Quantile Mechanics II: Changes of Variables in Monte Carlo methods and a GPU-Optimized Normal Quantile

William T. Shaw*, N. Brickman†

August 27, 2010

Abstract

This article presents differential equations and solution methods for the functions of the form $A(z) = F^{-1}(G(z))$, where F and G are cumulative distribution functions. Such functions allow the direct recycling of Monte Carlo samples from one distribution into samples from another. The method may be developed analytically for certain special cases, and illuminate the idea that it is a more precise form of the traditional Cornish-Fisher expansion. In this manner the model risk of distributional risk may be assessed free of the Monte Carlo noise associated with resampling. The method may also be regarded as providing both analytical and numerical bases for doing more precise Cornish-Fisher transformations. Examples are given of equations for converting normal samples to Student t, and converting exponential to hyperbolic, variance gamma and normal. In the case of the normal distribution, the change of variables employed allows the sampling to take place to good accuracy based on a single rational approximation over a very wide range of the sample space. The avoidance of any branching statement is of use in optimal GPU computations, and we give example of branch-free normal quantiles that offer performance improvements in a GPU environment, while retaining the precision characteristics of well-known methods. Comparisons are made on Nvidia Quadro and GTX 285 and 480 GPU cards.

Keywords: Monte Carlo, Student, hyperbolic, variance gamma, computational finance, quantile mechanics, normal quantile, Gaussian quantile, GPU, Acklam, AS241.

*Corresponding author: Department of Mathematics King's College, The Strand, London WC2R 2LS, England; E-mail: william.shaw@kcl.ac.uk

†Taylor Brickman Ltd; E-mail: nick@taylorbrickman.co.uk

1 Introduction

The construction of Monte Carlo samples from a distribution is facilitated if one has a knowledge of the quantile function $w(u)$ of a distribution. If $F(x)$ is the cumulative distribution function, then the quantile $w(u)$ is the solution of the equation

$$F(w(u)) = u \ . \tag{1}$$

A knowledge of the function $w(u)$ makes Monte Carlo simulation straightforward: given a random sample U from the uniform distribution, a sample from the target distribution characterized by $f(x), F(x)$ is

$$X = w(U) \ . \tag{2}$$

While it is commonplace to use the uniform distribution on the unit interval as the base distribution for sampling, there is in fact no need to do so[1]. In his critique of copula theory [12], T. Mikosch stated *There is no particular mathematical or practical reason for transforming the marginals to the uniform distribution on (0; 1)* and proceeded to consider exponential and normal coordinates.

For example, a great deal of intellectual effort has been expended on highly efficient sampling from the normal and other well-known distributions. Given such samples, can we leverage the work done to create samples from other distributions in an efficient manner? This article will address this question in the affirmative. In principle the answer is trivial: given a sample Z from a distribution with CDF $G(x)$, we first work out $G(Z)$ which is uniform. Then we can apply the quantile function $F^{-1}(x)$ associated with a target distribution with CDF F and form $F^{-1}(G(z))$ as a sample from that target distribution. In general F, G and their inverses can be rather awkward special functions (see e.g. [15]) , so a direct route to the object $A(z) = F^{-1}(G(z))$ would be helpful.

There are at least two ways of developing this idea. One route is to *postulate* interesting forms for the composite mapping. This has been explored by Shaw and Buckley [17] based on Gilchrist's theory of quantile transformations [8]. In this way we can find skew and kurtotic variations of *any* base distribution, while avoiding, in a controlled manner, the introduction of "negative density" problems that arise in traditional Gram-Charlier methods. The second route is to try to simplify the mapping *given* a choice of F and G. Such a route can be found by the method of differential equations for quantile functions developed by Steinbrecher and Shaw [18]. In the next section we will give a brief review of that approach.

A particular application of our approach will be to present new methods of constructing the normal quantile by first filtering it through a two-sided exponential distribution. We will show that this offers a useful performance benefit in a GPU environment, where branching algorithms may be subject to significant performance penalties. Our change-of-variables approach will allow costly branching to be avoided and we will demonstrate the benefits in the CUDA environment for programming NVIDIA GPUs.

[1]This rather clear observation was first made to me by Peter Jaeckel

2 Quantile mechanics

If $f(x)$ is the probability density function for a real random variable X, the first order quantile ODE is obtained by differentiating Eqn. (1), to obtain:

$$f(w(u))\frac{dw(u)}{du} = 1, \tag{3}$$

where $w(u)$ is the quantile function considered as a function of u, with $0 \leq u \leq 1$. Applying the product rule with a further differentiation we obtain:

$$f(w(u))\frac{d^2w(u)}{du^2} + f'(w(u))\left(\frac{dw(u)}{du}\right)^2 = 0. \tag{4}$$

This may be reorganized as

$$\frac{d^2w(u)}{du^2} = H(w(u))\left(\frac{dw(u)}{du}\right)^2 , \tag{5}$$

where

$$H(w) = -\frac{d}{dw}\log\{f(w)\} . \tag{6}$$

and the simple rational form of $H(w)$ for many common distributions, particularly the Pearson family, allows analytical series solutions to be developed [18]. This last equation we refer to as the second order quantile equation.

2.1 The Recycling ODE

Now suppose that we make a change of *independent* variable in the second order quantile equation Eqn (8). We let $v = q(u)$, and regard w as a function of v. We write $w(u) = Q(v)$, where $v = q(u)$. Elementary application of the chain rule and some algebra gives us:

$$\frac{d^2Q(v)}{dv^2} + \frac{q''(u)}{[q'(u)]^2}\frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 , \tag{7}$$

In general this is a rather awkward differential equation. However, when we regard $q(u)$ as being itself a quantile function, we can make some simplifications. If $q(u)$ is a quantile mapping, it satisfies an ODE of the form

$$\frac{d^2q(u)}{du^2} = \hat{H}(q(u))\left(\frac{dq(u)}{du}\right)^2 , \tag{8}$$

where

$$\hat{H}(w) = -\frac{d}{dw}\log\{\hat{f}(w)\} . \tag{9}$$

and $\hat{f}$ is the probability density function associated with the quantile $q(u)$. So we can simplify the ODE to

$$\frac{d^2Q(v)}{dv^2} + \hat{H}(q(u))\frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 , \tag{10}$$

and bearing in mind that $v = q(u)$ we arrive at the "Recycling Ordinary Differential Equation":

$$\frac{d^2Q(v)}{dv^2} + \hat{H}(v)\frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 , \tag{11}$$

We now turn to two particularly interesting cases, rather inspired by Mikosch's suggestions [12].

2.2 The Recycling ODE for a Gaussian background

In this case we have the following obvious sequence of manipulations:

$$\hat{f}(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2} \tag{12}$$

$$\log \hat{f}(x) = -1/2\log(2\pi) - x^2/2 \tag{13}$$

$$\frac{d}{dx}\log \hat{f}(x) = -x \tag{14}$$

$$\hat{H}(v) = v \tag{15}$$

and we arrive at the Recycling ODE for a Gaussian background as

$$\frac{d^2Q(v)}{dv^2} + v\frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 , \tag{16}$$

This is an interesting example to consider for target distributions along the entire real line.

2.3 The Recycling ODE for a one-sided exponential background

In this case we have the following obvious sequence of manipulations:

$$\hat{f}(x) = e^{-x}, \quad \log \hat{f}(x) = -x, \quad \frac{d}{dx}\log \hat{f}(x) = -1, \quad \hat{H}(v) = 1 \tag{17}$$

and we arrive at the Recycling ODE for a exponential background as

$$\frac{d^2Q(v)}{dv^2} + \frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 , \tag{18}$$

This is an interesting example to consider for target distributions along the positive real line. For distributions that are asymptotically exponential in both directions it can be used in two pieces.

3 Example with a Gaussian background

In a Gaussian background we work with the Recycling ODE in the form

$$Q'' + vQ' = H(Q)(Q')^2 \tag{19}$$

where the explicit dependence on v is suppressed for brevity, and ' denotes d/dv. The target distribution is encoded through the function H. Note that it is not required in any sense that the target distribution is "close" to, or asymptotic to a Gaussian. This is an *exact* relationship governing the function Q that is the composition of the Gaussian CDF followed by the ordinary quantile of the target distribution. But such a relationship must contain all information relevant to the creation of an expansion of one distribution in terms of another. In particular, we should be able to re-create known and new expansions of Cornish-Fisher type. Generalized Cornish-Fisher expansions have been considered in the notable paper by Hill and Davis [9], but the step to considering the matter as the solution of a single differential equation is, so far as this author is aware, a new one.

