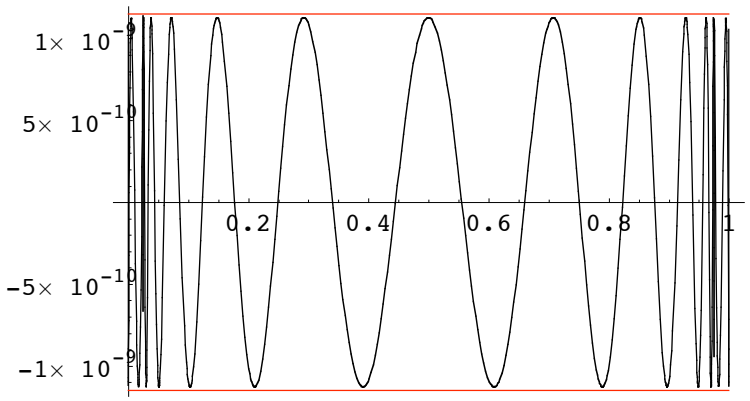
This is good, but growing, but we should note that *Acklam sought to minimize the maximum* ***relative*** *error*. Here is the maximum *absolute* error

```
Max[Table[Abs[Acklam[u] - QuantileN[u]],
  {u, 0.5, 0.99999, 0.00001}]]
```

3.92266×10^{-9}

Here is the relative error for comparison with Acklam's plot and claim that the Max Rel Error is less than 1.15 x 10E-9:

```
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}}]}];
```



This graph is consistent with the claim that the relative error is no more than 1.15×10^{-9}.

```
Max[Table[Abs[Acklam[u] / QuantileN[u] - 1],
  {u, 0.5000001, 0.99999, 0.00001}]]
```

1.12885×10^{-9}

So the error data is very good. This graph also shows the correct properties of a minimax approximation - the error is not bulging anywhere and sits between two level lines. A further probe into the tails confirm all is OK to at least $u = 10^{-22}$.

But it should also be borne in mind that the Acklam approximation uses a numerator of degree two higher, and a denominator of degree one higher, than the Beasley-Springer method - we need to ask if we can do better in some sense, either, being more accurate with only the same amount of computation as BS, or using the extra terms in Acklam's method to do better. Of course these considerations depend on the tail region as well, and Acklam's approach uses a simple rational approximation much deeper into the tail than does BS or BSM, which is a big benefit.

5 Wichura's AS241

The central portion of the Wichura (1988) AS241 algorithm is given on a region

$$0 < u - \frac{1}{2} < 0.425 \tag{3}$$

It works in a similar way to the BSM and Acklam formulae, except that this is now a larger rational approximation of type (7,7), involving substantially more calculation, and the polynomials in the numerator and denominator are expressed in terms of a reflection in the endpoint: $0.186025 = 0.425^2$. The outer region is also split again into two sub-regions.

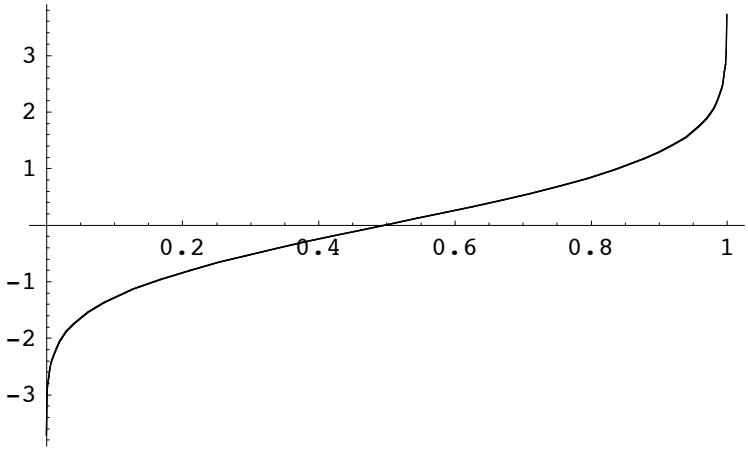
```
AS241[u_] :=
 Module[{a = {3.3871328727963666080, 133.14166789178437745,
     1971.5909503065514427, 13731.693765509461125,
     45921.953931549871457, 67265.770927008700853,
     33430.575583588128105, 2509.0809287301226727},
   b = {42.313330701600911252, 687.18700749205790830,
     5394.1960214247511077, 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,
     0.026532189526576123093O, 0.001242660947388078438 60,
     0.000027115555687434875781 5,
     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]])))))))
    )]
  ]]
```

This is now a larger rational approximation of type (7,7), involving substantially more calculation.

```
AS241[0.975]
```

1.95996

```
Plot[{AS241[u], QuantileN[u] }, {u, 0.0001, 0.9999}];
```



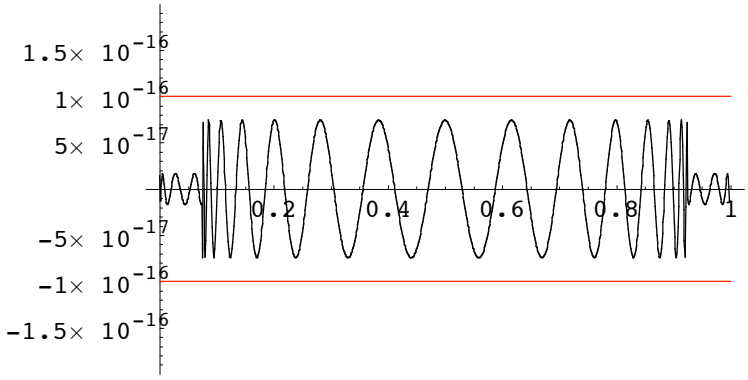
So we just test the claim that the error is of order machine precision. This requires some more specialized plotting routines to dig down below machine precision. The red lines are drawn at 10^{-16} and the relative error is less than this down to less than $u = 10^{-22}$. First here is an analysis that ignores the deep tails.

```
edata = Table[{u, AS241[u] / N[QuantileN[u], 200] - 1},
   {u, 10^(-6), 1 - 10^(-6), 1 / 1000}];

Max[Abs[Transpose[edata][[2]]]]
```

7.43×10^{-17}

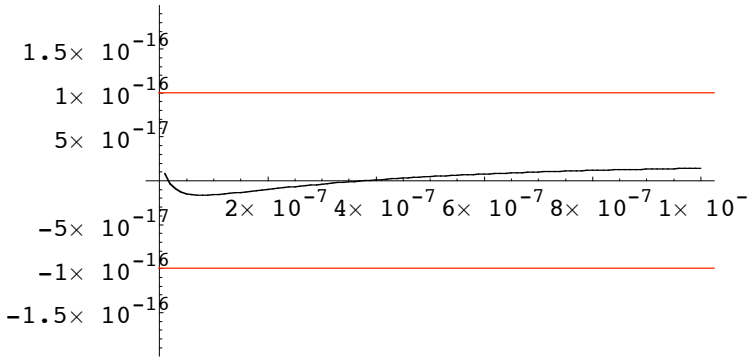
So the relative error is less than 10^{-16}. Here is a plot that shows the errors in the central and inner tail regions:

```
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}];
```



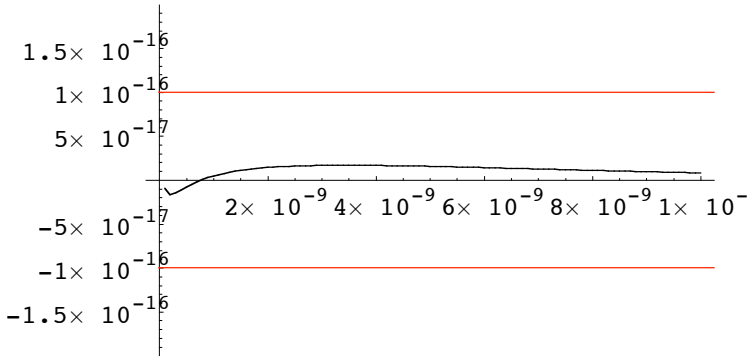
Here we check the left tail:

```
edata = Table[{u, AS241[u] / N[QuantileN[u], 200] - 1},
   {u, 10^(-8), 10^(-6), 10^(-8)}];

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}];
```



Now we go deeper into the left tail:

```
edata = Table[{u, AS241[u] / N[QuantileN[u], 200] - 1},
   {u, 10^(-10), 10^(-8), 10^(-10)}];

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}];
```



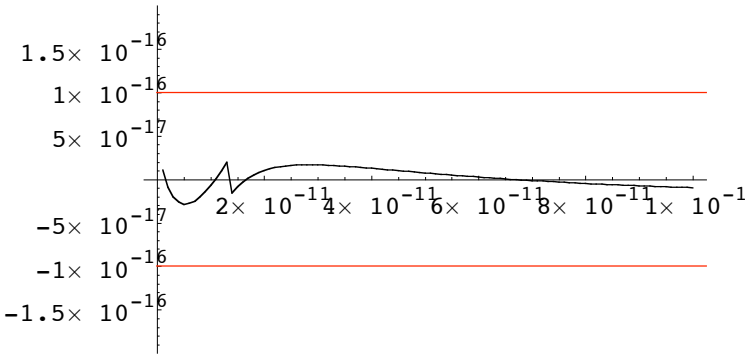
Deeper still, to investigate the switch at $\sqrt{-\log[u]} = 5$, i.e. $u = \text{Exp}[-25]$:

```
Exp[-25] // N
```

1.38879×10^{-11}

```
edata = Table[{u, AS241[u] / N[QuantileN[u], 200] - 1},
   {u, 10^(-12), 10^(-10), 10^(-12)}];
```

```
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}];
```



I have repeated this down to 10^{-22} with no problems emerging. This is outstanding.

