Euro. Jnl of Applied Mathematics (2014), *vol.* 25, *pp.* 177–212. © Cambridge University Press 2014
doi:10.1017/S0956792513000417

Quantile mechanics II: changes of variables in Monte Carlo methods and GPU-optimised normal quantiles

WILLIAM T. SHAW[1], THOMAS LUU[2] and NICK BRICKMAN[3]

[1]*Department of Mathematics, University College London, Gower Street, London WC1E 6BT, UK*
email: w.shaw@ucl.ac.uk
[2]*Department of Mathematics, University College London, Gower Street, London WC1E 6BT, UK*
email: t.luu@ucl.ac.uk
[3]*20 Racton Road London SW6 1LP, UK*
email: nick@taylorbrickman.co.uk

(*Received 26 August 2010; revised 1 December 2013; accepted 3 December 2013; first published online 24 January 2014*)

With financial modelling requiring a better understanding of model risk, it is helpful to be able to vary assumptions about underlying probability distributions in an efficient manner, preferably without the noise induced by resampling distributions managed by Monte Carlo methods. This paper presents differential equations and solution methods for the functions of the form $Q(x) = F^{-1}(G(x))$, 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 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 sample space. The avoidance of branching statements is of use in optimal graphics processing unit (GPU) computations as it avoids the effect of *branch divergence*. We give a branch-free normal quantile that offers performance improvements in a GPU environment while retaining the best precision characteristics of well-known methods. We also offer models with low probability branch divergence. Comparisons of new and existing forms are made on NVIDIA GeForce GTX Titan and Tesla C2050 GPUs. We argue that in both single- and double-precisions, the change-of-variables approach offers the most GPU-optimal Gaussian quantile yet, working faster than the CUDA 5.5 built-in function.

Key words: Monte Carlo; Student; Computational finance; Quantile mechanics; Normal, Gaussian; Quantile; GPU; CUDA; Acklam; Wichura; Moro; Inverse CDF; Probit

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 the distribution. If $F(x)$ is the cumulative

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

distribution function (CDF), then the quantile $w(u)$ is the solution of the equation

$$F(w(u)) = u. \tag{1.1}$$

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 characterised by $F(x)$ is

$$X = w(U)\ . \tag{1.2}$$

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 [9], Mikosch stated that *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. On the other hand, samples from the uniform distribution are readily generated, and in the computer science domain it remains a topic of continuing study to develop better uniform random number generators (RNGs).

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 paper will address this question in the affirmative. In principle, the answer is trivial: Given a sample X from a distribution with CDF $G(x)$, we first work out $G(X)$ which is uniform. Then we can apply the quantile function $F^{-1}(x)$ associated with a target distribution with CDF F and form $F^{-1}(G(x))$ as a sample from that target distribution. In general, F, G and their inverses can be rather awkward special functions (see e.g. [14]), so a direct route to the object $Q(x) = F^{-1}(G(x))$ 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 [5]. 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 the 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]. We will give a brief review of that approach, and shall use this insight to build a new representation of the Student t quantile in terms of a series expansion of a Gaussian. This neatly tidies up the well-known expansions already available for the Student distribution as the degrees of freedom become large, and we obtain a more compact power series in the normal random variable.

In view of the importance of Gaussian sampling, we will also explore how to make normal random variables by the change-of-variables approach, and explore the use of the exponential and even Student distributions as intermediaries. In these simple cases we can write down the objects $F^{-1}(G(x))$ very explicitly and the issue becomes one of the efficient approximation and computation of such objects. We will show that this offers a useful performance benefit in a graphics processing unit (GPU) environment, where branching algorithms may be subject to significant performance penalties. Our change-of-variables

[1] This rather clear observation was first made to W. Shaw by Peter Jaeckel.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

approach will allow costly branching to be avoided or reduced to a low probability and we will demonstrate the benefits in the CUDA environment for programming NVIDIA GPUs [12]. This part of the paper of necessity involves ideas from computer science as much as applied mathematics.

The plan of this work is as follows. In Section 2 we review the differential approach to quantile functions and introduce a new differential equation that characterises the mapping from samples of one probability distribution to another. We give specific examples for 'recycling' samples of exponential and Gaussian random variables. In Section 3 we focus on Gaussian backgrounds and review the transformation from Gaussian random variables to those from Student t distribution. We will show how this recasts the traditional Cornish–Fisher expansions into power series solutions of non-linear ordinary differential equations (ODEs), and give a brief introduction to how this might be applied.

In the second part of the paper we focus on the issue of building quantile approximations for the normal distribution based on the recycling approach. In Section 4 we set out some ideas needed for the development of approximations to quantile functions, however constructed, for any distribution. Section 5 implements the change of variables approach based on an exponential distribution for the Gaussian quantile. We also introduce the novel idea of generating Gaussian samples by first sampling a distribution inspired by the Student t distribution with two degrees of freedom. The combination of these two approaches results in a new formula for the Gaussian quantile in double-precision that is substantially faster on a GPU than any method known to us while preserving full double-precision on a range more than enough for Monte Carlo work. Section 6 concludes, and relevant codes are supplied in the appendices.

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 (1.1) to obtain

$$f(w(u))\frac{dw(u)}{du} = 1, \tag{2.1}$$

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

$$f(w(u))\frac{d^2w(u)}{du^2} + f'(w(u))\left(\frac{dw(u)}{du}\right)^2 = 0. \tag{2.2}$$

This may be reorganised as

$$\frac{d^2w(u)}{du^2} = H(w(u))\left(\frac{dw(u)}{du}\right)^2, \tag{2.3}$$

where

$$H(w) = -\frac{d}{dw}\log\{f(w)\}\,. \tag{2.4}$$

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

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 (2.3). We let $v = q(u)$, and regard w as a function of v by writing $w(u) = Q(v)$. Elementary application of the chain rule and some algebra gives us the following:

$$\frac{d^2Q(v)}{dv^2} + \frac{q''(u)}{[q'(u)]^2}\frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 . \tag{2.5}$$

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

$$\frac{d^2q(u)}{du^2} = \hat{H}(q(u))\left(\frac{dq(u)}{du}\right)^2 , \tag{2.6}$$

where

$$\hat{H}(w) = -\frac{d}{dw}\log\{\hat{f}(w)\} \tag{2.7}$$

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

$$\frac{d^2Q(v)}{dv^2} + \hat{H}(q(u))\frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 , \tag{2.8}$$

and bearing in mind that $v = q(u)$, we arrive at the 'Recycling ODE':

$$\frac{d^2Q(v)}{dv^2} + \hat{H}(v)\frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 . \tag{2.9}$$

We now turn to two particularly interesting cases, rather inspired by Mikosch's suggestions [9].