3.1 The Student distribution

This is an interesting case for several reasons:

1. We can illustrate the method;
2. We can recover a well known asymptotic series;
3. We can develop that series to arbitrary numbers of terms;
4. We can explore the limitations of the known series;
5. We can develop an alternative numerical method and explore purely numerical options.

The H-function for the Student case can be written down as

$$H_{T_n}(Q) = \left(1 + \frac{1}{n}\right) \frac{Q}{1 + Q^2/n} \tag{20}$$

and the Recycling ODE can be written in the form

$$\left(1 + \frac{Q^2}{n}\right)\left(Q'' + vQ'\right) = \left(1 + \frac{1}{n}\right) Q(Q')^2 \tag{21}$$

We note that if we let $n \to \infty$ we obtain

$$Q'' + vQ' = Q(Q')^2 \tag{22}$$

and this has the desired solution $Q = v$. More generally we can look at series solutions, but should be mindful of the fact that the term Q^2/n is present - this is a hint that the behaviour of series for $Q \ll \sqrt{n}$ and $Q \gg \sqrt{n}$ could be rather different. Such considerations do not always apply if one is thinking in a purely asymptotic framework. For any *finite* n, no matter how large, there will always be values of Q such that the behaviour is far from Gaussian. This was alluded to in [15], where it was noted that the known Cornish-Fisher expansion always goes wrong in the tails as some point.

We also need to consider boundary conditions. The derivative of any ordinary quantile function at a point is the inverse of the PDF at the corresponding quantile. We first work around the point $u = 1/2$ which corresponds to $v = 0$

in the Gaussian coordinate. If $z(u)$ and $t(u)$ are the ordinary quantiles then we have

$$\begin{aligned} z(1/2) &= 0, \\ z'(1/2) &= \sqrt{2\pi} \\ t(1/2) &= 0, \\ t'(1/2) &= \sqrt{n\pi}\frac{\Gamma[n/2]}{\Gamma[(n+1)/2]} \end{aligned} \tag{23}$$

It follows that the centre conditions we wish to apply to the Recycling ODE are just:

$$\begin{aligned} Q(0) &= 0, \\ Q'(0) &= \gamma \equiv \sqrt{\frac{n}{2}}\frac{\Gamma[n/2]}{\Gamma[(n+1)/2]} \end{aligned} \tag{24}$$

where the latter expression γ arises as the ratio of the derivatives.

3.2 The central expansion

We now develop a series solution about the centre, and we expect that it will be reasonable to treat the solution as "close to Gaussian" if $Q^2 \ll n$. We assume, as both the normal and Student quantiles are symmetric, that

$$Q(v) \sim \sum_{k=0}^{\infty} c_k v^{2k+1} \tag{25}$$

where $c_0 = \gamma$. We use the tilde notation to indicate that at this point we have no presumption as to whether the resulting series will be convergent for all v or form some kind of asymptotic series. We find that

$$\begin{aligned} c_1 &= \frac{(n+1)\gamma^3 - n\gamma}{6n} \\ c_2 &= \frac{\left(7n^2+8n+1\right)\gamma^5 + \left(-10n^2-10n\right)\gamma^3 + 3n^2\gamma}{120n^2} \end{aligned} \tag{26}$$

Subsequent terms may be generated by iteration of the RODE, and in this case, after some algebra, we find that

$$\begin{aligned} (2i+3)(2i+2)c_{i+1} = &-(2i+1)c_i \\ &+ \sum_{l=0}^{i}\sum_{m=0}^{i-l} a_{lm}(n) c_{i-l-m}c_l c_m \\ &- \frac{\theta(i)}{n}\sum_{l=0}^{i-1}\sum_{m=0}^{i-1-l}(2m+1)c_{i-1-l-m}c_l c_m, \end{aligned} \tag{27}$$

where $\theta(0) = 0, \theta(i) = 1$ if $i \geq 1$, and

$$a_{lm}(n) = (1+\frac{1}{n})(2l+1)(2m+1) - \frac{2}{n}m(2m+1) \tag{28}$$

3.3 The tail expansion

We now develop a series solution about the right tail $Q \to \infty$. We begin by assuming that $Q^2 \gg n$. The Recycling ODE becomes

$$Q(Q'' + vQ') = (n+1)(Q')^2 \tag{29}$$

Following some experimentation, we make the change of variables

$$P(v) = \frac{1}{Q(v)^n} \tag{30}$$

and this reduces the ODE to

$$P''(v) + vP'(v) = 0 \tag{31}$$

The solution of this satisfying the condition that $P(v) \to 0$ as $v \to \infty$ is

$$P(v) \propto \text{erfc}(\frac{v}{\sqrt{2}}) \tag{32}$$

and we deduce that for some constant c,

$$Q(v) \sim d\left[\frac{1}{2}\text{erfc}(\frac{v}{\sqrt{2}})\right]^{-1/n} \tag{33}$$

We see that the solution has emerged naturally as

$$Q(v) \sim d\Big[1 - \Phi(v)\Big]^{-1/n} \tag{34}$$

where Φ is the Gaussian CDF. The asymptotic differential equation is scale invariant so we have to determine d by other means. It is possible that it might be possible to determine it by a matching argument, but it is simpler to now appeal to other known properties of the Student distribution. In [15] the tail behaviour of the Student CDF was determined (see Eqns. (60-62) of [15]) and we can deduce that

$$d = \sqrt{n}\left[n\sqrt{\pi}\frac{\Gamma(n/2)}{\Gamma((n+1)/2)}\right]^{-1/n} \tag{35}$$

If we step back from these calculations it becomes clear what is happening. The Recycling ODE is starting to reconstruct a solution that combines the change of variable $w = 1 - \Phi(v)$ with the asymptotic power series of the ordinary Student quantile.

3.4 Comparison with traditional asymptotics

Expansions of Cornish-Fisher type can be found in the statistics literature. One that is reasonably well known is the expansion of the Student random variable t in terms of the Gaussian random variable z, for larger values of the degrees of freedom n. It is quoted, for example, as identity 26.7.5 of [3].

$$\begin{aligned} t =& z + \frac{z^3+z}{4n} + \frac{5z^5+16z^3+3z}{96n^2} + \frac{3z^7+19z^5+17z^3-15z}{384n^3} \\ &+ \frac{79z^9+776z^7+1482z^5-1920z^3-945z}{92160n^4} + \dots \end{aligned} \tag{36}$$

An equation of true Cornish-Fisher type (cf identity 26.2.49 of [3]) can be obtained by transforming (provided $n > 2$) to a variable s with *unit variance*: $s = t\sqrt{1-2/n}$ and re-expanding in inverse powers of n. That Eqn. (36) is somehow incomplete is evident by the fact that z appears in every term, z^3 in all but the first, and so on. The matter is resolved nicely by first observing that

$$\gamma = 1+\frac{1}{4n}+\frac{1}{32}\left(\frac{1}{n}\right)^2-\frac{5}{128}\left(\frac{1}{n}\right)^3-\frac{21\left(\frac{1}{n}\right)^4}{2048}+\frac{399\left(\frac{1}{n}\right)^5}{8192}+O\left(\left(\frac{1}{n}\right)^6\right)\ , \quad (37)$$

which sums up all the z-terms. Similarly

$$c_2 = \frac{1}{4n}+\frac{1}{6}\left(\frac{1}{n}\right)^2+\frac{17}{384}\left(\frac{1}{n}\right)^3-\frac{1}{48}\left(\frac{1}{n}\right)^4-\frac{17\left(\frac{1}{n}\right)^5}{8192}+O\left(\left(\frac{1}{n}\right)^6\right) \quad (38)$$

and so on. So the series solution of the differential equation constitutes a resummation of the known asymptotic series where the coefficient of each power of z is computed exactly.