Note that Wichura's method gets to machine precision with only a (7, 7) rational approximation, so is roughly 40% more effort than the Acklam scheme which is (5, 5), albeit on a shorter interval. Acklam's scheme goes further into the tail and so the more complex tail series needs to be used less frequently. Of course Acklam also allows for a Halley iteration to get to machine precision.

6 Rational Approximations in *Mathematica* and checking BSM

Mathematica has a package for constructing rational approximations and may be set up to find approximations that minimize either the maximum relative error or the maximum absolute error. We load it with the following package load statement.

```
Needs["NumericalMath`Approximations` "]
```

Now we want to work on optimizing a rational approximation exploiting the fact that the inverse is an odd function, so we work on the key part of the inverse function. Let's look at it using the built in function and the first part of our big polynomial:

```
KeyBitA[x_, data_] :=
 Module[{len = Length[data], w = Pi * x^2},
  Sqrt[2 Pi] * Sum[data[[k]] * w^(k - 1) / (2 k - 1), {k, 1, len}]]

KeyBit[x_] := Sqrt[2] InverseErf[2 x] / x

Series[KeyBit[y], {y, 0, 14}]
```

$$\sqrt{2\pi} + \frac{1}{3}\sqrt{2}\,\pi^{3/2}\,y^2 + \frac{7\,\pi^{5/2}\,y^4}{15\sqrt{2}} + \frac{127\,\pi^{7/2}\,y^6}{315\sqrt{2}} + \frac{4369\,\pi^{9/2}\,y^8}{11340\sqrt{2}} + \frac{34807\,\pi^{11/2}\,y^{10}}{89100\sqrt{2}} + \frac{20036983\,\pi^{13/2}\,y^{12}}{48648600\sqrt{2}} + \frac{2280356863\,\pi^{15/2}\,y^{14}}{5108103000\sqrt{2}} + O(y^{15})$$

```
N[%]
```

$$2.50663 + 2.62493\,(y+0.)^2 + 5.77253\,(y+0.)^4 + 15.6676\,(y+0.)^6 + 47.0358\,(y+0.)^8 + 149.83\,(y+0.)^{10} + 496.274\,(y+0.)^{12} + 1689.87\,(y+0.)^{14} + O((y+0.)^{15})$$

The truncated series is a polynomial in y^2 so we will work on this basis and put a square root into the definition. Also, Mathematica's rational approximation stuff is a bit fussy about what you give it (it objects to using **KeyBit** directly, which may well have been found by others as an obstacle to using this tool), so the first thing we do is truncate the power series at a sufficiently high order such that the resulting polynomial is very accurate on the Beasley Springer region $0.5 < u < 0.92$ or the corresponding Acklam region. So we use our big polynomial:

```
rootthing = KeyBitA[Sqrt[y], polydata];
```

```
PolyQuantileOne[x_] = (x - 1 / 2) rootthing /. y → (x - 1 / 2)^2;
```

We already know that this is extremely accurate over the given region.

Now do a rational approximation on the remaining BSM interval. The settings of the following follow from a little experimentation. The important thing is to note that the 3,4 means a numerator of degree three and a denominator of degree 4.

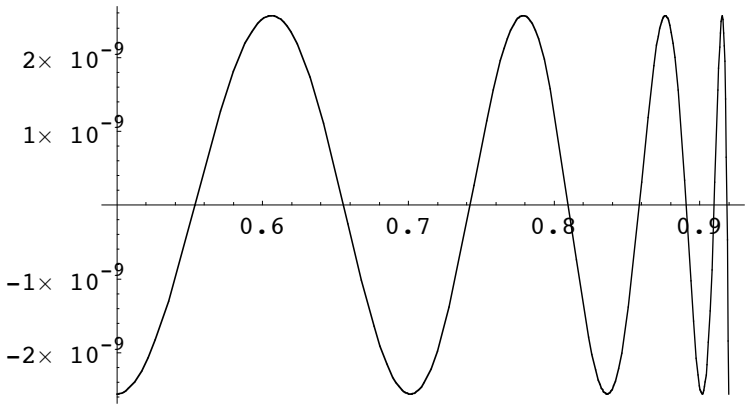
```
revrelBSM =
 MiniMaxApproximation[rootthing, {y, {0, 0.42^2}, 3, 4},
   Brake → {10, 10}, MaxIterations → 40,
   WorkingPrecision → 20, PlotFlag → False][[2, 1]]
```

$$(-24.820209533706798850\,y^3 + 40.864622120467790785\,y^2 - 18.515898959450185753\,y + 2.5066282682076065359) / (3.0154847661978822127\,y^4 - 20.641301545177201274\,y^3 + 22.831834928541562628\,y^2 - 8.4339736056039657294\,y + 1)$$

```
RatQuantileOne[x_] = (x - 1 / 2) % /. y → (x - 1 / 2)^2;
```

Here is a plot of the relative error over the region:

```
Plot[RatQuantileOne[x] / QuantileN[x] - 1,
  {x, 0.5, 0.92}, PlotRange → All];
```



```
Max[Table[Abs[RatQuantileOne[u] / QuantileN[u] - 1],
  {u, 0.5000001, 0.92, 0.00001}]]
```

2.56256×10^{-9}

So this is better by a factor of more than 5 than the maximum relative error with the BSM data. And the error plot has the characteristics of a minimax approximation that we expect. *So to use an rational approximation of BSM form that minimizes the maximum* ***relative*** *error the list of "a" coefficients should be adjusted slightly, to:*

```
CoefficientList[Numerator[revrelBSM], y]
```

{2.5066282682076065359, −18.515898959450185753,
 40.864622120467790785, −24.820209533706798850}

While the list of "b" coefficients should be adjusted to.

```
Rest[CoefficientList[Denominator[revrelBSM], y]]
```

{−8.4339736056039657294, 22.831834928541562628,
 −20.641301545177201274, 3.0154847661978822127}

This decreases the maximum relative error from about 1.4 E-8 to 2.6E-9, over the central BSM region.

■ **Absolute error minimization**

This is a slightly trickier issue due to the need to use a generalization of the function used above, and the introduction of an error weighting function:

```
?GeneralMiniMaxApproximation
```

```
GeneralMiniMaxApproximation[{x, y, (g)}, {t, {t0, t1}, m, k}, xvar,
   (opts)] finds the mini-max approximation to the function y[x]
   on the interval t0 < t < t1 where x and y are functions of t,
   which parametrically define y[x], m and k are the degrees of the
   numerator and denominator, respectively, and xvar is the variable
   to be used for x in the approximation.  The optional g is also a
   function of t, and the error to be minimized is (y[x[t]] - approx[
   x[t]])/g[t]. The answer returned is {AbscissaList, {Approximation,
   MaxError}}, where AbscissaList is a list of the abscissas (values
   of t) where the maximum error occurs. Approximation is the rational
   approximation desired, and MaxError is the value of the mini-max error.
 GeneralMiniMaxApproximation[{x, y, (g)}, approx, {t, {t0, t1}, m,
   k}, xvar, (opts)] is a form that allows the user to start the
   iteration from a known approximation.  Here approx must be in the
   form of an answer returned by GeneralMiniMaxApproximation. More…
```

Some thought reveals that the appropriate weight function is just $1/\sqrt{t}$, but this cannot be used as is due to the singular behaviour at the origin. So we perturb it to $1/\left(\epsilon + \sqrt{t}\right)$ and then optimize over ϵ. Some rather extensive search calculations lead to $\epsilon = 0.035771$ and the following rational approximation.

```
compthing = rootthing /. y → t;
```

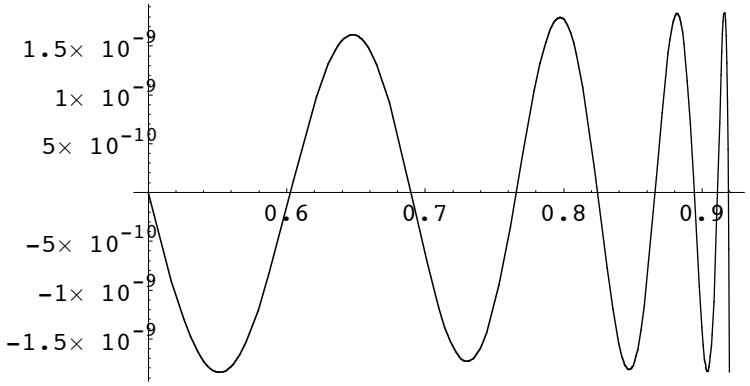
```
revabsBSM = GeneralMiniMaxApproximation[
   {t, compthing, 1 / (0.035771 + Sqrt[t])}, {t, {0, 0.42^2},
    3, 4}, t, Brake → {10, 10}, MaxIterations → 40,
   WorkingPrecision → 20, PlotFlag → False][[2, 1]]
```

$$(-25.965959614983341521\, t^3 + 41.841057826285191701\, t^2 - 18.700226003845536893\, t + 2.5066282186077179315) / (3.2269921657498348465\, t^4 - 21.419275595427556433\, t^3 + 23.298480931332967694\, t^2 - 8.5075119518749713281\, t + 1)$$

```
RatQuantileOne[x_] = (x - 1 / 2) revabsBSM /. t → (x - 1 / 2)^2;

Max[Abs[Table[
   RatQuantileOne[x] - QuantileN[x], {x, 0.5, 0.92, 0.01}]]]
```

1.84673×10^{-9}

```
Plot[RatQuantileOne[x] - QuantileN[x],
 {x, 0.5, 0.92}, PlotRange → All]
```



- Graphics -

So this is better than the maximum absolute error with the BSM data. And the error plot has more of the characteristics of a minimax approximation that we expect. However, in this case given the method employed we can only claim that this is better than BSM, and not necessarily optimal. *So to use a rational approximation of BSM form that lowers the maximum* ***absolute*** *error the list of "a" coefficients should be adjusted slightly, to:*

```
CoefficientList[Numerator[revabsBSM], t]
```

{2.5066282186077179315, −18.700226003845536893, 41.841057826285191701, −25.965959614983341521}

While the list of "b" coefficients should be adjusted to.

```
Rest[CoefficientList[Denominator[revabsBSM], t]]
```

{−8.5075119518749713281, 23.298480931332967694, −21.419275595427556433, 3.2269921657498348465}

However, from a precision point of view the set that minimizes relative error is preferable, and indeed better than BSM by a factor of five in the relative error.