2.2 The Recycling ODE for a Gaussian background

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

$$\hat{f}(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}, \tag{2.10}$$

$$\log\hat{f}(x) = -1/2\log(2\pi) - x^2/2, \tag{2.11}$$

$$\frac{d}{dx}\log\hat{f}(x) = -x, \tag{2.12}$$

$$\hat{H}(v) = v, \tag{2.13}$$

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

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

$$\frac{d^2Q(v)}{dv^2} + v\frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 . \tag{2.14}$$

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:

$$\hat{f}(x) = e^{-x}, \quad \log\hat{f}(x) = -x, \quad \frac{d}{dx}\log\hat{f}(x) = -1, \quad \hat{H}(v) = 1, \tag{2.15}$$

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

$$\frac{d^2Q(v)}{dv^2} + \frac{dQ(v)}{dv} = H(Q(v))\left(\frac{dQ(v)}{dv}\right)^2 . \tag{2.16}$$

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 of the form

$$Q'' + v\,Q' = H(Q)\left(Q'\right)^2, \tag{3.1}$$

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 the Cornish–Fisher-type expansions. The generalised Cornish–Fisher expansions have been considered in a notable paper by Hill and Davis [8], 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.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

(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 density and associated H-function for the Student case can be written down as

$$f_{t_n} = \frac{1}{\sqrt{n\pi}} \frac{\Gamma\left[(n+1)/2\right]}{\Gamma\left[n/2\right]} \left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}} \quad , \quad H_{t_n}(Q) = \left(1 + \frac{1}{n}\right) \frac{Q}{1 + Q^2/n}, \tag{3.2}$$

where n is the degrees of freedom (not necessarily an integer), and the Recycling ODE can be written in the form

$$\left(1 + \frac{Q^2}{n}\right)\left(Q'' + v\,Q'\right) = \left(1 + \frac{1}{n}\right) Q\left(Q'\right)^2 \ . \tag{3.3}$$

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

$$Q'' + v\,Q' = Q\left(Q'\right)^2 \tag{3.4}$$

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 [14], where it was noted that the known Cornish–Fisher expansion always goes wrong in the tails at some point.

We also need to consider boundary conditions. The derivative of any ordinary quantile function at a point is the reciprocal of the probability density function (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 for the Gaussian and Student distributions, then we have

$$\begin{aligned} z(1/2) &= 0, \\ z'(1/2) &= \sqrt{2\pi}, \\ t(1/2) &= 0, \\ t'(1/2) &= \sqrt{n\pi}\frac{\Gamma\left[n/2\right]}{\Gamma\left[(n+1)/2\right]} \ . \end{aligned} \tag{3.5}$$

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

$$\begin{aligned} Q(0) &= 0, \\ Q'(0) &= \gamma \equiv \sqrt{\frac{n}{2}}\frac{\Gamma\left[n/2\right]}{\Gamma\left[(n+1)/2\right]}, \end{aligned} \tag{3.6}$$

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

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

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

$$Q(v) \sim \sum_{k=0}^{\infty} c_k v^{2k+1}, \tag{3.7}$$

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

$$\begin{aligned} c_1 &= \frac{(n+1)\gamma^3 - n\gamma}{6n}, \\ c_2 &= \frac{\left(7n^2+8n+1\right)\gamma^5 + \left(-10n^2-10n\right)\gamma^3 + 3n^2\gamma}{120n^2}\ . \end{aligned} \tag{3.8}$$

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

$$\begin{aligned} (2i+3)(2i+2)c_{i+1} = &-(2i+1)c_i \\ &+ \sum_{l=0}^{i}\sum_{m=0}^{i-l} a_{lm}(n) c_{i-l-m}c_l c_m \\ &- \frac{\theta(i)}{n}\sum_{l=0}^{i-1}\sum_{m=0}^{i-1-l}(2m+1)c_{i-1-l-m}c_l c_m, \end{aligned} \tag{3.9}$$

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

$$a_{lm}(n) = \left(1+\frac{1}{n}\right)(2l+1)(2m+1) - \frac{2m}{n}(2m+1)\ . \tag{3.10}$$

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

$$Q\left(Q'' + v\,Q'\right) = (n+1)\left(Q'\right)^2. \tag{3.11}$$

Following some experimentation, we make the change of variables

$$P(v) = \frac{1}{Q(v)^n} \tag{3.12}$$

and this reduces the ODE to

$$P''(v) + v\,P'(v) = 0\ . \tag{3.13}$$

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

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

$$P(v) \propto \operatorname{erfc}\left(\frac{v}{\sqrt{2}}\right) \tag{3.14}$$

and we deduce that for some constant c,

$$Q(v) \sim c\left[\frac{1}{2}\operatorname{erfc}\left(\frac{v}{\sqrt{2}}\right)\right]^{-1/n}. \tag{3.15}$$

We see that the solution has emerged naturally as

$$Q(v) \sim c\,[1-\Phi(v)]^{-1/n}, \tag{3.16}$$

where Φ is the Gaussian CDF. The asymptotic differential equation is scale invariant, so we have to determine c 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 [16, Equations (60–62)] the tail behaviour of the Student CDF was determined, and we can deduce that

$$c = \sqrt{n}\left[n\sqrt{\pi}\frac{\Gamma(n/2)}{\Gamma((n+1)/2)}\right]^{-1/n}. \tag{3.17}$$