3.5 Accuracy and numerical methods

We now turn to the quality of the results. This can be assessed precisely by the use of an exact representation of the composite function $F_N^{-1}(\Phi(z))$, where Φ is the normal CDF and F_n the Student CDF. The exact formula for the Student CDF for all real n is given in terms of inverse beta functions by Shaw [15], and there are known simpler forms for $n = 1, 2, 4$. These are also given in [15] and are also now available on the *Wikipedia* page on quantile functions [14]. The case $n = 4$ is an interesting case as it is known exactly, is the boundary case where kurtosis is infinite, and there is some evidence from work by Fergusson and Platen [7] that it is a good case for modelling daily world index log-returns. We shall therefore develop this in some detail. It turns out that working as far as c_{10} is a useful point. A detailed calculation shows that the precision (i.e. relative error) of the central power series is then less than 2×10^{-5} on $|z| < 4$. For this case we find that

$$\gamma = \frac{4}{3}\sqrt{\frac{2}{\pi}} \sim 1.06384608107048714 \quad (39)$$

and the full C-code form for the central series is, with $y = z * z$,

```
t = z*(1.06384608107048714 +
      y*(0.0735313753642658509 +
        y*(0.00408737916150927847 +
           y*(0.00015737627666323056 2 +
              y*(4.3193924140363509e-6 +
                 y*(9.5688146463922728e-8 +
                    y*(2.0925688180361446e-9 +
                    y*(3.8796293820933352e-11 +
                    y*(2.7232608454191567e-13 +
                    (2.9052893016237328e-15 +
                    4.5949013399590137 5e-16*y)*y)))))))
        ))
```

To treat the tail regions $|z| > 4$ with corresponding accuracy when $n = 4$ it is sufficient to use just *two* terms of the known tail series. This gives us, in general, for the positive tail (the negative tail being treated by symmetry)

$$\begin{aligned} w &= (1 - \Phi(z))n\sqrt{\pi}\frac{\Gamma(n/2)}{\Gamma((n+1)/2)} \\ t &= \sqrt{n}w^{-1/n}(1 - \frac{n+1}{2(n+2)}w^{2/n}) \end{aligned} \tag{40}$$

and for the case $n = 4$:

$$\begin{aligned} w &= (1 - \Phi(z))\frac{16}{3} \\ t &= 2w^{-1/4}(1 - \frac{5}{12}w^{1/2}) \end{aligned} \tag{41}$$

The optimal crossover is then in fact at $z = 3.93473$ with maximum relative error less than 1.4×10^{-5} over the entire range of z

3.6 Purely numerical methods

The analysis for the Student t case, although rather specialized, also allows the appraisal of direct numerical schemes. The direct numerical solution of the RODE can be done using standard methods. Within *Mathematica* version 6, the use of `NDSolve` with high precision and accuracy goals, explicit Runge-Kutta and sixth-order differences leads to an precision of better than 5×10^{-8} on the range $|z| < 6$, which is excellent. Of course, one must also consider sampling efficiency issues arising from such interpolated numerical schemes, but they can be made the basis of a further, e.g. rational approximation if speed is an issue. Such a numerical scheme will be exploited in the examples considered below.

4 Hyperbolic and Variance Gamma

In this section we move to other distributions of interest to finance. First we consider the *hyperbolic* distribution, and then the variance gamma. These will have in common a non-normal base distribution, and will illustrate the use of a 2-sided exponential base instead.

4.1 Hyperbolic quantile from exponential base

This was originally motivated by Bagnold's classic study of sand [4], and was given a clear mathematical description by Barndorff-Nielsen [5], who also generalized it. The applications to finance have been explored Eberlein and Keller [6]. A direct treatment of the quantile function for the symmetric case has been given by Xiong [21]. He we shall explore the conversion of samples from a suitable exponential distribution to samples from the hyperbolic. Hyperbolic distributions can of course be sampled as random mixtures of a normal distribution. Our method facilitates the use of hyperbolic marginals coupled to an arbitrary copula, and and this example also illustrates how cleanly the choice of a suitable base simplifies the computations of the quantile - the exponential base regularizes the tail in an elegant way.

The probability density function is known explicitly as

$$f(x,\alpha,\beta,\delta,\mu) = \frac{\gamma}{2\alpha\delta K_1(\delta\gamma)} \exp\{-\alpha\sqrt{\delta^2+(x-\mu)^2}+\beta(x-\mu)\} \tag{42}$$

where $\gamma = \sqrt{\alpha^2-\beta^2}$, with $|\beta| < \alpha > 0$. In what follows we shall translate the origin so that $\mu = 0$, with density

$$f(x,\alpha,\beta,\delta) = \frac{\gamma}{2\alpha\delta K_1(\delta\gamma)} \exp\{-\alpha\sqrt{\delta^2+x^2}+\beta x\} \tag{43}$$

The H-function for the target distribution is given by the negative of the logarithmic derivative:

$$H(x) = -\frac{d}{dx}\log f(x,\alpha,\beta,\delta) = \frac{\alpha x}{\sqrt{\delta^2+x^2}} - \beta \tag{44}$$

and it is evident that for large x,

$$H(x) \sim \text{sign}(x)\alpha - \beta = \pm\alpha - \beta \ . \tag{45}$$

Bearing mind that the exponential distribution is characterized by a constant H-function, we will use a pair of exponential distributions for the base case. In order to get the proportion of the random variables that are positive and negative correct, we let

$$\begin{aligned} p_+ &= \int_0^\infty dx \frac{\gamma}{2\alpha\delta K_1(\delta\gamma)} \exp\{-\alpha\sqrt{\delta^2+x^2}+\beta x\} \\ p_- &= \int_{-\infty}^0 dx \frac{\gamma}{2\alpha\delta K_1(\delta\gamma)} \exp\{-\alpha\sqrt{\delta^2+x^2}+\beta x\} \end{aligned} \tag{46}$$

so clearly $p_+ + p_- = 1$.

$$f_0(x) = \begin{cases} p_+(\alpha-\beta)e^{-(\alpha-\beta)x} & \text{if } x > 0, \\ p_-(\alpha+\beta)e^{(\alpha+\beta)x} & \text{if } x < 0, \end{cases} \tag{47}$$

The quantile function for sampling from f_0 has the trivial form:

$$v = Q_0(u) = \begin{cases} \frac{1}{\alpha+\beta}\log(u/p_-) & \text{if } u < p_-, \\ \frac{-1}{\alpha-\beta}\log((1-u)/p_+) & \text{if } u > p_-, \end{cases} \tag{48}$$

So samples from the base can be made easily. To convert them into samples from the hyperbolic we solve a *left* and *right* differential equation. The right problem is of the form

$$\frac{d^2Q}{dv^2} + (\alpha-\beta)\frac{dQ}{dv} = \left(\frac{\alpha Q}{\sqrt{\delta^2+Q^2}} - \beta\right)\left(\frac{dQ}{dv}\right)^2 \tag{49}$$

on $v > 0$ with the initial condition $Q(0) = 0$ and

$$\frac{dQ}{dv}|_{v=0} = \frac{Q'(p_-)}{Q_0'(p_-)} = \frac{f_0(0_+)}{f(0)} = p_+(\alpha-\beta)2\frac{\alpha\delta}{\gamma}K_1(\delta\gamma)e^{\alpha\delta} \tag{50}$$

The left problem is

$$\frac{d^2Q}{dv^2} - (\alpha+\beta)\frac{dQ}{dv} = \left(\frac{\alpha Q}{\sqrt{\delta^2+Q^2}} - \beta\right)\left(\frac{dQ}{dv}\right)^2 \tag{51}$$

on $v < 0$ with the initial condition $Q(0) = 0$ and

$$\frac{dQ}{dv}|_{v=0} = \frac{Q'(p_-)}{Q_0'(p_-)} = \frac{f_0(0_-)}{f(0)} = p_-(\alpha+\beta)2\frac{\alpha\delta}{\gamma}K_1(\delta\gamma)e^{\alpha\delta} \tag{52}$$

The solution to this differential system is readily visualized as a kind of 'QQ' plot. If we use a sixth-order explicit RK method as before, with parameters $\alpha = 1 = \delta, \beta = 0$ for illustration, the result is show below, together with the identity map (diagonal line).