■ Variations using the BSM region

▪ One more term in numerator and denominator

Let's do a little more work and allow one more square in both the numerator and denominator

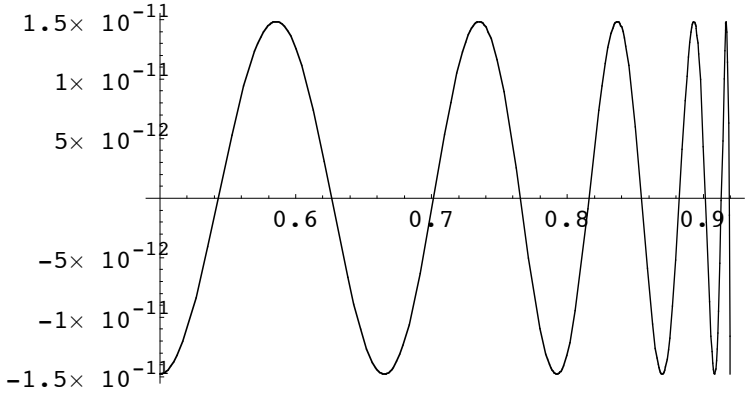
```
revrelBSMA =
 MiniMaxApproximation[rootthing, {y, {0, 0.42^2}, 4, 5},
   Brake → {10, 10}, MaxIterations → 40,
   WorkingPrecision → 20, PlotFlag → False][[2, 1]]
```

$$(41.336656803386989331\, y^4 - 105.84210116376198023\, y^3 + 81.124697593450729775\, y^2 - 24.301357591165500310\, y + 2.5066282745938837616)/(-3.9151522373688291510\, y^5 + 39.896238821445256473\, y^4 - 66.997432181293659162\, y^3 + 41.310199682875185297\, y^2 - 10.742036575321449665\, y + 1)$$

```
RatQuantileVarOne[x_] = (x - 1 / 2) revrelBSMA /. y → (x - 1 / 2)^2;
```

Here is a plot of the relative error over the region:

```
Plot[RatQuantileVarOne[x] / QuantileN[x] - 1,
  {x, 0.5, 0.92}, PlotRange → All];
```




```
Max[Table[Abs[RatQuantileVarOne[u] / QuantileN[u] - 1],
  {u, 0.5000001, 0.92, 0.00001}]]
```

$$1.48106 \times 10^{-11}$$

So we get down to better than 1.5E-11. This is less work than in Acklam's fomula but over a shorter interval. The a list is now

```
CoefficientList[Numerator[revrelBSMA], y]
```

{2.5066282745938837616, −24.301357591165500310,
81.124697593450729775, −105.84210116376198023, 41.336656803386989331}

While the list of "b" coefficients should be adjusted to.

```
Rest[CoefficientList[Denominator[revrelBSMA], y]]
```

{−10.742036575321449665, 41.310199682875185297,
 −66.997432181293659162, 39.896238821445256473, −3.9151522373688291510}

- **Two more terms in numerator and denominator**

Let's do a little more work and allow one more square in both the numerator and denominator

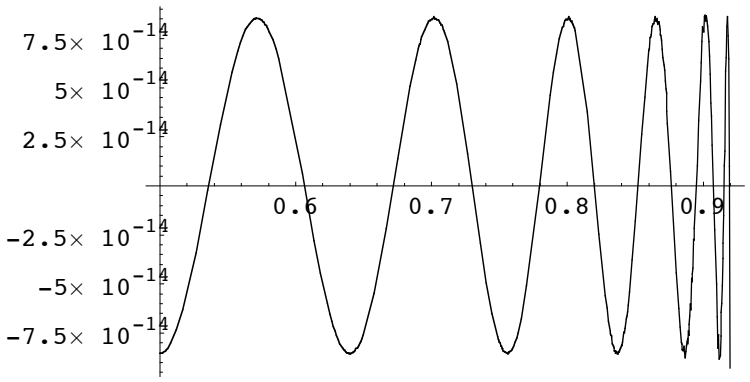
```
revrelBSMB =
 MiniMaxApproximation[rootthing, {y, {0, 0.42^2}, 5, 6},
   Brake → {10, 10}, MaxIterations → 40,
   WorkingPrecision → 20, PlotFlag → False][[2, 1]]
```

$$(-65.871752394631995007\, y^5 + 240.90242193180930121\, y^4 - 273.80041456455819039\, y^3 + 134.66512972532018904\, y^2 - 30.079483805333604735\, y + 2.5066282746307863413)/(5.0928736250315728292\, y^6 - 72.194957674033174332\, y^5 + 169.85006173352583420\, y^4 - 153.59006018987562373\, y^3 + 65.083676406361719335\, y^2 - 13.047175413875805207\, y + 1)$$

```
RatQuantileVarTwo[x_] = (x - 1 / 2) revrelBSMB /. y → (x - 1 / 2)^2;
```

Here is a plot of the relative error over the region:

```
Plot[RatQuantileVarTwo[x] / QuantileN[x] - 1,
  {x, 0.5, 0.92}, PlotRange → All];
```




```
Max[Table[Abs[RatQuantileVarTwo[u] / QuantileN[u] - 1],
  {u, 0.5000001, 0.92, 0.00001}]]
```

9.90319×10^{-14}

So we get down to better than 1.0E-13. This is a bit more work than in Acklam's fomula and over a shorter interval. The coefficients are

```
CoefficientList[Numerator[revrelBSMB], y]
```

```
{2.5066282746307863413, -30.079483805333604735, 134.66512972532018904,
  -273.80041456455819039, 240.90242193180930121, -65.871752394631995007}
```

While the list of "b" coefficients should be adjusted to.

```
Rest[CoefficientList[Denominator[revrelBSMB], y]]
```

```
{-13.047175413875805207, 65.083676406361719335, -153.59006018987562373,
  169.85006173352583420, -72.194957674033174332, 5.0928736250315728292}
```

7 A corresponding investigation of Acklam's method

Acklam's graphs on his web site and the one we have given above indicate a rather better approach and that the formula given is almost certainly an optimal minimax for the relative error. Let's see what *Mathematica* comes up with! We will need to work quite hard as Acklam's formula is *very* good!

```
PolyQuantileTwo[x_] = (x - 1 / 2) rootthing /. y → (x - 1 / 2)^2;
```

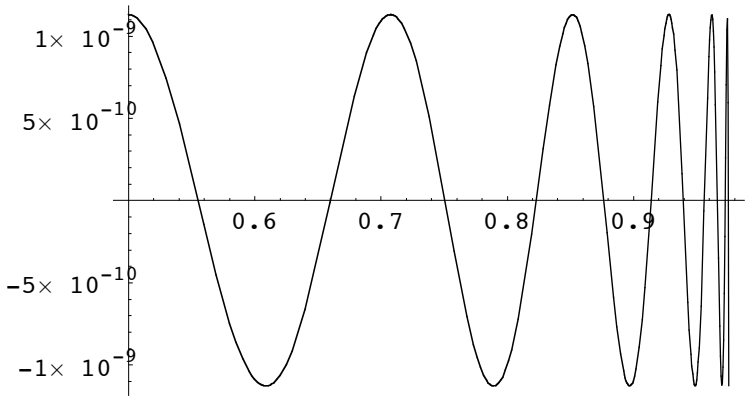
```
revrelAck =
 MiniMaxApproximation[rootthing, {y, {0, 0.47575^2}, 5, 5},
   Brake → {70, 70}, MaxIterations → 50,
   WorkingPrecision → 25, PlotFlag → False][[2]]
```

$$\{(-39.69686278848757498642429\, y^5 + 220.946180132765781294259 5\, y^4 - 275.928565433135151031267 1\, y^3 + 138.3577661367497249833873\, y^2 - 30.664799346260741067642 35\, y + 2.5066282774603192767592 23)/ (-54.476129570751472897325 31\, y^5 + 161.585886643290622826597 8\, y^4 - 155.699007115034677196217 4\, y^3 + 66.801318113589230182098 95\, y^2 - 13.280682063530322334222 45\, y + 1), -1.12873488381916541063565 1 \times 10^{-9}\}$$

```
RatQuantileThree[x_] =
  (x - 1 / 2) revrelAck[[1]] /. y → (x - 1 / 2)^2;
```