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 the 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 in [2],

$$\begin{aligned} t =& z + \frac{z^3+z}{4n} + \frac{5z^5+16z^3+3z}{96n^2} + \frac{3z^7+19z^5+17z^3-15z}{384n^3} \\ &+ \frac{79z^9+776z^7+1482z^5-1920z^3-945z}{92160n^4} + \ldots \end{aligned} \tag{3.18}$$

An equation of the true Cornish–Fisher-type expansion (cf identity 26.2.49 in [2]) 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 (3.18) 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

$$\gamma = 1 + \frac{1}{4n} + \frac{1}{32}\left(\frac{1}{n}\right)^2 - \frac{5}{128}\left(\frac{1}{n}\right)^3 - \frac{21\left(\frac{1}{n}\right)^4}{2048} + \frac{399\left(\frac{1}{n}\right)^5}{8192} + O\left(\left(\frac{1}{n}\right)^6\right), \tag{3.19}$$

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

which sums up all the z-terms. Similarly,

$$c_2 = \frac{1}{4n} + \frac{1}{6}\left(\frac{1}{n}\right)^2 + \frac{17}{384}\left(\frac{1}{n}\right)^3 - \frac{1}{48}\left(\frac{1}{n}\right)^4 - \frac{17\left(\frac{1}{n}\right)^5}{8192} + O\left(\left(\frac{1}{n}\right)^6\right) \tag{3.20}$$

and so on. So the series solution of the differential equation constitutes a re-summation of the known asymptotic series where the coefficient of each power of z is computed exactly. This means that we can use the series without assuming that n is large, as it is a series valid in some domain for z, the existence of which does not depend on n being large.

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 is the Student CDF. The exact formula for the Student CDF for all real n is given in terms of inverse beta functions by Shaw [14], and there are known simpler forms for $n = 1, 2, 4$. These are also given in [14], and are now available on the *Wikipedia* page on quantile functions [20]. The case $n = 4$ is an interesting one as it is known exactly in the boundary case where kurtosis is infinite, and there is some evidence from work by Fergusson and Platen [4] 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 $|z| < 4$. Given that z is a standard Gaussian, this is within the range of ± 4 standard deviations for the underlying Gaussian.