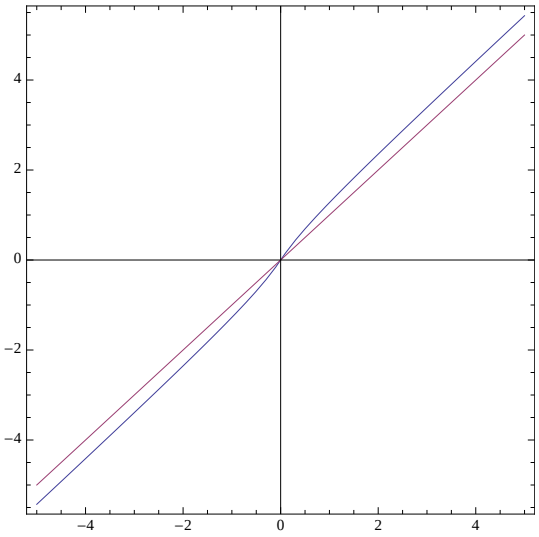

Figure 1: QQ Plot for conversion of exponential to hyperbolic

4.2 VG quantile from exponential base

The variance-gamma density was introduced by Madan and Seneta [11] as a model for share market returns. The density is given, for $\lambda > 0, \alpha > 0, |\beta| < \alpha$, by

$$\frac{e^{\beta x}|x|^{\lambda-\frac{1}{2}}\left(\alpha^2-\beta^2\right)^{\lambda}K_{\lambda-\frac{1}{2}}(\alpha|x|)}{(2\alpha)^{\lambda-1/2}\sqrt{\pi}\Gamma(\lambda)} \tag{53}$$

In the region $x > 0$ the H-function is given by

$$H(x) = -\frac{d}{dx}\log(f) = \frac{\alpha K_{\lambda-3/2}(\alpha x)}{K_{\lambda-1/2}(\alpha x)} - \beta \sim (\alpha-\beta) + \frac{1-\lambda}{x} + O\left(\left(\frac{1}{x}\right)^2\right) \tag{54}$$

In the region $x < 0$ the H-function is given by

$$H(x) = -\frac{d}{dx}\log(f) = -\frac{\alpha K_{\lambda-3/2}(-\alpha x)}{K_{\lambda-1/2}(-\alpha x)} - \beta \sim -(\alpha+\beta) + \frac{1-\lambda}{x} + O\left(\left(\frac{1}{x}\right)^2\right) \tag{55}$$

These asymptotic relationships suggest that the VG model may be treated in a similar way to the hyperbolic case, as the asymptotics are closely related with a good match to the exponential base. This time the probabilities $p_\pm$ are given by

$$\begin{aligned} p_+ &= \frac{\left(\alpha^2-\beta^2\right)^\lambda}{(2\alpha)^{\lambda-1/2}\sqrt{\pi}\Gamma(\lambda)}\int_0^\infty dx e^{\beta x}x^{\lambda-\frac{1}{2}}K_{\lambda-\frac{1}{2}}(\alpha x) \\ &= \frac{2^{2\lambda-1}\left(\frac{\alpha+\beta}{\alpha-\beta}\right)^\lambda\Gamma\left(\lambda+\frac{1}{2}\right)\,{}_2F_1\left(2\lambda,\lambda;\lambda+1;\frac{\alpha+\beta}{\beta-\alpha}\right)}{\sqrt{\pi}\Gamma(\lambda+1)}, \\ p_- &= \frac{\left(\alpha^2-\beta^2\right)^\lambda}{(2\alpha)^{\lambda-1/2}\sqrt{\pi}\Gamma(\lambda)}\int_0^\infty dx e^{-\beta x}x^{\lambda-\frac{1}{2}}K_{\lambda-\frac{1}{2}}(\alpha x) \\ &= \frac{2^{2\lambda-1}\left(\frac{\alpha-\beta}{\alpha+\beta}\right)^\lambda\Gamma\left(\lambda+\frac{1}{2}\right)\,{}_2F_1\left(2\lambda,\lambda;\lambda+1;\frac{\beta-\alpha}{\alpha+\beta}\right)}{\sqrt{\pi}\Gamma(\lambda+1)}, \end{aligned} \tag{56}$$

where we have used identity 6.621.3 from [3] to evaluate the integrals giving the probabilities that the VG random variables is positive or negative. It is easily checked that if $\beta = 0$ then $p_+ = p_- = 1/2$.

The difference between VG and hyperbolic is that in the case of VG the details of what has to be done are sensitive to the value of λ. First, we note that if $\lambda = 1$ the VG model is trivial as it is identical to the base, so that $Q(v) \equiv v$. If $\lambda > 1$ matters remain reasonably straightforward, as both f and H exist at $v = 0$, with $H(0) = 0$. The recycling ODE may be solved as before, though many steps may be needed near $v = 0$ if λ remains close to and just above 1. When $0 < \lambda < 1$ matters are more complicated, as then $H(0)$ is divergent, and furthermore the density becomes singular in the range $0 < \lambda \leq 1/2$. The density has a log divergence when $\lambda = 1/2$, and otherwise diverges as $x^{2\lambda-1}$. All of these issues may in principle be addressed by doing analytical estimates in a small neighbourhood of the origin and starting the numerical treatment at a small distance from the origin - as noted several different cases must be considered and full details will be given elsewhere.

5 Normal samples from exponential

The construction of the normal quantile, also known as "probit" has a long and interesting history - see [18] and the references therein for details. Here we consider the construction of normal samples from exponential samples, and proceed to a detailed practical implementation. We work on the right hand region and extend the mapping to the left region by odd symmetry. The recyling ordinary differential equation in the right hand region, $v \geq 0$ is simply

$$\frac{d^2Q}{dv^2} + \frac{dQ}{dv} = Q\left(\frac{dQ}{dv}\right)^2 \tag{57}$$

with the initial conditions $Q(0) = 0$, $Q'(0) = \sqrt{\pi/2}$. This has the formal solution

$$Q(v) = \Phi^{-1}(1 - 1/2e^{-v}) \tag{58}$$

Where Φ is the normal CDF. To extract useful representations we proceed as follows. This equation may first be solved by the method of series. However, the resulting solution turns out to be an asymptotic series best used to a small number of terms in a neighbourhood of the $v = 0$. The series solution is easily found to be, using exact coefficients:

$$\begin{aligned} Q(v) =& \sqrt{\frac{\pi}{2}}v - \frac{1}{2}\sqrt{\frac{\pi}{2}}v^2 + \frac{\left(2\sqrt{\pi}+\pi^{3/2}\right)v^3}{12\sqrt{2}} - \frac{\left(\sqrt{\pi}+3\pi^{3/2}\right)v^4}{24\sqrt{2}} \\ &+ \frac{\left(4\sqrt{\pi}+50\pi^{3/2}+7\pi^{5/2}\right)v^5}{480\sqrt{2}} - \frac{\left(4\sqrt{\pi}+180\pi^{3/2}+105\pi^{5/2}\right)v^6}{2880\sqrt{2}} \\ &+ \frac{\left(8\sqrt{\pi}+1204\pi^{3/2}+1960\pi^{5/2}+127\pi^{7/2}\right)v^7}{40320\sqrt{2}} \\ &- \frac{\left(2\sqrt{\pi}+966\pi^{3/2}+3675\pi^{5/2}+889\pi^{7/2}\right)v^8}{80640\sqrt{2}} \\ &+ \frac{\left(16\sqrt{\pi}+24200\pi^{3/2}+194628\pi^{5/2}+117348\pi^{7/2}+4369\pi^{9/2}\right)v^9}{5806080\sqrt{2}} \\ &- \frac{\left(16\sqrt{\pi}+74640\pi^{3/2}+1190700\pi^{5/2}+1493520\pi^{7/2}+196605\pi^{9/2}\right)v^{10}}{58060800\sqrt{2}} \\ &+ O\left(v^{11}\right) \end{aligned} \tag{59}$$

While this expression is interesting, it does not work far enough out to be of much practical use, so a different approach is needed - if we wish to retain the use of the above expression we would need to patch in another algorithm. One could consider solving the differential equations about several points. However, an important point for modern computation is to try to avoid "IF" statements in the computer implementation. Such branches do not make use of the best features of modern GPU systems, such as the NVIDIA Tesla system [19]. The standard rational approximations all have breaks as follows in the positive quantile region $Z \geq 0, 0.5 \leq u < 1$:

- Wichura's AS241 [20]: two breaks, at $u = 0.925$ and $u = 1 - e^{-25}$.
- Moro [13]: breaks at $u = 0.92$
- Acklam Level 1[1]: breaks at $u = 0.97575$;

Wichura's model is double precision, as is the iterated Acklam (Level 2) model. The non-iterated Level 1 Acklam model is popular in financial applications and has maximum relative error less than 1.15×10^{-9}. We shall use this as a target for fast single-precision computation.