Here is a plot of the relative error over the region:

```
Plot[RatQuantileThree[x] / QuantileN[x] - 1,
  {x, 0.5, 0.97575}, PlotRange → All];
```




```
Max[Table[Abs[RatQuantileThree[u] / QuantileN[u] - 1],
  {u, 0.5000001, 0.97575, 0.00001}]]
```

1.12882×10^{-9}

So we are ever so slightly better than what we get from Acklam's formula, which is:

```
Max[Table[Abs[Acklam[u] / QuantileN[u] - 1],
  {u, 0.5000001, 0.97575, 0.00001}]]
```

1.12885×10^{-9}

i.e. the max error from our method is very marginally better than Acklam. So we will not be necessarily suggesting replacing his coefficients with ours, which for the numerator are, in the order given by Acklam

```
Reverse[CoefficientList[Numerator[revrelAck[[1]]], y]]
```

{−39.6968627884875749864 2429, 220.9461801327657812942595,
 −275.9285654331351510312671, 138.3577661367497249833873,
 −30.6647993462607410676 4235, 2.5066282774603192767592 23}

and the denominator list is

```
Reverse[Rest[CoefficientList[Denominator[revrelAck[[1]]], y]]]
```

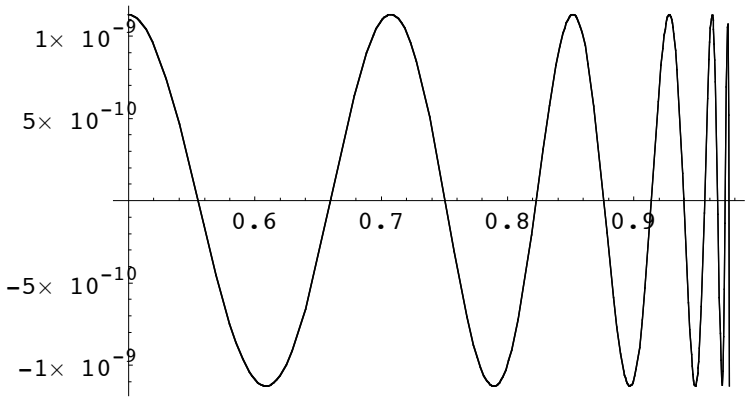
{−54.4761295707514728973 2531,
 161.5858866432906228265978, −155.6990071150346771962174,
 66.8013181135892301820 9895, −13.2806820635303223342 2245}

We remind the reader that the corresponding data from Acklam's web site is:

```
{-39.69683028665376, 220.9460984245205, -275.9285104469687,
  138.3577518672690, -30.66479806614716, 2.506628277459239};
{-54.47609879822406, 161.5858368580409, -155.6989798598866,
  66.80131188771972, -13.28068155288572};
```

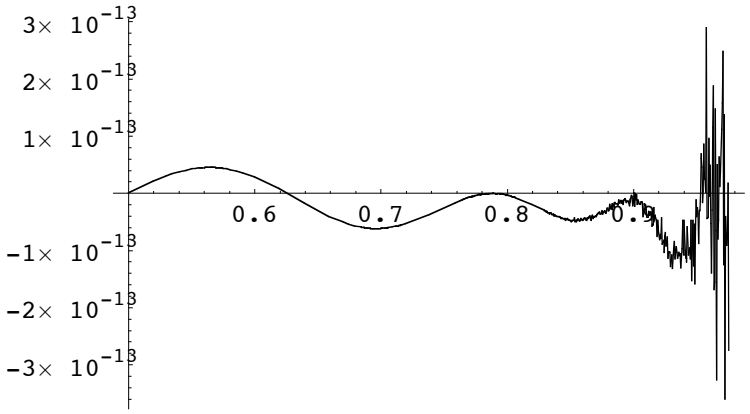
Clearly the coefficients are also almost identical, and here is a plot of the errors of the two functions overlaid:

```
Plot[{RatQuantileThree[x] / QuantileN[x] - 1,
   Acklam[x] / QuantileN[x] - 1},
  {x, 0.5, 0.97575}, PlotRange → All];
```



Here is the difference

```
Plot[RatQuantileThree[x] - Acklam[x],
  {x, 0.5, 0.97575}, PlotRange → All];
```



■ **A modification to Acklam with one more power top and bottom**

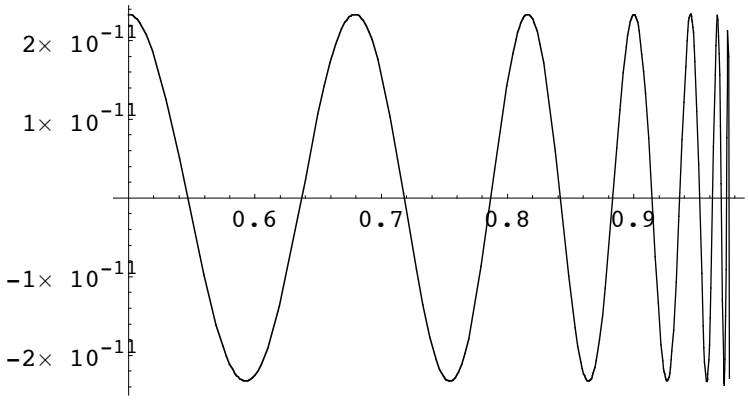
```
revrelAckA =
 MiniMaxApproximation[rootthing, {y, {0, 0.47575^2}, 6, 6},
   Brake → {70, 70}, MaxIterations → 50,
   WorkingPrecision → 25, PlotFlag → False][[2, 1]]
```

$$\begin{aligned}&(63.832546933221365661888807\, y^6 - \\ &\quad 489.5133581983256689241613\, y^5 + 838.5319954259973887488977\, y^4 - \\ &\quad 604.2274869543201246471315\, y^3 + 214.1861555101569440118991\, y^2 - \\ &\quad 37.060678256898776150327 46\, y + 2.5066282746893892734316 13)\,/ \\ &(99.9537811783919421719655 9\, y^6 - 394.7736036000437859716064\, y^5 + \\ &\quad 515.2163224848749999753231\, y^4 - 315.2733228580261880164159\, y^3 + \\ &\quad 99.7245176128451954291374 0\, y^2 - 15.8322690348872252362 7608\, y + 1)\end{aligned}$$

```
RatQuantileVarThree[x_] = (x - 1 / 2) revrelAckA /. y → (x - 1 / 2) ^ 2;
```

Here is a plot of the relative error over the region:

```
Plot[RatQuantileVarThree[x] / QuantileN[x] - 1,
  {x, 0.5, 0.97575}, PlotRange → All];
```




```
Max[Table[Abs[RatQuantileVarThree[u] / QuantileN[u] - 1],
  {u, 0.5000001, 0.97575, 0.00001}]]
```

2.40931×10^{-11}

So if want wants to work with a method of Acklam type with this level of error in the central region it is a simple matter of extending the two sets of coefficients to be, first for the numerator,

```
Reverse[CoefficientList[Numerator[revrelAckA], y]]
```

{63.83254693322136566188807,
 −489.5133581983256689241613, 838.5319954259973887488977,
 −604.2274869543201246471315, 214.1861555101569440118991,
 −37.06067825689877615032746, 2.506628274689389273431613}

and the denominator list is

```
Reverse[Rest[CoefficientList[Denominator[revrelAckA], y]]]
```

{99.95378117839194217196559, −394.7736036000437859716064,
 515.2163224848749999753231, −315.2733228580261880164159,
 99.7245176128451954291374 0, −15.832269034887225236276 08}

■ **A modification to Acklam with one more power in the bottom**

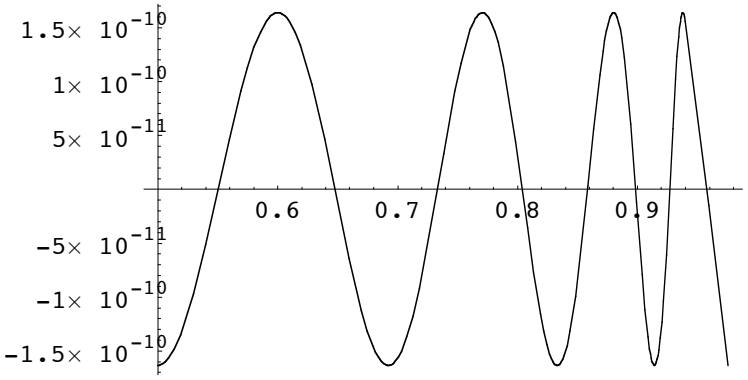
```
revrelAckB =
 MiniMaxApproximation[rootthing, {y, {0, 0.47575^2}, 5, 6},
   Brake → {70, 70}, MaxIterations → 50,
   WorkingPrecision → 25, PlotFlag → False][[2, 1]]
```

$$(-164.75765536603472003145720\, y^5 + 451.71396795186158445376190\, y^4 - 416.65677423141578891902240\, y^3 + 173.68215409585370460161870\, y^2 - 33.81938143513469679728244\, y + 2.50662827422104742909166 3)/(16.28319283447602122654025\, y^6 - 161.04625512759194454116120\, y^5 + 298.70413371853203947907730\, y^4 - 225.08217304833483031890790\, y^3 + 82.21164639322003641753099\, y^2 - 14.53917878291234960873113\, y + 1)$$

```
RatQuantileVarFour[x_] = (x - 1 / 2) revrelAckB /. y → (x - 1 / 2)^2;
```