For this case we find that

$$\gamma = \frac{4}{3}\sqrt{\frac{2}{\pi}} \sim 1.06384608107048714 \tag{3.21}$$

and the full C language form for the central series is, with $y = z^2$,

```
t = z*(1.06384608107048714 +
       y*(0.0735313753642658509 +
          y*(0.00408737916150927847 +
             y*(0.000157376276663230562 +
                y*(4.3193982414036350 9e-6 +
                   y*(9.5688146463922727 8e-8 +
                      y*(2.0925688180361444 6e-9 +
                         y*(3.8796293820909335 2e-11 +
                            y*(2.7232608454191567 1e-13 +
                               (2.9052893016237332 8e-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

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

tail (the negative tail being treated by symmetry)

$$w = (1 - \Phi(z))\, n\sqrt{\pi}\frac{\Gamma(n/2)}{\Gamma((n+1)/2)},$$
$$t = \sqrt{n}w^{-1/n}\left(1 - \frac{n+1}{2(n+2)}w^{2/n}\right), \tag{3.22}$$

and for the case $n = 4$:

$$w = (1 - \Phi(z))\frac{16}{3},$$
$$t = 2w^{-1/4}\left(1 - \frac{5}{12}w^{1/2}\right). \tag{3.23}$$

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 specialised, also allows the appraisal of direct numerical schemes. The direct numerical solution of the Recycling ODE can be done using standard methods. Within *Mathematica*, the use of `NDSolve` with high precision and accuracy goals, explicit Runge–Kutta and sixth-order differences leads to a 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 these can be made the basis of a further, for example, rational approximation if speed is an issue. Such a numerical scheme will be exploited for the Gaussian quantile in Section 5.

There is clearly much more scope for transforming between Student distributions of different n and Gaussians, and later in this paper we will provide an example of transforming a t_2 to a Gaussian. Given that there are closed forms for the quantiles for t_1, t_2 and t_4, these might serve as useful intermediate distributions for other applications as well, but this will be explored elsewhere.

4 Issues in building better quantile approximations

In this section we consider the issues involved in building more accurate and/or faster quantiles for live use on computer simulations. These will be important for *any* distribution, but the particular focus here will be on finding new forms of quantiles for the normal distribution. The construction of the normal quantile, also known as 'probit', has a long and interesting history – see [18] and the references therein for details. This is important, given its base case use in general modelling applications, and in particular, given its massively repeated use in the solution of stochastic differential equations. The issues we consider are as follows:

(1) The target precision.

(2) The domain over which the target is to be rigorously achieved.

(3) The choice of intermediate probability distribution to be used in the recycling scheme.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

(4) The algorithm's performance on typical central processing unit (CPU) and GPU architectures.

The last point applies to the particular approach developed here, but the other points are relevant to any approach. In the following sections we explore these issues and the associated choices. This gives some basic parameters relevant to our general scheme of implementing 'Quantile Mechanics'. In the past, diverse choices have been made for precision and domain, and we think it appropriate to better characterise these in mathematical form.

4.1 The precision target

Given an exact representation of a function $Q(u)$ and an approximation of it $Q_a(u)$, we can consider various norms on their differences. We will work in the supremum norm for the relative precision over a domain $\mathscr{D}$ to be specified, and seek to minimise the relative error,

$$\epsilon_R = \sup_{\mathscr{D}} \left| \frac{Q_a(u)}{Q(u)} - 1 \right| . \tag{4.1}$$

What this seeks to do is to maximise the quality of the significant figures of the result. We do have to bear in mind that such a goal may be interpreted in two ways. The first approach, that will be taken here, is to set such a target for the theoretical precision of an approximation (typically polynomial or rational), given an abundance of computer precision. The second approach is to set the target based on the actual errors in a given limited precision environment. While the second approach is clearly more pragmatic, the answers that we get will depend significantly on implementation/hardware/compiler details. We will therefore consider the first approach, which allows us to develop the analysis mathematically, and is in common use in modelling. So, for example, the first form (i.e. the simple form without an application of the Newton–Raphson–Halley iteration) of the normal quantile developed by Acklam [1] sets

$$\epsilon_R \lesssim 1.15 \times 10^{-9} \tag{4.2}$$

and our own computational analysis [15] of Wichura's AS 241 [19] suggests that in that case

$$\epsilon_R < 7.45 \times 10^{-17} \tag{4.3}$$

when worked out in high-precision arithmetic. The Acklam error bound is also attained in double-precision arithmetic, while AS 241 errors (in our implementation) peak at about 5×10^{-16}.

In our own studies we have chosen to set the target theoretical precision based on IEEE levels.[2] Given that IEEE single-precision has a 24-bit mantissa, we will aim in single-precision for

$$\epsilon_{Rs} = 2^{-25} \approx 2.98 \times 10^{-8} . \tag{4.4}$$

[2] We are grateful to M. Pont of the Numerical Algorithms Group for advice on these points.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

Since IEEE double-precision has 53 bits of precision, we will set

$$\epsilon_{Rd} = 2^{-54} \approx 5.55 \times 10^{-17} \tag{4.5}$$

for double-precision. So these values are half the values often quoted as 'machine epsilon' for the two cases. The matter is somewhat ethereal in view of different possible rounding methods and compiler variations, and in some circumstances a single-precision target of $2^{-23} \approx 1.19 \times 10^{-7}$ may be perfectly acceptable. The error target might also be expressed in a different norm. So, for example, Giles works with an L_2 on the relative error and a theoretical target of about 10^{-7} in the chapter in GPU Computing Gems [6] for approximating the inverse error function. That work also includes a very useful discussion of realised errors.

4.2 The target domain

This is a somewhat subtle matter, and massively different choices are possible. The quantile mapping for distributions along the entire real line is from the domain $(0, 1)$ to the range that is the real line. In practice computation will take place on an interval

$$U(\epsilon_L, \epsilon_R) = (\epsilon_L, 1 - \epsilon_R) \tag{4.6}$$

and the question for this section is the choice of ϵ_L, ϵ_R. The difficulty is that there is some asymmetry in the matter, as is well discussed on Acklam's web page [1]. It is clear that the value of ϵ_R should be related to the machine epsilon, as there is simply no possibility of getting closer to unity in finite-precision arithmetic. For example, Acklam's discussion considers the errors up to both the points $1 - 2\epsilon_{Rd}$ and $1 - \epsilon_{Rd}$.

The problems are more to do with the choices made for ϵ_L. Acklam [1] takes the view that the quantile needs to have a controlled precision down to at least 2.23×10^{-308}, which is the smallest full precision number that can be given in double-precision, and his results show controlled errors on a still larger range.

If one is interested in representing a mathematical function for diverse applications to as much precision as possible, we take the view that a domain on the lines of the extended form proposed by Acklam [1] is entirely appropriate. However, for Monte Carlo applications this is massively over-engineered. To make the case for a different choice, we consider the typical scope of a Monte Carlo simulation.

Consider a set of N independent uniformly distributed random numbers, $\{U_i\}$, $i = 1, \ldots, N$ taken from the interval $(0, 1)$, and let

$$U_{\min} = \min\{U_i\} \ . \tag{4.7}$$