How we we avoid the break, at least for most practical computations? The first thing to point out is that the "break" at $u = 1/2$ is fictitious in practical applications. It is more sensible to work on a half region, e.g. $0.5 \leq u < 1$, an output both $Z = \Phi^{-1}(u)$ and $-Z$ for simulation purposes, i.e. always work antithetically. So we focus on the real breaks as in the list above. This break arises in standard approaches due to the fact that the standard quantile $\Phi^{-1}(u)$ has rather a split personality - it is slowly varying in the central region where u is between a half and about 0.9, and then diverges to infinity as $u \to 1_-$. This is

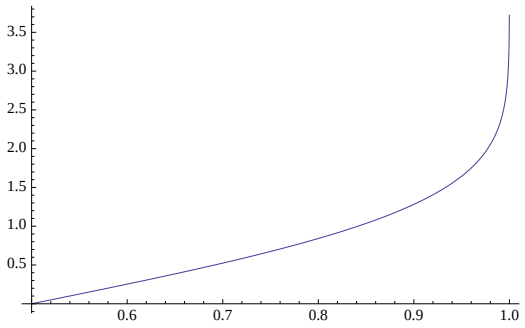

Figure 2: The normal quantile in standard coordinates

shown in Fig. 2. If we work in an exponential base the situation changes. The function $Q(v) = \Phi^{-1}(1 - 1/2e^{-v})$ is shown in Fig. 3 for the region $0 \leq v \leq 37$. This function now has a much simpler quality and we can aim to build a single

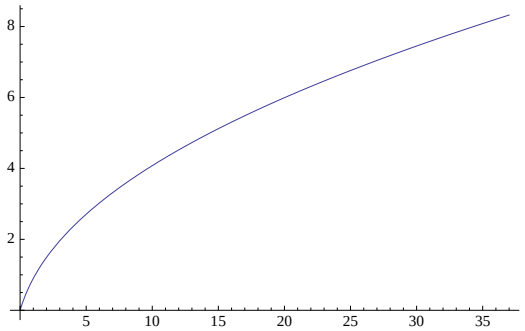

Figure 3: The normal quantile in exponential coordinates

useful rational approximation. It is then a matter of picking a target range and precision for the desired result. In Fig. 3 we have plotted the function in the range $0 \leq v \leq 37$, which is equivalent to the u-range $[0.5, 1 - e^{-37}] = [0.5, 1 - 5.55 \times 10^{-17}]$, and the Z-range $0 \leq Z < 8.3236$. So we would not expect to visit the region outside this for sample sizes less than about 10^{16}. Crudely, we are safe for samples of no bigger than a million billion. We we shall work on $v \in [0, 37]$. For precision we shall use the Acklam level one algorithm as a target. It is then a matter of taking a rational approximation of sufficient degree. This was explored using the high-precision arithmetic of Mathematica to work out the normal quantile deep into the tail, and the function `MiniMaxApproximation` to create the rational approximation. The function actually approximated was

$$\frac{Q(v)}{v} = \frac{1}{v}\Phi^{-1}(1 - 1/2e^{-v}) \tag{60}$$

and the power series for Q was used in a small neighbourhood of the origin to allow `MiniMaxApproximation` to work preserving precision near the origin, where $Q(0) = 0$. The settings employed for the computation were

- `Brake -> 10, 10`;

- `WorkingPrecision -> 20;`
- `MaxIterations -> 300;`

and a rational approximation of degree $(7, 7)$ was found with the desired accuracy. The relative error is show in Fig. 4 and is less than 1.06×10^{-9} on $0 \leq v \leq 37$.

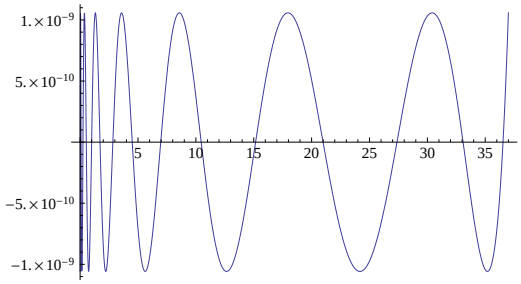

Figure 4: Precision of exponential-normal quantile on $[0, 37]$.

The resulting form for $Q(v)$ is as follows

$$Q(v) = v * P(v)/Q(v) \tag{61}$$

where P and Q are polynomials of degree 7, with nested C-forms as follows, where we produce the higher-precision output generated by Mathematica. The numerator P is

```
1.2533141359896652729 +
   v*(3.0333178251950406994 +
      v*(2.3884158540184385711 +
         v*(0.73176759583280610539 +
            v*(0.085838533424158257377 +
               v*(0.0034424140686962222423 +
                  (0.00003631387018023761224 +
                   4.3304513840364031401e-8*v)*v))))
      )
```

and the denominator Q is

```
1 + v*(2.9202373175993672857 +
      v*(2.9373357991677046357 +
         v*(1.2356513216582148689 +
            v*(0.2168237095066675527 +
               v*(0.014494272424798068406 +
                  (0.0003061726475300879397 6 +
                   1.3141263119543315917e-6*v)*v))))
      )
```

For completeness, an algorithm for normal samples based on this under standard conditions is (in the first two steps we give in brackets the better form using a reflection and scaling to simplify the first part and avoid precision reduction near unity):

- sample u in $1/2 \leq u < 1$(or, better, $0 < u < 1$);
- evaluate $v = -\log[2(1-u)]$, (then, better, $v = -\log[u]$);
- evaluate $Z = Q(v)$ with Q given by the rational approximation;
- output the antithetic pair $Z, -Z$.

If an exponential base is used we are essentially employing the last two steps.

How reasonable is it to claim that this algorithm is "essentially IF-less"? One test is to ask what would happen if we introduce a very small u into the algorithm above - what is then the margin of error if it is generates a value of $v > 37$? The precision does then deteriorate to levels above the Acklam target, but very slowly. Below $v = 50$, corresponding to u differing from an end-point by about 10^{-22}, the precision remains at better than 10^{-6}. If we double the v-range to 74, where u is $O(10^{-33})$, the precision is still better than 2×10^{-5}. So we can safely use the breakless algorithm on the basis that if a fluke sample falls outside its very wide formally-defined range the answer returned remains very good. For example, with $v = 74$ the exact result is $Z = 11.94047$ and the rational approximation yields $Z = 11.94084$. We get this stability due to the nice behaviour of the exponentially-transformed quantile, and this is then a safe algorithm for use with single-precision arithmetic, which is the particular strength of a GPU.

Of course, another very simple approach to preserving precision and avoiding an "IF" in the code is to sample the tail interval completely separately and apply a transformed quantile to that region by itself. We now turn to what that construction should be.

5.1 A supplementary tail model

If one *does* wish to penetrate the deep tail with precision preservation, the asymptotic analysis developed in the Appendix to [18] may be used - indeed, the exponential base is well adapted to the Gaussian tail. Converting coordinates, and introducing just one further group of terms into the series, we find that

$$Q(v) = \sqrt{2q(a,b)} \tag{62}$$

where

$$a = \log(v - 1/2\log(\pi)) \ , \quad b = \log(a) \tag{63}$$

and

$$\begin{aligned} q(a,b) \sim a - \frac{b}{2} + \frac{\frac{b}{4} - \frac{1}{2}}{a} + \frac{b^2 - 6b + 14}{16a^2} + \frac{2b^3 - 21b^2 + 102b - 214}{96a^3} \\ + \frac{3b^4 - 46b^3 + 348b^2 - 1488b + 2978}{384a^4} + O(a^{-5}) \end{aligned} \tag{64}$$

This again has precision better than 1.06×10^{-9}, now in the region $v \geq 37$, and indeed becomes more precise as $v \to +\infty$, as shown in Fig. 5.

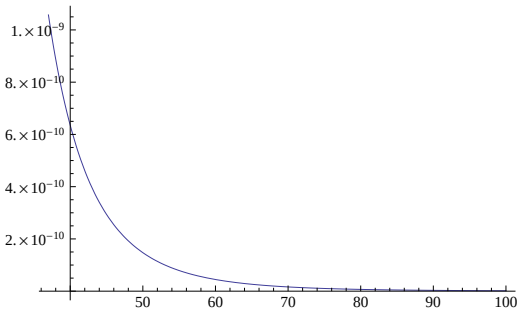

Figure 5: Precision of supplementary tail model in $v \geq 37$.