Here is a plot of the relative error over the region:

```
Plot[RatQuantileVarFour[x] / QuantileN[x] - 1,
  {x, 0.5, 0.97575}, PlotRange → All];
```




```
Max[Table[Abs[RatQuantileVarFour[u] / QuantileN[u] - 1],
  {u, 0.5000001, 0.97575, 0.00001}]]
```

$$1.64121 \times 10^{-10}$$

So if want wants to work with a method of Acklam type with this level of error in the central region it is a simple matter of extending the two sets of coefficients to be, first for the numerator,

```
Reverse[CoefficientList[Numerator[revrelAckB], y]]
```

$$\{-164.75765536603472003145720,\ 451.71396795186158445376190,\ -416.65677423141578891902240,\ 173.68215409585370460161870,\ -33.81938143513469679728244,\ 2.50662827422104742909166 3\}$$

and the denominator list is

```
Reverse[Rest[CoefficientList[Denominator[revrelAckB], y]]]
```

```
{16.283192834476021226540 25, -161.046255127591944541 1612,
  298.704133718532039479 0773, -225.082173048334830318 9079,
  82.211646393220036417 53099, -14.539178782912349608 73113}
```

8 A corresponding investigation of Wichura's AS241

It will be evident by now that all these different schemes correspond to making different choices for the degree and numerator of the rational approximation, and a good choice of split in the interval. If we go to (7,7) and split at $u = 0.925$ we can try to generate something like Wichura's AS241 central formula. The algorithm AS241 reverses the coordinates so we shall build this in, by considering:

```
(425 / 1000)^2 // N
```

```
0.180625
```

```
reversed = rootthing /. y → (425 / 1000)^2 - z;
```

Now we form an approximation to this object, using a lot of internal precision during the calculation.

```
sevenseven = MiniMaxApproximation[reversed,
  {z, {0, 0.425^2}, 7, 7}, Brake → {80, 80},
  MaxIterations → 200, WorkingPrecision → 50, PlotFlag → False]
```

$$\{\{0,\ 0.00100758583811113930573375457255345681352404044011 26,\ 0.00406402957534417356756218716367794538292934031892 51,\ 0.00926550088738042142088059676310397519018598609723 54,\ 0.01675493557110292701475984024717803826635223838933 9,\ 0.02669253498693509970410905295628922197142446134259 4,\ 0.03920836366450326295914926789049772426993150593330 8,\ 0.05433346976363083411904910831787519472560830822521 6,\ 0.07190968877070012362351196508869226739235407302253 6,\ 0.09148684056809706355445679319098952026994494418578 4,\ 0.11222984216309460821755551961765023856139515740147,\ 0.13287422010834830264555261159524408100317757982694,\ 0.15177763514462983388233272818551374535836767378744,\ 0.16710360760259937067006242674050505744700311885425,\ 0.17713176486572202576747318588839360920652045854678,\ 0.18062499999999998001598555674718227237462997436523\},$$

$$\{(2509.08140698705347815298315604072787310680282742 70\ z^7 + 33430.58042350699302236654533763641580997322380 8597\ z^6 + 67265.77810276401858844915743186573054174816871 0612\ z^5 + 45921.95740003281554481998065770249884463729352 7101\ z^4 + 13731.69445662431521786423640889620077279305422 8337\ z^3 + 1971.59100942750274246976835910383393469369536278 53\ z^2 + 133.14166968794228852770808564748670287633346154 120\ z + 3.38713287279636660802595287794277881587392103797 64)/(5226.49617249903408488555712966436578438052183766 88\ z^7 + 28729.08949161561153506792644231667653311569335 1172\ z^6 + 39307.89960944879279687983377497216778856530418 4291\ z^5 + 21213.79576597311165197005286930550778957359515 4341\ z^4 + 5394.19627101372483054989976111930695086481593821 45\ z^3 + 687.18702654028253804115761174023624065078998751 050\ z^2 + 42.31333123188950001718728759213198183010960669 9891\ z + 1),\ 7.42819488602871736895317502612455142582822501194 27 \times 10^{-17}\}\}$$

```
RatQuantileVarFive[x_] =
  (x - 1 / 2) sevenseven[[2, 1]] /. z → (425 / 1000)^2 - (x - 1 / 2)^2;

RatQuantileVarFive[0.975]
```

1.95996

Here is a plot of the relative error over the region:

```
edata =
  Table[{u, RatQuantileVarFive[u] / N[QuantileN[u], 200] - 1},
   {u, 501 / 1000, 925 / 1000, 1 / 1000}];

Max[Abs[Transpose[edata][[2]]]]
```

$$7.42819488602871736895317502612455 \times 10^{-17}$$

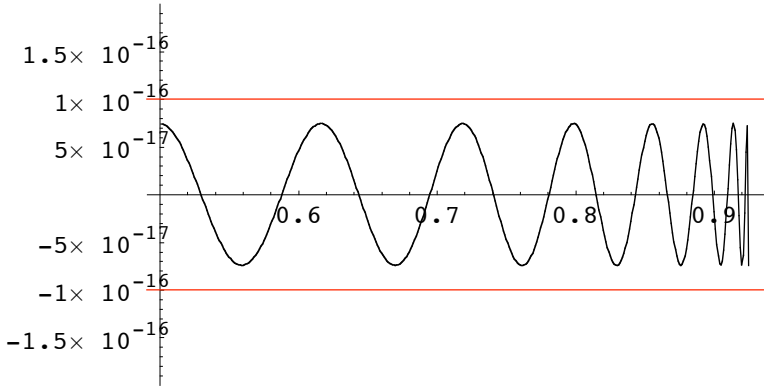
We cannot really claim any difference from the original error of the published scheme, which over this interval is

```
oldedata = Table[{u, AS241[u] / N[QuantileN[u], 200] - 1},
   {u, 501 / 1000, 925 / 1000, 1 / 1000}];

Max[Abs[Transpose[oldedata][[2]]]]
```

7.43×10^{-17}

So the relative error is less than 10^{-16} and almost identical to the published scheme. Here is a plot that shows the errors in the central and inner tail regions:

```
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}];
```



The coefficients in the numerator in the original AS241 were

```
{3.3871328727963666080, 133.14166789178437745,
 1971.5909503065514427, 13731.693765509461125,
 45921.953931549871457, 67265.770927008700853,
 33430.575583588128105, 2509.0809287301226727}
```

whereas our formula has

```
CoefficientList[Numerator[sevenseven[[2, 1]]], z]
```

{3.3871328727963666080259528779427788158739210379764,
133.14166968794228852770808564748670287633346154120,
1971.5910094275027424697683591038339346936953627853,
13731.694456624315217864236408896200772793054228337,
45921.957400032815544819980657702498844637293527101,
67265.778102764018588449157431865730541748168710612,
33430.580423506993022366545337636415809973223808597,
2509.081406987053478152983156040727873106802827427 0}

The corresponding coefficients in the denominator in AS241 were

```
{42.313330701600911252, 687.18700749205790830,
 5394.1960214247511077, 21213.794301586595867,
 39307.895800092710610, 28729.085735721942674,
 5226.4952788528545610},
```

whereas we have generated the following:

```
Rest[CoefficientList[Denominator[sevenseven[[2, 1]]], z]]
```

```
{42.313331231889500017187287592131981830109606699891,
 687.18702654028253804115761174023624065078998751050,
 5394.19627101372483054989976111930695086481593821 45,
 21213.795765973111651970052869305507789573595154341,
 39307.899609448792796879833774972167788565304184291,
 28729.089491615611535067926442316676533115693351172,
 5226.49617249903408488555712966436578438052183766 88}
```

There is clearly not a lot of difference, and the higher significant figures are not that "significant".

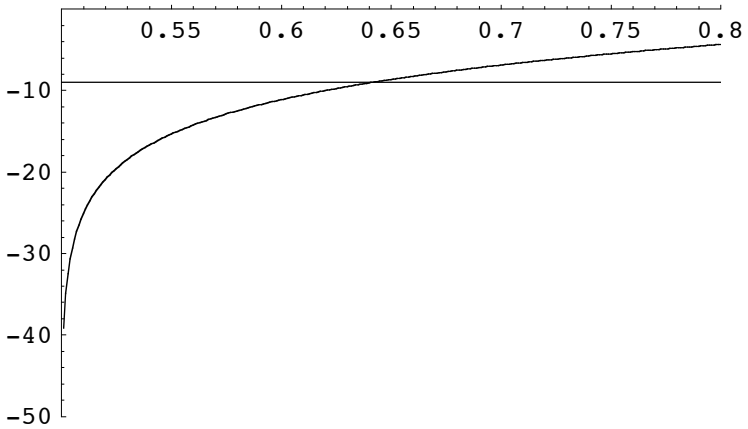
9 Hybrid Polynomial-Rational Approximations - a first look

n.b. This section is new and not at all optimized.