The distribution of $U_{\min}$ is easily worked out using elementary probability:

$$P(U_{\min} < u) = 1 - P(U_{\min} \geqslant u) = 1 - P(U_i \geqslant u, \ \forall i) = 1 - P(U_i \geqslant u)^N, \tag{4.8}$$

and given that each U_i has a uniform distribution, we see that

$$P(U_{\min} < u) = 1 - (1 - u)^N \tag{4.9}$$

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

and it is then easily computed that

$$E(U_{\min}) = \frac{1}{N+1}\ , \quad \mathrm{Var}(U_{\min}) = \frac{N}{(N+1)^2(N+2)}\ , \tag{4.10}$$

and we see that the average smallest value is about $1/N$ for large N, but the standard deviation is about the same as the mean. Inverting the distribution function, we see that the quantile function for this minimum is given by

$$Q_{\min}(v) = 1 - (1-v)^{1/N} \sim -\frac{1}{N}\log(1-u) \text{ as } N \to \infty, \tag{4.11}$$

which is an exponential distribution. For large N and small U this quantile function is approximately

$$Q_{\min}(v) \sim \frac{u}{N}\ . \tag{4.12}$$

In other words, we have about a 1% chance of encountering numbers lower than

$$\frac{1}{100N} \tag{4.13}$$

in our simulation. Even a rather modest Monte Carlo simulation therefore has a significant chance of producing numbers going well below the machine epsilon in single-precision, though not in double-precision.

One might argue that the influence of a few such small numbers on the overall computation of expectations might be small. The difficulty with such a view is that one can imagine situations where the computation is dominated by minima (lookbacks, options on the worst performer of a basket, systems coupled by a Clayton copula with a lot of downside correlation, deeply out of the money Puts, and so on). Furthermore, the use of antithetic methods might result in low values being reflected to high values as well after the application of the target distribution's quantile function.

In view of this we argue that the largest allowable low-end truncation point should be no higher than about 10^{-10}, meaning that 99% of random numbers would be mapped to the target distribution accurately for samples up to $N = 10^8$ in size. Options should be provided to go to smaller values for special problems requiring very large samples.

The above analysis fits nicely with practical RNGs. Most pseudo- and quasi-RNGs work with 32- or, to a lesser extent, 64-bit unsigned integer data types. So these RNGs output random integral values between 0 and $2^{32}-1$ or $2^{64}-1$. The smallest (non-zero) uniform number that we will encounter is thus $2^{-32} \approx 2.33 \times 10^{-10}$ or $2^{-64} \approx 5.42 \times 10^{-20}$. The largest uniform number (excluding 1) produced by a 32- and 64-bit RNG is $1-2^{-32}$ and $1-2^{-64}$ respectively. However, machine precision can affect these limits. If single-precision numbers are required, the largest number will be limited to $1-2^{-24}$ in either case. If double-precision numbers are requested, the largest number will be $1-2^{-32}$ or $1-2^{-53}$ depending on the RNG.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

4.3 Choice of filtering distribution

Moving now to the particular case of the normal quantile, we have several options for intermediate distributions of interest. The choices we will explore are suitable scalings of the following:

- The one-sided exponential distribution.
- The Student t distribution with two degrees of freedom.
- An 'almost-normal' distribution.

The one-sided exponential distribution has PDF, CDF and quantile given by, for $x > 0$, $0 < u < 1$,

$$f_E(x) = e^{-x}\ , F_E(x) = 1 - e^{-x}\ , Q_E(u) = -\log(1-u)\ . \tag{4.14}$$

The Student t distribution with two degrees of freedom has PDF, CDF and quantile given by, for all real x, $0 < u < 1$,

$$f_t(x) = \frac{1}{(2+x^2)^{3/2}}\ , F_t(x) = \frac{1}{2} + \frac{x}{2\sqrt{x^2+2}}\ , Q_t(u) = \frac{2u-1}{\sqrt{2u(1-u)}}\ . \tag{4.15}$$

The almost-normal distribution[3] is literally one that is very close to a standard Gaussian but has a simple closed-form quantile. The density and distribution functions are given by

$$f_{AN}(x) = \frac{e^{-\frac{2x^2}{\pi}}|x|}{\pi\sqrt{1-e^{-\frac{2x^2}{\pi}}}}, \tag{4.16}$$

$$F_{AN}(x) = \frac{1}{2} + \frac{1}{2}\mathrm{sgn}(x)\sqrt{1-e^{-\frac{2x^2}{\pi}}}, \tag{4.17}$$

and it is then an elementary exercise to compute the quantile function as

$$Q_{AN}(u) = \mathrm{sgn}(u-1/2)\sqrt{-\frac{\pi}{2}\log(4u(1-u))}\ . \tag{4.18}$$

It is informative to look at the current list of approximate normal quantiles to see how prevalent are these changes of variables.

- Our earlier branchless formulae [16] relied on the one-sided exponential distribution.
- Acklam's [1] non-iterated formula is the asymptotic form of the almost-normal distribution in the tail, as is AS 241 [19].

The Moro method [10] uses a double-log in the tail region.

4.4 Computing architecture and compiler issues

In this section we consider some of the details of computer science that impact the mathematical analysis in the context of trying to be as efficient as possible.

[3] This is a distribution that W. Shaw invented for an examination question, and it is left to the reader to compare its density, distribution and quantile with the exact normal.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

4.4.1 *Branching management*

On a central processing unit there is normally very sophisticated management of algorithm branching, to the extent that (a) having a branch at all is managed efficiently, and (b) a slow branch will not compromise the operation of a fast branch. On a GPU we have to confront the issue of *warp divergence*. A warp is a group of 32 threads (in current generation NVIDIA GPUs) that evaluate concurrently. If there is an `if` statement and one thread needs to execute a different branch, then the current behaviour will be for all threads to execute both branches. This is especially problematic in quantile evaluation where typically two or more regions are considered for separate evaluation, with (as noted above) a more computationally expensive change of variables in the tail. There are at least two approaches to alleviate warp divergence:

- Create an essentially branchless algorithm where one formula applies for all relevant values.
- Lower the probability of a warp divergence to a much smaller value.

The first method is the one taken by an earlier version of this study [16], and the second approach is that taken by Giles [6] for the inverse error function. The essential question here is on the balance of effort required to go essentially branchless versus that where an unlikely branch remains, but with a simpler formula in the middle region. In our final model we will develop a third approach based on the idea of warp voting, in which one of the two optimised algorithms will be chosen depending on the warp as a whole.

4.4.2 *Fused multiply-add (FMA) support*

The presence in some architectures of an FMA operation in a single hardware operation creates a performance bias in support of algorithms that rely on repeated applications of operations such as found in Horner's scheme for fast polynomial evaluation, where, for example, a general cubic would be evaluated as

$$p \to d \times x + c\,, \quad p \to p \times x + b\,, \quad p \to p \times x + a\,. \tag{4.19}$$

Given that all modern CUDA GPUs support FMA, we have found that there is no benefit in performing any form of pre-processing to work out polynomials or rational approximations using other devices to reduce the number of multiplication operations. So, for example, the method of quadratic factorisation (see e.g. [15, Section 7]) has been found to produce no speed-up despite reducing the number of multiplications, and special methods available for polynomials of specific degree (e.g. a trick for degree 6 described in exercise 6 in Section 7.2 of [13]), both produce no speed-up, but generate a loss of precision due to several subtractions. So we will work with iterated linear maps on the basis of FMA support whenever a polynomial or rational function has to be evaluated, though we should note that in non-FMA architectures that quadratic factorisation may be a useful route for further optimisation.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