5.2 Real-world precision in C++

The following results are indicative of what happens in practice. The quantile was tested in the Bloodshed DEVC++ environment under Windows XP, using the listing in the Appendix. The output was benchmarked with all variables `double` against the internal high-precision quantile in Mathematica, and found to preserve the $O(10^{-9})$ precision. The plot of the precision of the C++ output is shown in Fig. 6.

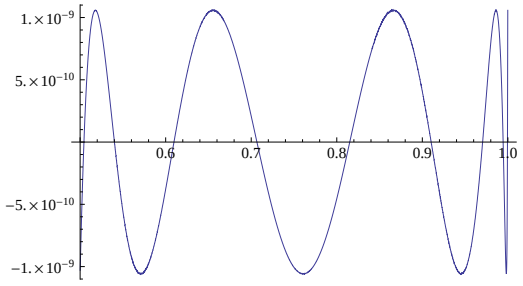

Figure 6: Precision of $(7, 7)$ "breakless" C++ model in double precision.

We do not make any claim that this algorithm is *universally* better than any other, regardless of whether one is working on a CPU or GPU. Rather, the point is that we can, by a change of variable, extend the interval over which we can cover the quantile accurately by a very large margin. The relative benefits of avoiding any IF-statement need to be assessed on a variety of computer architectures and compilers, and variations to the method above may be needed. It is certainly straightforward to generate other single-patch rational approximations with different properties. Each time we increase the degree of the numerator and denominator, keeping the interval fixed, the maximum relative error decreases by a factor of about 20. For example, a $(12, 12)$ rational approximation exists that covers the same interval $0 \leq v \leq 37$ with maximum relative error in Mathematica less than 5×10^{-16}, and in C++ with a meaningful "long double" the error remains below 7×10^{-16}. Alternatively a quite modest increase in computation allows the interval to be extended significantly. An $(8, 8)$ approximation exists with precision about 6×10^{-10} on the range $0 \leq v \leq 74$, corresponding, with reflection to $u \in [\epsilon, 1 - \epsilon]$ with $\epsilon = 3.6 \times 10^{-33}$.

The *Mathematica* notebook used to generate such schemes can be obtained on request from WS.

5.3 Benchmark results: CPU vs GPU single precision

We now turn to a more careful analysis of performance against the popular Acklam Level 1 method. The function given in Appendix A was re-written and the relevant function listed in Appendix B with some natural source-level optimizations. We call this model `ICNDfloat1`. Bearing in mind that in `float` (single-precision) mode typical of earlier GPUs, the precision of $O(10^{-9})$ is redundant, and we proposed for general single-precision use the listing in Appendix C. We call this `ICNDfloat2`. *This is the algorithm we propose for optimal GPU normal simulation based on quantiles.* If implemented in double precision the maximum relative error is less than 4×10^{-7}. In practice in *float* form it gave results slightly better than the *float* form of the Acklam result, particularly near the branch point.

First, consider why any improvement at all might be expected. On a GPU it is typically the case that a number of threads are executed at the same time. However, the GPU architecture is such that the timing of such a multi-threaded computation is influenced by the slowest outcome of any of the branches that are executed. In the Acklam model there is a fast rational approximation with no special function calls in the central region. In the tail there is the operation of taking a log followed by a square root. In the Moro model a log(log()) operation is carried out in the tail region. AS241 also uses composite special function calls. On a CPU the timing of the algorithm benefits from the fast central algorithm and the tail algorithm slows the routine down only on (for the Acklam case) 4.85% of calls. This is highly efficient on a CPU architecture that processes each calculation separately. A simple timing was done using the Bloodshed DEV C++ compiler on an Intel 2.8GHz machine. In each case the simple internal `rnd` function, normalized to return values of U in the unit interval, was run a billion times without the normal quantile call and then with the normal quantiles we are considering[2]. The timings for calling the quantile obtained by subtracting the two results *on the CPU* were as follows (results in seconds):

- Acklam Level 1; CPU: $59s$
- ICNDfloat1; CPU: $89s$:
- ICNDfloat2; CPU: $82s$

In each case the overhead of calling `rnd` was about $15s$. These results demonstrate clearly the efficiency of the Acklam approach in a traditional architecture.

For a proper GPU analysis the code was ported initially to an 8400GS GPU and re-run in the same way. For a fair comparison the Acklam algorithm was optimized. Timings for the quantile call were as follows

The benefit of working in branchless form is now clear. The improvement, though modest, can make a difference, especially if one is solving an SDE via many calls to a normal random variable prior to evaluating a payoff.

[2]The rnd() function is of course completely unsuitable for real-world use, but given that we only need a method for sampling the various regions and subtract the overhead, its use here is fine.

Algorithm	Timing $t[s]$
Acklam	5.04
ICNDfloat1	4.89
ICNDfloat2	4.64

Table 1: Single precision timings for normal quantile on GPU

In the full philosophy of this paper, one would of course use an exponential base for many different computations and distributions and possibly pre-store a large set of exponential samples created by efficient methods. The overhead of converting to normal is then the evaluation of a simple rational function and the performance benefits are magnified many times over those given in Table 1.

5.4 High precision work

One can also consider working to double precision on a modern TESLA GPU. The first matter to establish is the quality of standard methods. There are two well-known candidates. These are

1. Wichura's AS241;
2. The *refined Acklam* method, where the level one approximation is fed once through a Newton-Raphson-Halley method.

How do we to a quality check on such high precision methods? We will use the *Mathematica* function `InverseErf` as our benchmark. However, this will not be done blindly on the assumption that it is necessarily correct. The quantile based on this has been independently assessed against the known exact solution for the Gaussian quantile developed by Steinbrecher and Shaw [18] that is known in series form. The formula for this in a computation-suitable representation is also available at `http://en.wikipedia.org/wiki/Probit` and as a series has been coded up both in *Mathematica* and quadruple-precision FORTRAN based on the Absoft compiler. Based on these three implementations various cross-verifications have been carried out. For example, the quad-precision FORTRAN code that agrees with *Mathematica's* internal `InverseErf` to a precision of better than 10^{-29} on the interval $[e, 1-e]$, with $e = 0.0007$. Near the centre of the unit interval the truncated series written in *Mathematica* agrees with `InverseErf` to much better than quad precision. So we have considerable confidence that our benchmark is precise enough for any double-precision evaluation.

A precision test of AS241 was carried out in previous work [16] and the relative precision of a *Mathematica* representation of AS241 is shown in Fig. 7 This confirms the double-precision quality of the algorithm. We obtained less satisfactory results with the refined Acklam scheme. While the relative error is typically of order 10^{-15} away from the middle or tail, there is a loss of precision in the middle. The Newton-Raphson-Halley refinement was done first exclusively in *Mathematica*. The precision near the middle is shown in Fig. 8. Similar loss of precision was found in the implementation by J. Lea in C/C++ of the refined method, using the Cody formula for the CDF. Based on these observations we are completely satisfied that this algorithm is machine

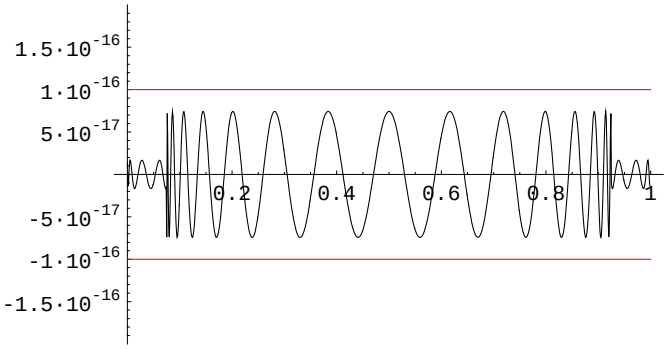

Figure 7: Precision of AS241

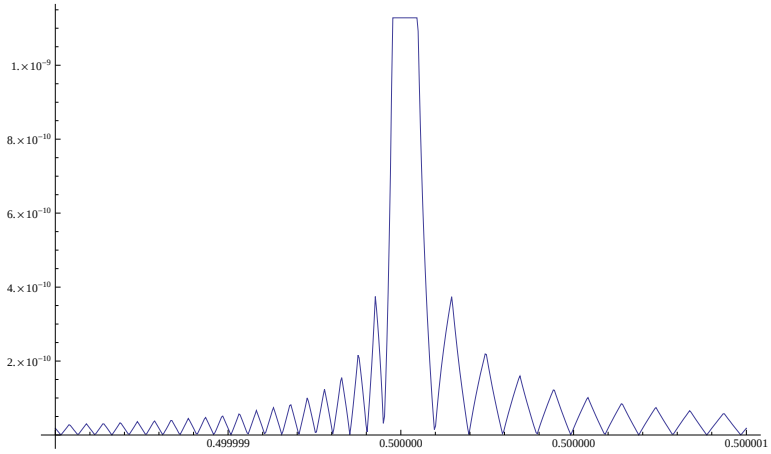

Figure 8: Precision of refined Acklam - centre region

precision, though we cannot of course rule out some problem caused by our own implementation.