One issue with all of the methods descibed above is that they throw away a known polynomial that is very accurate in a neighbourhood of $u = 1/2$. Of course, we would not, with current technology, propose using our benchmark polynomial of degree 1000. However, given that the popular schemes have rational degrees (3,4), (5,5) and (7,7), we can consider polynomials of degree 7, 10 and 14. The target accuracy for the three cases is about 10^{-9}, 10^{-9}, 10^{-16}. Using our high precision plotting tools we can see that polynomials of degree 7, 10, and 14 attain this level of relative accuracy in a region extending to about $u = 0.64$to 0.7:

■ **Polynomial of BSM Complexity**

```
edata = Table[{u, SuperPolyQuantileShort[u, polydata, 7] /
      N[QuantileN[u], 200] - 1},
   {u, 1 / 2 + 1 / 1000, 80 / 100, 1 / 1000}];
rawlogs = Map[{#[[1]], Log[10, Abs[#[[2]]]]} &, edata];
plotdata = Select[rawlogs, NumberQ[#[[2]]] &];
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.5, 0.8}, {-50, 0}},
  Epilog → Line[{{0.5, -9}, {0.8, -9}}]];
```



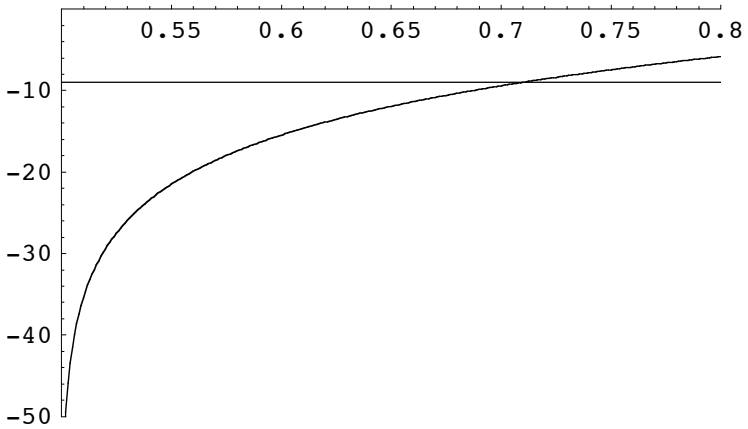
For example:

```
SuperPolyQuantileShort[u, polydata, 7] /
   N[QuantileN[u], 200] - 1 /. u → 0.64
```

-7.87073×10^{-10}

■ **Polynomial of Acklam Complexity**

```
edata = Table[{u, SuperPolyQuantileShort[u, polydata, 10] /
      N[QuantileN[u], 200] - 1},
   {u, 1 / 2 + 1 / 1000, 80 / 100, 1 / 1000}];
rawlogs = Map[{#[[1]], Log[10, Abs[#[[2]]]]} &, edata];
plotdata = Select[rawlogs, NumberQ[#[[2]]] &];
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.5, 0.8}, {-50, 0}},
  Epilog → Line[{{0.5, -9}, {0.8, -9}}]];
```

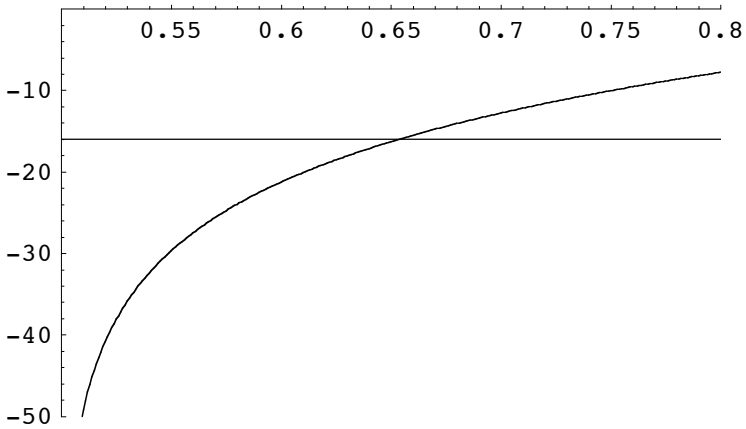



```
SuperPolyQuantileShort[u, polydata, 10] /
   N[QuantileN[u], 200] - 1 /. u → 0.7
```

-3.47025×10^{-10}

■ **Polynomial of Wichura Complexity**

```
edata = Table[{u, SuperPolyQuantileShort[u, polydata, 14] /
      N[QuantileN[u], 200] - 1},
   {u, 1 / 2 + 1 / 1000, 80 / 100, 1 / 1000}];
rawlogs = Map[{#[[1]], Log[10, Abs[#[[2]]]]} &, edata];
plotdata = Select[rawlogs, NumberQ[#[[2]]] &];
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.5, 0.8}, {-50, 0}},
  Epilog → Line[{{0.5, -16}, {0.8, -16}}]];
```



For example

```
SuperPolyQuantileShort[u, polydata, 14] /
   N[QuantileN[u], 200] - 1 /. u → 0.66
```

$$-4.44089 \times 10^{-16}$$

For non-*Mathematica* users wishing to employ such hybrids, here are the coefficients of the polynomials out to degree 20, both exactly and to high numerical precision

```
Table[q[k], {k, 1, 20}]
```

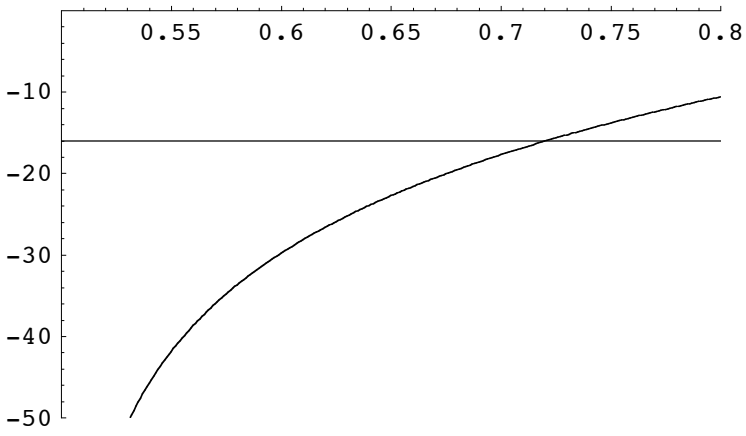
$$\left\{1, \frac{7}{6}, \frac{127}{90}, \frac{4369}{2520}, \frac{34807}{16200}, \frac{20036983}{7484400}, \frac{2280356863}{681080400}, \frac{49020204823}{11675664000},\right.$$
$$\frac{65967241200001}{12504636144000}, \frac{15773461423793767}{2375880867360000}, \frac{655889589032992201}{78404068622880000},$$
$$\frac{94020690191035873697}{8910391798788480000}, \frac{65578224979953171437548 9}{4922991468830635200000 0},$$
$$\frac{44737200694996264619809969}{26584153931685430080000 00}, \frac{10129509912509255673830968079}{47616911012930667408000 0000},$$
$$\frac{10802634947676204112783980061 7281}{40150579366103138758425600000 000}, \frac{10954814567103825758202995557819063}{32177821463405515490681088000 0000},$$
$$\frac{61154674195324330125295778531172438727}{14190419265361832331390359808000000 00},$$
$$\frac{54441029530574028687402753586278549396607}{99758647435493681289674229450240000 0000},$$
$$\left.\frac{45201583278660966562457941005618082456 2551}{65388021008138715467181427706880000 00000}\right\}$$

```
N[%, 24]
```

```
{1.0000000000000000000000, 1.1666666666666666666667,
  1.4111111111111111111111, 1.7337301587301587301587,
  2.1485802469135802469135, 2.6771662391106835551280,
  3.3481463612812819162025 5, 4.1984939634268337972041 7,
  5.2754226864612559013776 3, 6.6389950946154611067619 8,
  8.3655045019129206006055 7, 10.551802021076064565597 9,
  13.320808170226233526129 6, 16.828521535783905604242 1,
  21.272925305378428379447 4, 26.905302783242667884999 9,
  34.044612310259338520908 0, 43.095748653882346445562 2,
  54.572742243500139401909 5, 69.128232636119720461815 0}
```

Though we are not advocating this many terms, we note that a degree twenty expression has relative accuracy better than 10^−16 for $u < 0.72$ and better than 10^{-20} for $u < 0.675$, so is itself a simple checking tool.

```
edata = Table[{u, SuperPolyQuantileShort[u, polydata, 20] /
      N[QuantileN[u], 200] - 1},
   {u, 1 / 2 + 1 / 1000, 80 / 100, 1 / 1000}];
rawlogs = Map[{#[[1]], Log[10, Abs[#[[2]]]]} &, edata];
plotdata = Select[rawlogs, NumberQ[#[[2]]] &];
ListPlot[plotdata, PlotJoined → True,
  PlotRange → {{0.5, 0.8}, {-50, 0}},
  Epilog → Line[{{0.5, -16}, {0.8, -16}}]];
```




```
exdata = Table[q[k], {k, 0, 20}];
```

For example

```
N[SuperPolyQuantileShort[u, exdata, 20] /
    N[QuantileN[u], 200] - 1 /. u → 72 / 100]
```

$$-8.52987 \times 10^{-17}$$

```
N[SuperPolyQuantileShort[u, exdata, 20] / N[QuantileN[u], 200] -
   1 /. u → 675 / 1000]
```

$$-8.51869 \times 10^{-21}$$

■ Hybrid polynomial variations of Acklam's method - fixed length polynomials

It seems that the lack of use of the truncated power series might be most useful in the Acklam level of approximation, becase we can get out to $u = 0.7$ with a degree 10 polynomial and keep the relative error down to less than 3.5×10^{-10}. We do have to make one more switch in the code, but also do not have to do a division on the most central interval. In fact in a pure Monte Carlo method, the switch could be avoided altogether by grouping the samples appropriately. Let's create a target accuracy of 5×10^{-10}, which is less than half the standard Acklam formula, and see what we can achieve. First we note that for this target the polynomial is good out to just above 0.7036

```
SuperPolyQuantileShort[u, polydata, 10] /
   N[QuantileN[u], 200] - 1 /. u → 0.7036
```

$$-4.98005 \times 10^{-10}$$

Let's see what the accuracy is using a (5,5) rational on the original interval:

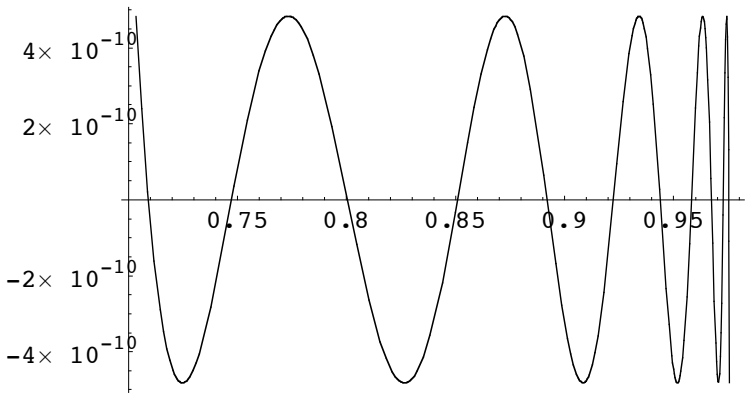
```
revrelAckC =
 MiniMaxApproximation[rootthing, {y, {0.2036^2, 0.47575^2},
    5, 5}, Brake → {70, 70}, MaxIterations → 50,
   WorkingPrecision → 25, PlotFlag → False][[2]]
```

$$\{(-48.92434080245331694782753\, y^5 + 249.31724543067266250509 68\, y^4 - 297.58204854998369631845 56\, y^3 + 144.57706089839803492051 66\, y^2 - 31.27420504146166039495158\, y + 2.50662856761028732328 9097)/(-64.14894607910946818205980\, y^5 + 179.54574405871416733561 28\, y^4 - 166.63823473258408360807 35\, y^3 + 69.53678255899736379967468\, y^2 - 13.52379108336019763885320\, y + 1),\ -4.82669186057616940840 1350 \times 10^{-10}\}$$

```
RatQuantileHybrid[x_] =
  (x - 1 / 2) revrelAckC[[1]] /. y → (x - 1 / 2)^2;
```

Here is a plot of the relative error over the region:

```
Plot[RatQuantileHybrid[x] / QuantileN[x] - 1,
  {x, 0.7036, 0.97575}, PlotRange → All];
```



```
Max[Table[Abs[RatQuantileHybrid[u] / QuantileN[u] - 1],
  {u, 0.7036, 0.97575, 0.00001}]]
```

4.82944×10^{-10}

So that comes out right. So we can make a hybrid with less than half the error with a similar amount of computation.

Alternatively, what about stretching the interval over which we have s simple formula while still keeping the relative error below 10^{-9}? The polynomial can be used out to $u = 0.71072$:

```
SuperPolyQuantileShort[u, polydata, 10] /
   N[QuantileN[u], 200] - 1 /. u → 0.71072
```

-9.99364×10^{-10}

After some experimentation, we see that

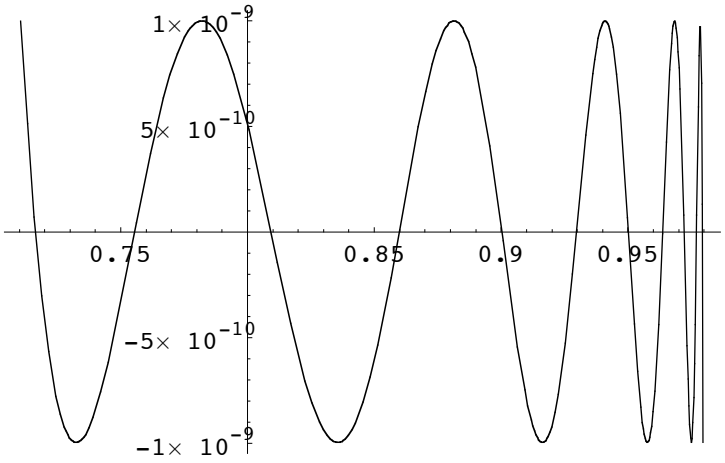
```
revrelAckC =
 MiniMaxApproximation[rootthing, {y, {0.21072^2, 0.4797^2},
    5, 5}, Brake → {70, 70}, MaxIterations → 50,
   WorkingPrecision → 25, PlotFlag → False][[2]]
```

$$\{(-56.763181316285490798201 09\, y^5 + 273.484570915004698116903 2\, y^4 - 316.066406392567673845101 6\, y^3 + 149.895476930850942607319 2\, y^2 - 31.796121280526494699027 21\, y + 2.506628878924548383753761)/ (-72.373879085463048386680 11\, y^5 + 194.853075884500355512170 0\, y^4 - 175.979343178389424676716 7\, y^3 + 71.876315370651555014168 78\, y^2 - 13.731996955555435022968 089\, y + 1),\ -9.978832331386022287522457 \times 10^{-10}\}$$

```
RatQuantileHybrid[x_] =
  (x - 1 / 2) revrelAckC[[1]] /. y → (x - 1 / 2)^2;
```

Here is a plot of the relative error over the region:

```
Plot[RatQuantileHybrid[x] / QuantileN[x] - 1,
  {x, 0.71072, 0.9797}, PlotRange → All];
```



```
Max[Table[Abs[RatQuantileHybrid[u] / QuantileN[u] - 1],
  {u, 0.71072, 0.9797, 0.00001}]]
```

9.98275×10^{-10}

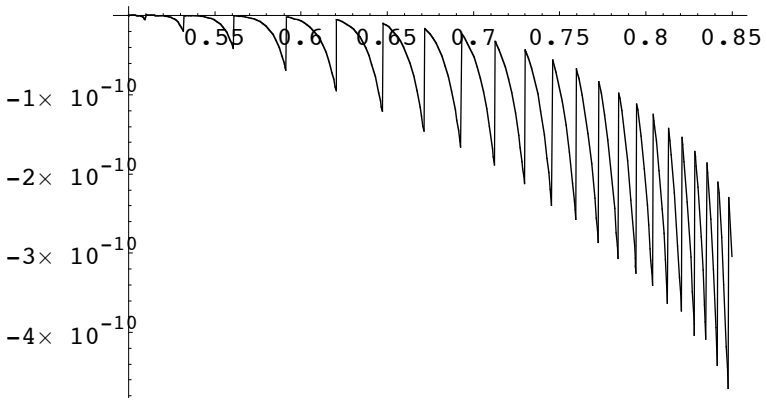
So further optimizations can indeed be made in two ways: the first more than halves the error over the same interval, the second allows a very small stretch of the interval while keeping the error smaller than the original scheme. In some ways it is a reflection of the quality of Acklam's original scheme that the benefits of further optimization are in fact quite small. But we are not quite done.

■ Hybrid polynomial variations of Acklam's method - variable length polynomials

The other approach we can try is based on the fact that even quite a short polynomial wastes some evaluations in a neighbourhood of $u = 0.5$. Recall the following definition:

```
SbNormalQuantileReport[u_, ratios_, tol_] :=
 Module[{b = Sqrt[Pi] * (u - 1 / 2), q , i = 0, s, t = 0},
  s = b; q = b * b;
  While[s - t > 10 ^ (-tol),
   (t = s; b *= q * ratios[[i += 1]]; s = t + b)];
  {Sqrt[2] * s, i}]
```

```
Plot[SbNormalQuantileReport[u, algratios, 9.4][[1]] /
    QuantileN[u] - 1, {u, 0.5, 0.85}, PlotRange → All];
```

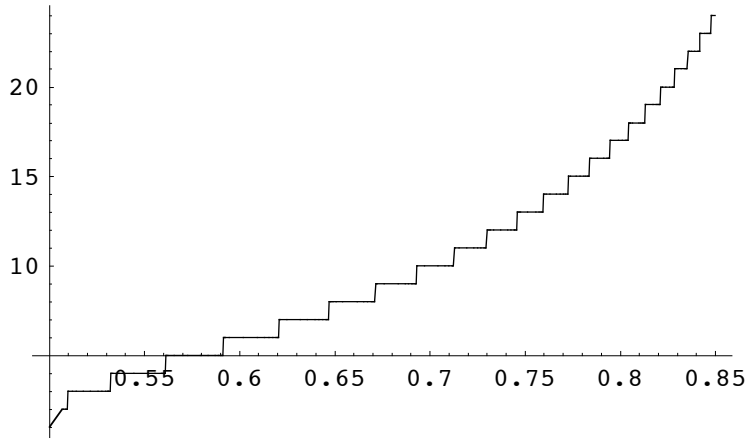



```
edata = Table[SbNormalQuantileReport[u, algratios, 9.4][[1]] /
     N[QuantileN[u], 30] - 1, {u, 501 / 1000, 850 / 1000, 1 / 1000}];
```

```
Max[Abs[edata]]
```

$4.23492700188231021050 \times 10^{-10}$

```
Plot[SbNormalQuantileReport[u, algratios, 9.4][[2]],
  {u, 0.5, 0.85}, PlotRange → All];
```




```
Mean[Table[SbNormalQuantileReport[u, algratios, 9.4][[2]],
   {u, 0.5, 0.85, 0.0001}]] // N
```

9.94487

So we can get out to $u = 0.85$ working out a polynomial whose average length is 10 terms, with relative error less than $5 * 10^{-10}$.