4.4.3 *Optimisation of simple function evaluation*

Given the comments made in Section 4.3, it is important to understand the costs of evaluating $\log x$ and $\sqrt{x}$ compared with working out polynomials of various sizes. In single-precision both functions are similarly expensive so that for the main block of the algorithm for the central quantile, one application of one of these functions is helpful, but applying both is inefficient. For example, the main brake on Acklam's formula is the slow computation of the tail, which as we have noted involves both a logarithm and square root application.

One of the surprises of our analysis is the benefit of the efficiency on GPUs of the reciprocal square root function `rsqrt`, especially in double-precision. This means that the t_2 distribution, suitably scaled, is an unexpectedly efficient filtering distribution. We will return to this later.

5 Numerical approximation of the normal quantile for GPUs

We now focus on the normal quantile Φ^{-1} with emphasis on computing it efficiently on GPUs.

5.1 Existing methods

There are several well-known approximations for $\Phi^{-1}(u) = \sqrt{2}\ \mathrm{erf}^{-1}(2u-1)$. These all use a rational approximation for $u \in [0+b, 1-b]$, where $b > 0$ is the tail *breakpoint*. The other values in $(0,1)$ are covered by at least one additional polynomial or rational approximation, which are in terms of a computationally expensive change of variable. The reason for this two-pronged approach is because Φ^{-1} has rather a split personality. It is slowly varying in the central region where u is between about 0.1 and 0.9, and then diverges to infinity as $u \to 0^+$ or $u \to 1_-$. This is shown in Figure 1. The values of b for existing algorithms are as follows:

- $b = 0.08$ for Moro [10],
- $b = 0.075$ for Wichura (Algorithm AS 241) [19],
- $b = 0.02425$ for Acklam [1].

The two-pronged approach is absolutely fine for central processing units, where branch prediction means that virtually no performance penalty is incurred for branching. Things are quite different for GPUs, which are sensitive to branching. Suppose Wichura's algorithm is run on a Nvidia Fermi or Kepler GPU. Threads on such a GPU execute in groups of 32. Nvidia call these groups *warps*. Assuming uniformly distributed input, the probability that all threads in a warp use the central approximation is $0.85^{32} \approx 0.006$. The equivalent probability for tail approximation is negligible, so there is a 99% likelihood of occurrence of branch divergence. Branch divergence will be minimised if b for a particular approximation is sufficiently small.

New approximations of special functions that deliberately address branch divergence on GPUs can be found in the literature. An earlier version of this study, [16], first gave branch-free approximations for Φ^{-1}. Giles [6, p. 1] has given single- and double-precision

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

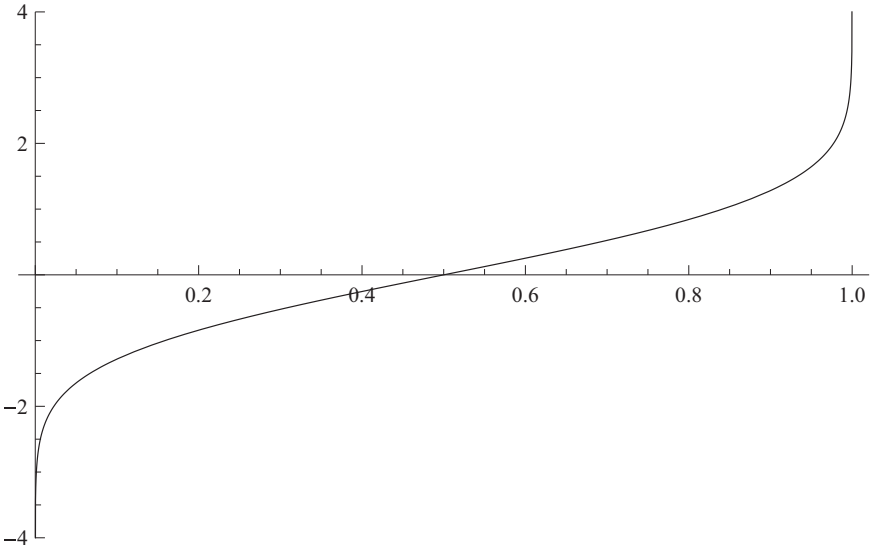


FIGURE 1. The normal quantile Φ^{-1}.

approximations for erf^{-1} and advocates its use for computing Φ^{-1}. The inverse error function can be found in the Intel Math Kernel Library (MKL) and the NVIDIA CUDA Math Library, implemented as `erfinv`. However, both libraries also provide the inverse complementary error function `erfcinv`, for good reason. $\mathrm{erfc}^{-1}(x) = \mathrm{erf}^{-1}(1-x)$, so what is the point of `erfcinv`? Suppose $\Phi^{-1}(u)$ is implemented in the form $\sqrt{2}\ \mathrm{erf}^{-1}(2u-1)$. Working in floating-point arithmetic, if u is less than machine epsilon, then $2u-1=-1$, so $\mathrm{erf}^{-1}(2u-1)$ will return $-\infty$ instead of a finite negative value. For u slightly larger than machine epsilon, there will be a partial loss of precision, because the density of floating-point numbers is far less around -1 than zero. One should instead implement $\Phi^{-1}(u)$ in the form $-\sqrt{2}\ \mathrm{erfc}^{-1}(2u)$ to maintain precision for low u, even though it is generally slower than `erfinv`. CUDA 5.0 onwards includes the normal quantile function `normcdfinv`, which is indeed based on `erfcinv`.

We propose filtering on a suitable intermediary distribution is an effective way to alleviate branch divergence. We found exponential and Student t backgrounds to be useful for the normal quantile.

5.2 Normal samples from an exponential background

Now 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-hand region by odd symmetry. The Recycling ODE in the right-hand region, $v \geqslant 0$ is simply

$$\frac{d^2Q}{dv^2} + \frac{dQ}{dv} = Q\left(\frac{dQ}{dv}\right)^2 \tag{5.1}$$

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

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

$$Q(v) = \Phi^{-1}(1 - 1/2\, e^{-v}) \,. \tag{5.2}$$

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 $v = 0$. The series solution is easily found to be, using exact coefficients,

$$\begin{aligned} Q(v) = & \sqrt{\frac{\pi}{2}} v - \frac{1}{2}\sqrt{\frac{\pi}{2}} v^2 + \frac{\left(2\sqrt{\pi} + \pi^{3/2}\right) v^3}{12\sqrt{2}} - \frac{\left(\sqrt{\pi} + 3\pi^{3/2}\right) v^4}{24\sqrt{2}} \\ & + \frac{\left(4\sqrt{\pi} + 50\pi^{3/2} + 7\pi^{5/2}\right) v^5}{480\sqrt{2}} - \frac{\left(4\sqrt{\pi} + 180\pi^{3/2} + 105\pi^{5/2}\right) v^6}{2880\sqrt{2}} \\ & + \frac{\left(8\sqrt{\pi} + 1204\pi^{3/2} + 1960\pi^{5/2} + 127\pi^{7/2}\right) v^7}{40320\sqrt{2}} \\ & - \frac{\left(2\sqrt{\pi} + 966\pi^{3/2} + 3675\pi^{5/2} + 889\pi^{7/2}\right) v^8}{80640\sqrt{2}} \\ & + \frac{\left(16\sqrt{\pi} + 24200\pi^{3/2} + 194628\pi^{5/2} + 117348\pi^{7/2} + 4369\pi^{9/2}\right) v^9}{5806080\sqrt{2}} \\ & - \frac{\left(16\sqrt{\pi} + 74640\pi^{3/2} + 1190700\pi^{5/2} + 1493520\pi^{7/2} + 196605\pi^{9/2}\right) v^{10}}{58060800\sqrt{2}} \\ & + O\left(v^{11}\right) . \end{aligned} \tag{5.3}$$

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 differential equations about several points. However, as already noted, an important point for modern computation is to try to avoid `if` statements in the computer implementation. In Figure 2 we have plotted the function $Q(v)$ in the range $0 \leqslant v \leqslant 44$, which is equivalent to $0.5 \leqslant u < 1 - 3.89 \times 10^{-20}$, showing the single character of the function, that will allow a single rational approximation to be made. It is then a matter of picking a target range and precision for the desired result.

For completeness, an algorithm for normal samples based on this 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):

(1) sample u in $1/2 \leqslant u < 1$ (or, better, $0 < u < 1$);

(2) evaluate $v = -\log[2(1-u)]$ (then, better, $v = -\log[u]$);

(3) evaluate $Z = Q(v)$ with Q given by a rational approximation;

(4) output the antithetic pair $Z, -Z$.

If an exponential base is used, we are essentially employing the last two steps. This is what should be done ideally to get maximum precision.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

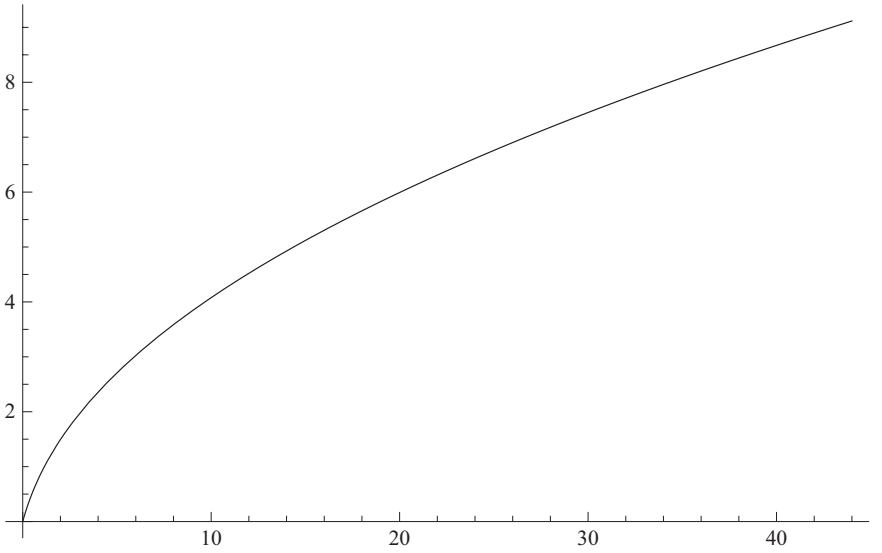


FIGURE 2. The normal quantile in exponential coordinates.

5.2.1 *Branchless approximation for single-precision*

We shall now proceed to consider the construction of a quantile approximation optimised for speed subject to a precision goal of $\epsilon_{Rs} = 2^{-25} \approx 2.98 \times 10^{-8}$ working down to about 10^{-10}, based on the essentially branchless approach and an exponential filtering distribution. 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 of $Q(v)$. The function actually approximated was

$$\frac{Q(v)}{v} = \frac{1}{v}\Phi^{-1}(1 - 1/2\, e^{-v}), \tag{5.4}$$

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 $Q(0) = 0$.

A rational approximation of degree 6 was found with the desired precision. The relative error is shown in Figure 3 and is less than 2.98×10^{-8} on $0 \leqslant v \leqslant 22$, corresponding to u differing from zero by 1.39×10^{-10}. The resulting form for $Q(v)$ is as follows:

$$Q(v) = v\,\frac{p(v)}{q(v)}, \tag{5.5}$$

where both p and q are polynomials of degree 6, with nested C language forms as follows, where we reproduce the higher precision output generated by *Mathematica*. The numerator p is