We now turn attention to real-world precision and performance in C/C++ using a `double` type specification in AS241 and our own proposal. The C++ implementation for AS241 is that supplied by John Burkardt at

`http://people.sc.fsu.edu/~burkardt/cpp_src/asa241/asa241.html`

For completeness we also considered the coding of the refined Acklam algorithm supplied by Jeremy Lea at

`http://home.online.no/~pjacklam/notes/invnorm/impl/lea/lea.c`

Our own suggestion for double precision work is listed in Appendix D, and is given there in a form suitable for use as a CUDA kernel. A CPU version for C/C++ is easily extracted with a little editing. The theoretical precision of this algorithm when evaluated in arbitrary precision in *Mathematica* is $O(10^{-15})$ on the interval $[\epsilon, 1-\epsilon]$, with $\epsilon < 10^{-32}$, and so is good for any practical size

Monte Carlo[3]. In practice the precision in a `double` implementation in C/C++ is similar to AS241. Even the *CPU* timings are revealing and are as follows, for half a billion samples on $[0, 0.5]$.

Algorithm	Timing $t[s]$
Acklam (refined)-Lea	179
AS241-Burkardt	104
GPU DP model	82

Table 2: Double precision timings for normal quantile algorithms on *CPU*

Due to the elimination of branches and the avoidance of calls to a `sqrt(log())` operation we expect the GPU advantage to be better still. The refined Acklam method is slow probably due to expensive calls to the error function for all arguments - the GPU method is now more than twice as fast even when evaluated on a CPU, notwithstanding our precision issues. AS241 stands up well as a high precision benchmark but it is now possible to proceed faster. We re-iterate that the single precision form of the Acklam method remains optimal for *float*-class calculations on a CPU, but is also outrun on a GPU by an optimized algorithm. Further optimizations may of course be possible - the codes presented here in Appendices C and D are our current optimal forms and may be subject to further improvement as regards speed and precision. We will also explore OpenCL implementations.

The comparisons on a modern GPU are very interesting. We have completed a comparison of four algorithms on three GPUS. The four algorithms are:

- Acklam's one-pass method, but coded in double precision (DP)
- The Acklam-Lea DP method
- The breakless GPU approach (code exactly as in Appendix D)
- AS241 (Burkardt code as of early 2009)

The three GPUs consider are

- Quadro 4800
- GTX 285
- GTX 480

The host machine was in each case was a Mac Pro (2008) model running OS X 10.6.3 for the Quadro and 285 tests, and running Windows XP32 for the 285 and 480 tests. The Windows and OS X 285 numbers are almost identical so only one set is reported. The timings in ms for the test program[4]

These initial results suggest that if one is working in double precision, the breakless method is even faster than the one-step Acklam method, with rather more precision.

[3]The value of ϵ is now so small that we could in fact add a break and a tail model without compromising GPU efficiency, as the probability of probing the tails is now so small.

[4]In all cases compiled with the CUDA 1.3 compute architecture so the numbers may be conservative for the GTX 480, a 2.0 CUDA device.

Algorithm/	Timing $t[ms]$	Timing $t[ms]$	Timing $t[ms]$
/GPU	Quadro 4800	GTX 285	GTX 480
Acklam single as double	3999	2588	1106
Acklam double	9405	6064	2735
BreaklessGPU	3499	2240	1046
AS241	5051	3237	1476

Such forms of the quantile have been exploited by Joshi [10] in the GPU form of an Asian option model. We also wish to point out that the almost breakless form here might not be the optimal thing to do. One can consider putting the break in a less extreme location and having shorter rational approximations in two regions.[5] We hope to explore the possibilities more fully in future studies.

6 Conclusions

In the post-credit-crunch environment, risk simulations depend critically on having a realistic (fat-tailed) model of asset returns. The methods developed here allow traditional Gaussian samples to be converted to other distributions via the application of the solution differential equation to the samples. The differential equation is the recycling ODE for transforming samples from a density f_1 to a density f_2, and is

$$\frac{d^2Q(v)}{dv^2} + H_1(v)\frac{dQ(v)}{dv} = H_2(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 ,$$

where

$$H_i = -\frac{d}{dx}\log[f_i(x)]$$

We have given an explicit example for the Student t case, where a power series emerges coupled to a tail model. Other more complicated distributions with an explicit density may be handled similarly or numerically, and other base distributions may be treated. In particular we can use changes of variable to construct "essentially IF-less" algorithms for objects like the normal quantile. The efficiency of such algorithms in GPU computation is of interest, and the methods introduced here can be considered for other target distributions. In contrast to the normal case, where there are no parameters beyond the translation and scale, we must first solve the RODE with the relevant parameters and then develop a suitable fast approximation.

These methods also simplify the use of a Gaussian or T-copula, since the two steps of mapping to the unit hypercube and back to the marginals may be folded together into one operation, where the solution to the RODE is applied directly in one step.

Of course, the methods developed here rely on the ability to compute the logarithmic derivatives of the target and base densities. Where the target density is not known explicitly, but whose characteristic function is known, other methods must be used. Investigations of the resulting integro-differential equations will be reported elsewhere.

[5] M. Giles, private communication

We have reported a new formula for the normal quantile and demonstrated a modest performance benefit on a GPU architecture by working in a branchless form for single precision work. Initial CPU tests on Double precision variations suggest more significant performance enhancements.

Acknowledgments

WS wishes to thank I. Buckley, W. Gilchrist, P. Jäckel, D. Scott, G. Steinbrecher and Y. Xiong for discussions on various aspects of quantile theory. Presentations by C. Albanese, M.Giles and G. Ziegler and the NAG team at the King's College London workshop on GPU computing [2] stimulated the development of the essentially "IF-less" normal quantile. A. Munir provided assistance in producing further Windows binaries for the 1.3 CUDA architecture.

References

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

[2] C. Albanese, G. Ziegler, D. Sayers, M. Giles, Presentations at Nov 2007 King's College London Workshop on GPU computing in finance, Workshop home page at `http://www.level3finance.com/gpuworkshop.html`.

[3] M. Abramowitz, I.A. Stegun, Handbook of Mathematical Functions, Dover, 1975.

[4] R.A. Bagnold, The Physics of Blown Sands and Desert Dunes. Methuen, London, 1941.

[5] O.E. Barndorff-Nielsen Exponentially Decreasing Distributions for the Logarithm of a particle size. Proceedings of the Royal Society (London), Series A, 353, 401-419, 1977.

[6] E. Eberlein, E. and U. Keller, Hyperbolic Distribution in Finance. *Bernoulli*, Vol. 1, No. 3 (Sept, 1995), pp. 281-299, 1995.

[7] K. Fergusson, E. Platen, On the Distributional Characterization of daily Log-returns of a World Stock Index, *Applied Mathematical Finance*, 13 (1), 19-38, March 2006.

[8] Warren Gilchrist, *Statistical modelling with quantile functions*, CRC Press Inc, 2000.

[9] G.W. Hill, A.W. Davis, Generalized Asymptotic Expansions of Cornish-Fisher Type, *Annals of Mathematical Statistics*, 39, 4, 1264-1273, 1968.

[10] M. Joshi Graphical Asian options, to appear in *Wilmott* Journal.

`http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1473563`

[11] D.B. Madan, E. Seneta, The variance gamma (V.G.) model for share market returns, Journal of Business, 63, pp. 511 - 524, 1990.

[12] T. Mikosch, Copulas: tales and facts. Extremes, 9, pp 3-20, 2006.