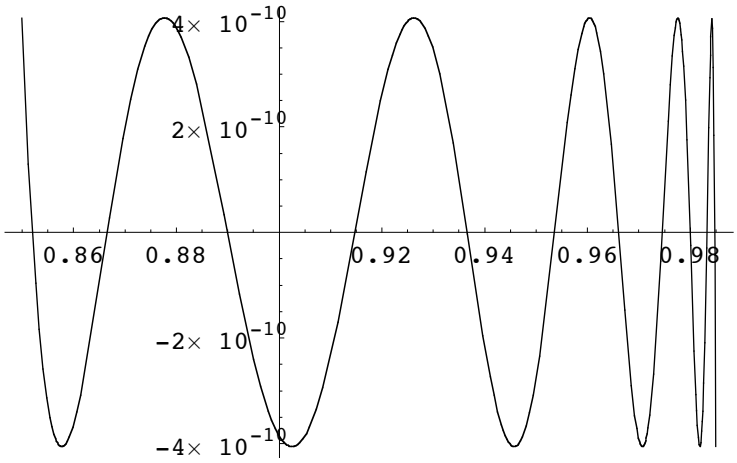
```
revrelAckD =
 MiniMaxApproximation[rootthing, {y, {0.35^2, 0.485^2}, 5, 5},
   Brake → {70, 70}, MaxIterations → 50,
   WorkingPrecision → 25, PlotFlag → False][[2]]
```

$$\{(-114.947487696219715815476 2\, y^5 + 413.947331902647505424893 5\, y^4 - 411.384636245746988322656 9\, y^3 + 175.161855231048646317176 4\, y^2 - 34.1241033774290411783238 9\, y + 2.50667514627875846312300 9)/(-125.047034998646798964847 5\, y^5 + 280.035693092524052293594 3\, y^4 - 223.287889364371161256041 9\, y^3 + 82.9136050118953472639924 1\, y^2 - 14.6599362773866262221704 6\, y + 1),\ -4.06017218006560649348536 0 \times 10^{-10}\}$$

```
RatQuantileHybridA[x_] =
  (x - 1 / 2) revrelAckD[[1]] /. y → (x - 1 / 2)^2;
```

Here is a plot of the relative error over the region:

```
Plot[RatQuantileHybridA[x] / QuantileN[x] - 1,
  {x, 0.85, 0.985}, PlotRange → All];
```




```
Max[Table[Abs[RatQuantileHybridA[u] / QuantileN[u] - 1],
  {u, 0.85, 0.985, 0.00001}]]
```

4.07026×10^{-10}

So the hybrid polynomial-rational form of "Acklam style", i.e. the rational part is (5,5) and the average length of the polynomial is about 10, is capable of getting a relative error less than $5 * 10^{-10}$ on the interval $0.5 < u < 0.985$, and to do so without an increase in net effort. A suitable tail partner for this is under consideration.

10 A Look at the Moro Tail Expression

We already looked at the Acklam and Wichura AS241 formulae into the tails. Let's go back to Moro's formula.

■ Moro's Formula for the tail

▪ Dealing with the Tail Issue

Moro's main contribution in his RISK article was to produce an expression for the tail using a polynomial in

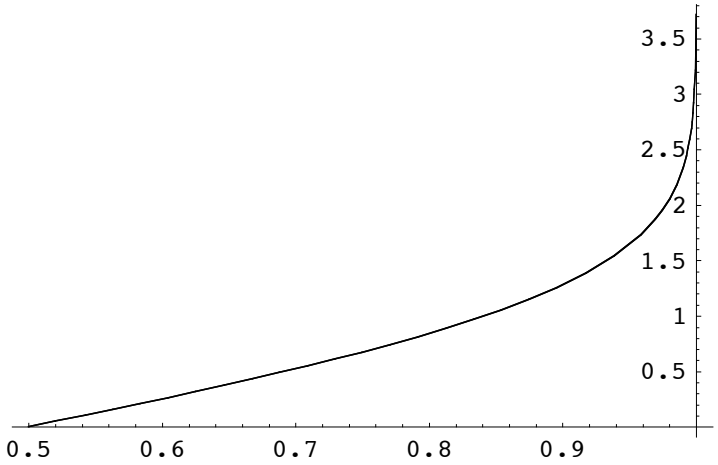
$$r = \log(-\log(1 - u)) \tag{4}$$

```
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]
```

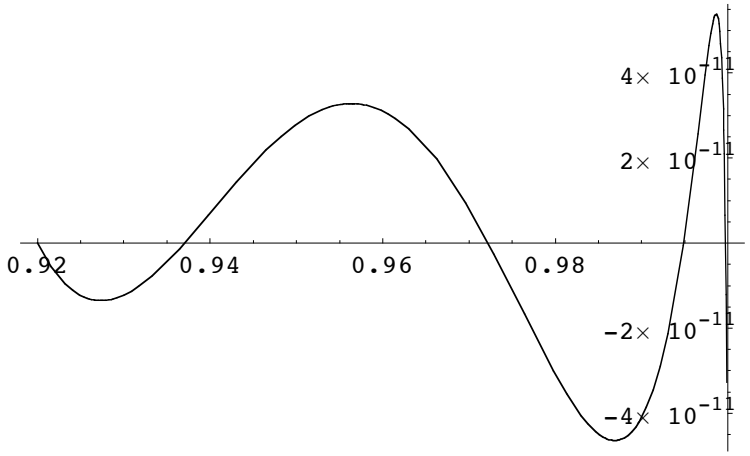
1.95996

Superficially this looks good everywhere!

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




```
Plot[MoroTail[u] / QuantileN[u] - 1 ,
  {u, 0.92, 0.9999}, PlotRange → All];
```



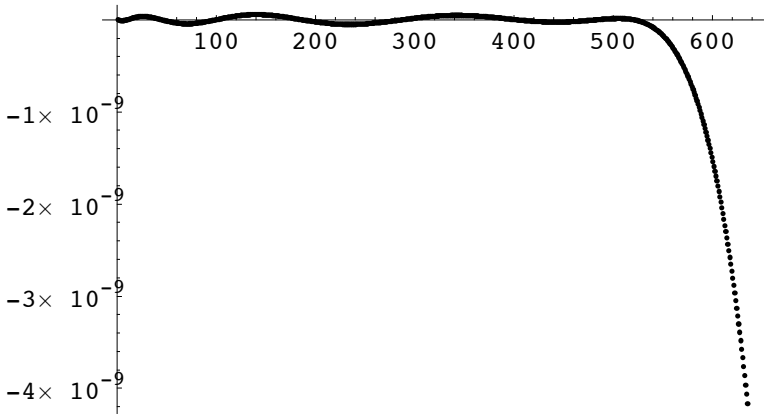
Let's probe the tail properly. Here is a parametric plot using $t = \sqrt{-2\,\mathrm{Log}(1-u)}$

```
Sqrt[-2 Log[1 - 0.92]]
```

2.24754

```
mtaildata = Table[MoroTail[1 - Exp[-t^2 / 2]] /
     N[QuantileN[1 - Exp[-t^2 / 2]], 40] - 1,
   {t, 224754 / 100000, 860000 / 100000, 1 / 100}];
```

```
ListPlot[mtaildata, PlotRange → All];
```

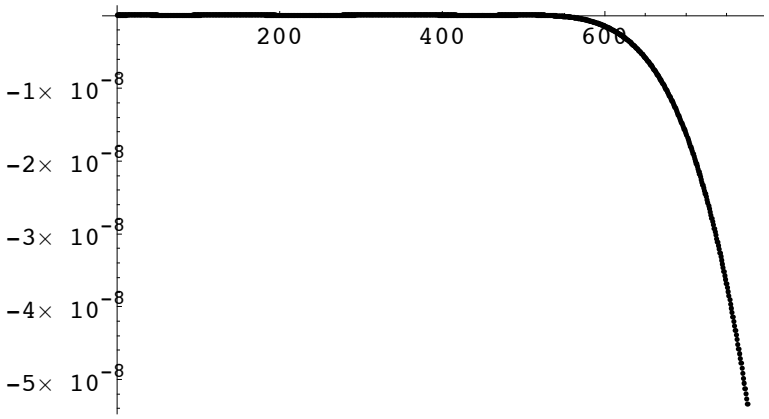



```
mtaildata = Table[MoroTail[1 - Exp[-t^2 / 2]] /
     N[QuantileN[1 - Exp[-t^2 / 2]], 40] - 1,
   {t, 224754 / 100000, 10, 1 / 100}];
```

```
Exp[-50] // N
```

1.92875×10^{-22}

```
ListPlot[mtaildata, PlotRange → All];
```




```
relMoro[10] // N
```

-5.3623×10^{-8}

So the Moro tail expression starts to behave badly in the far tail, with error passing through $5 * 10^{-8}$ at $u = 2 * 10^{-22}$. Note that Acklam's formula remains good down to at least 10^{-30} from my own testing.

```
relMoro[t_] := MoroTail[1 - Exp[-t^2 / 2]] /
   N[QuantileN[1 - Exp[-t^2 / 2]], 400] - 1
```

```
$MaxExtraPrecision = 4000
```

4000

```
relMoro[120] // N
```

−0.00628473

So the error does keep growing, but not at values of samples that are attainable with current double precision technology!!

```
Exp[-120^2 / 2] // N
```

$1.201518042084 \times 10^{-3127}$