```
1.253314132218523976 +
   v*(2.7484757927894917811 +
      v*(1.8611004656658237299 +
```

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

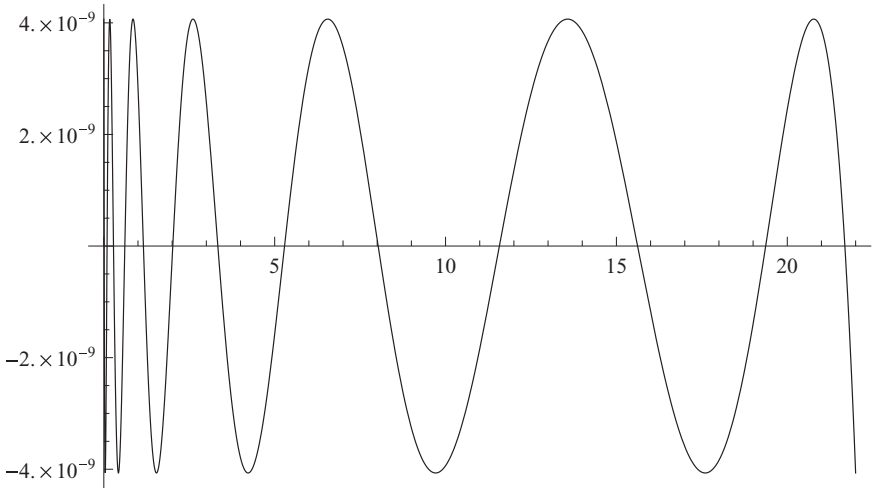


FIGURE 3. Precision of single-precision branchless exponential-normal quantile on $[0, 22]$.