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

[14] Wikipedia, entry on "Quantile function", `http://en.wikipedia.org/wiki/Quantile_function`

[15] W. T. Shaw, Sampling Student's T distribution - use of the inverse cumulative distribution function. *Journal of Computational Finance*, Vol. 9, No. 4, 2006.

[16] W. T. Shaw, Refinement of the Normal Quantile, Simple improvements to the Beasley-Springer-Moro method of simulating the Normal Distribution, and a comparison with Acklam's method and Wichuras AS241

`http://www.mth.kcl.ac.uk/~shaww/web_page/papers/NormalQuantile1.nb`.

[17] W. T. Shaw, I.R.C. Buckley, The alchemy of probability distributions: beyond Gram-Charlier expansions, and a skew-kurtotic-normal distribution from a rank transmutation map. Presented at the First IMA Conference on Computational Finance, March 2007. `http://arxiv.org/abs/0901.0434v1`

[18] G. Steinbrecher and W.T. Shaw, Quantile Mechanics. *European Journal of Applied Mathematics*, 19(2), pp 87-112, 2008.

[19] Nvidia TESLA computing solutions, `http://www.nvidia.com/object/tesla_computing_solutions.html`

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

[21] Y. Xiong Sampling hyperbolic distribution by quantile function, *MSc thesis*, King's College London, September 2008.

Appendix A: C++ listing for precision testing

This is the C++ listing for the test program for the "breakless" positive normal quantile used to generate the output in Fig. 6, when compared with the internal high-precision quantile in Mathematica.

```
//breaklessquantile.cpp
#include <cmath>
#include <iostream>
#include <fstream>
using namespace std;

double BreaklessQuantile(double u)
{
```

```
double v=-log(2*(1-u));
double P = 1.2533141359896652729 +
   v*(3.0333178251950406994 +
      v*(2.3884158540184385711 +
         v*(0.73176759583280610539 +
            v*(0.085838533424158257377 +
               v*(0.0034424140686962222423 +
                  (0.00003631387018023761224 +
                   4.3304513840364031401e-8*v)*v)))));

double Q=1+v*(2.9202373175993672857 +
      v*(2.9373357991677046357 +
         v*(1.2356513216582148689 +
            v*(0.2168237095066675527 +
               v*(0.014494272424798068406 +
                  (0.00030617264753008793976 +
                   1.3141263119543315917e-6*v)*v)))));
return v*P/Q;
};

// The function is all above - that below is the simple test program.

int main()
{
    double q;
    double quantile;

    char name[5];
    int k,m;

    cout << "Outputting test values of breakless quantiles " << "\n";
    ofstream out("breaklessquantiles.txt");
    for (k=5000; k<=9999; k++)
    {q = k/10000.;
    quantile = BreaklessQuantile(q);
    out.precision(12);
    out << q <<","<< quantile << "\n";}
    cout << "Output written to breaklessquantiles.txt \n";
    cout << "Hit any key to quit \n";
    cin >> name;
    return(0);
}
```

Appendix B: ICNDfloat1 listing

Here is full quantile form of the function in Appendix A in a form suitable for GPU work under CUDA.

```
#include <cmath>
```

```
using namespace std;
#define BQP(v) (P1+v*(P2+v*(P3+v*(P4+v*(P5+v*(P6+(P7+P8*v)*v))))))
#define BQQ(v) (Q1+v*(Q2+v*(Q3+v*(Q4+v*(Q5+v*(Q6+(Q7+Q8*v)*v))))))
float ICNDfloat1(float v)
{
    const float P1 = 1.2533141359896652729;
    const float P2 = 3.0333178251950406994;
    const float P3 = 2.3884158540184385711;
    const float P4 = 0.73176759583280610539;
    const float P5 = 0.085838533424158257377;
    const float P6 = 0.0034424140686962222423;
    const float P7 = 0.00003631387081802376122 4;
    const float P8 = 4.3304513840364031401e-8;
    const float Q1 = 1.0;
    const float Q2 = 2.9202373175993672857;
    const float Q3 = 2.9373357991677046357;
    const float Q4 = 1.2356513216582148689;
    const float Q5 = 0.2168237095066675527;
    const float Q6 = 0.014494272424798068406;
    const float Q7 = 0.00030617264753008793976;
    const float Q8 = 1.3141263119543315917e-6;
    float z;
    int sgn;
    sgn = (v >= 0.5);
    sgn = sgn - !sgn;
    z = -logf(1.0 - (sgn * ((2.0 * v) - 1.0)));
    return sgn * z * BQP(z) / BQQ(z);
}
```

Appendix C: ICNDfloat2 listing

Here is full quantile form of the optimized single precision algorithm in a form suitable for GPU work under CUDA.

```
#include <cmath>
using namespace std;

#define CQP(v) (P1+v*(P2+v*(P3+v*(P4+(P5+P6*v)*v))))
#define CQQ(v) (Q1+v*(Q2+v*(Q3+v*(Q4+(Q5+Q6*v)*v))))
float ICNDfloat2(float v)
{
    const float P1 = 1.2533136835212087879;
    const float P2 = 1.9797154223229267471;
    const float P3 = 0.80002295072483916762;
    const float P4 = 0.087403248265958578062;
    const float P5 = 0.0020751409553756572917;
    const float P6 = 4.744820732427972462e-6;
    const float Q1 = 1.0;
    const float Q2 = 2.0795584360534589311;
```

```
    const float Q3 = 1.2499328117341603014;
    const float Q4 = 0.23668431621373705623;
    const float Q5 = 0.0120098270559197768;
    const float Q6 = 0.00010590620919921025259;
    float z;
    int sgn;

    sgn = (v >= 0.5);
    sgn = sgn - !sgn;
    z = -logf(1.0 - (sgn * ((2.0 * v) - 1.0)));
    return sgn * z * CQP(z) / CQQ(z);
}
```

Appendix D: Double branchless quantile

Here is optimized double precision branchless algorithm in a form suitable for CUDA kernel use.

```
extern "C" __global__ void EDPBreaklessInvCNDgpu(FP * aa, FP * bb, int N)

{
    const double P1 = 1.2533141373154989811;
    const double P2 = 5.5870183514814983104;
    const double P3 = 9.9373788223105148469;
    const double P4 = 9.11745910783758368;
    const double P5 = 4.6865666928347513004;
    const double P6 = 1.3841649695441184484;
    const double P7 = 0.23434950424605615377;
    const double P8 = 0.022306824510199724768;
    const double P9 = 0.0011538603964070818722;
    const double P10 = 0.00003079662069141156756 3;
    const double P11 = 3.9115723028719510263e-7;
    const double P12 = 2.0589573468131996933e-9;
    const double P13 = 3.3944224725087481454e-12;
    const double P14 = 7.3936480912071325978e-16;
    const double Q1 = 1.0000000000000000000;
    const double Q2 = 4.9577956835689939051;
    const double Q3 = 9.9793129245112074476;
    const double Q4 = 10.574454910639356539;
    const double Q5 = 6.4247521669505779535;
    const double Q6 = 2.3008904864351121026;
    const double Q7 = 0.48545999687461771635;
    const double Q8 = 0.059283082737079006352;
    const double Q9 = 0.004061850620607899582 1;
    const double Q10 = 0.00014919732843986856251;
    const double Q11 = 2.747706139204994706 6e-6;
    const double Q12 = 2.2815008011613816939e-8;
    const double Q13 = 7.0445790305953963457e-11;
    const double Q14 = 5.1535907808963289678e-14;
```

```
    double v,z,vv;
    int sgn;
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    v = aa[idx];
    sgn = (v >= 0.5);
    sgn = sgn - !sgn;
    if (sgn == -1) {vv = v;} else {vv = 1.0-v;}
    z = -log(2.0*vv);
    double num =(P1+z*(P2+z*(P3+z*(P4+z*(P5+z*(P6+z*(P7+z*(P8+z*(P9+z*(P10+z*(P11+z*
    (P12+z*(P13+P14*z))))))))))));
    double den =(Q1+z*(Q2+z*(Q3+z*(Q4+z*(Q5+z*(Q6+z*(Q7+z*(Q8+z*(Q9+z*(Q10+z*(Q11+z*
    (Q12+z*(Q13+Q14*z))))))))))));

bb[idx] = sgn*z*num/den;
}
```