```
v*(0.45004430722390362624 +
   v*(0.03602364405755013184 +
      (0.00074043143125639889353 +
         1.6826777496563851759e-6*v)*v))))
```

and the denominator q is

```
1 + v*(2.692965914291489687 +
       v*(2.4029694317855107369 +
          v*(0.84102029872961135341 +
             v*(0.11017016365113210302 +
                (0.004513659248999904147 1 +
                   0.00003709787137959748864 4*v)*v))))
```

For clarity, Algorithm 1 gives the pseudocode for our new normal quantile algorithm based on a branchless approximation. Code implementing this is given in CUDA device function form in Appendix A.

Algorithm 1 Branchless approximation of the normal quantile

Input: $u \in (0, 1)$
Output: $\Phi^{-1}(u)$
  if $u > \frac{1}{2}$ **then**
    $u \leftarrow 1 - u$
  end if
  $v \leftarrow -\log(2u)$ ▷ double-sided exponential quantile
  return $\operatorname{sgn}\left(u - \frac{1}{2}\right) Q(v)$ ▷ Q rational approximation

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

5.2.2 *Low-probability branch divergence approximation for single-precision*

The branchless solution uses a single rational approximation over the interval of interest. It covers $2^{-32} \leqslant u \leqslant 1$ by transforming two-sided exponential variates in the range $[-22, 22]$. Most exponential variates will be in a much shorter range, most of the time; $(-8, 8)$, for example, covers $e^{-8}/2 < u < 1 - e^{-8}/2 \approx 0.999832$. Bearing in mind, because of symmetry, only $[0, 22]$ has to be considered, a shorter rational approximation can cover $[0, 8)$ and another can manage $[8, 22]$. Of course, this does introduce the possibility of branch divergence, but the probability is relatively small – most input will still be handled by the same rational approximation, most of the time. On a CUDA GPU with a warp size of 32, the probability of not diverging is about $0.999832^{32} \approx 0.994647$. Even branching at $v = 6$ results in an acceptable level of branch divergence: 0.038908. Giles' [6] approach to approximating erf^{-1} is based on a branching formula with a low warp divergence probability. NVIDIA's build-in CUDA single-precision inverse error function is modelled on Giles' version because of this.

A single-precision rational approximation of degree $(4, 5)$ was found for $[0, 7]$ and one of degree $(3, 4)$ to cover $[7, 22]$. Using these in a branched approximation scheme actually had a slight adverse effect on speed. It turned out that the overhead of simply evaluating an `if` statement negated the advantage gained by splitting $[0, 22]$. (Other sensible splits were tested.) However, branching allows us to cover the deep lower tail without compromising performance significantly. The smallest normal positive single-precision value is $2^{-126} \approx 1.18 \times 10^{-38}$, which requires an approximation over $[0, 87]$ since $e^{-87}/2 \approx 8.23 \times 10^{-39}$. A single-precision rational approximation of degree $(3, 4)$ was found for $[22, 87]$. Appendix B gives the CUDA implementation of the branched algorithm for single-precision.

5.2.3 *Low-probability branch divergence approximation for double-precision*

The need for double-precision is somewhat contentious in the Monte Carlo framework, but it seems to us that while some computations might be satisfactory in single-precision, one does need to check. While purely additive Monte Carlo summations are almost certainly OK, one can never be sure about any computation in single-precision until one has seen that repeating in double-precision gives consistent answers. Furthermore, the commonplace approach to working out mathematical derivatives by differencing can generate a substantial precision loss, as it involves an essential subtraction. These are well-established concerns. To these we add the issues associated with generating regions in the right tail with u close to unity, for which the single-precision form of machine epsilon is a killer issue. The right tail no longer has a proper distribution of values but instead has a concentration of particular values of

$$\Phi^{-1}(1 - 2^{-24}) = 5.2497 \text{ or } \Phi^{-1}(1 - 2^{-23}) = 5.16658 \tag{5.6}$$

and no higher. Thus, the sample is thereby distorted and capped at a low value. While this problem persists in double-precision, the limiting levels (as is already noted by Acklam [1]) are at

$$\Phi^{-1}(1 - 2^{-54}) = 8.29236 \text{ or } \Phi^{-1}(1 - 2^{-53}) = 8.20954, \tag{5.7}$$

which are unlikely to be achieved in a simulation of foreseeable size.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

A double-precision rational approximation of degree 10, with accuracy of no less than $\epsilon_{Rd} = 5.55 \times 10^{-17}$, was found for $[0, 8]$ and one of degree $(6, 7)$ to cover $[8, 22]$. The smallest normal double-precision value is $2^{-1022} \approx 2.23 \times 10^{-308}$, which requires an approximation over $[0, 708]$ since $e^{-708}/2 \approx 1.65 \times 10^{-308}$. A rational approximation of degree 6 was found for $[22, 44]$. Up to $v = 44$ covers 64-bit RNGs ($2^{-64} \approx 5.42 \times 10^{-20}$ and $e^{-44}/2 \approx 3.89 \times 10^{-20}$). Up to $v = 708$ is possible with several approximations, but delegating to the NVIDIA algorithm turned out to be better for $44 \leqslant v \leqslant 708$. Appendix C gives the CUDA implementation of the branched algorithm for double-precision.

5.3 Using a Student t-inspired filter

An interesting feature of GPUs is their unusually efficient implementation of the reciprocal square root function,

$$\texttt{rsqrt}(x) \equiv \frac{1}{\sqrt{x}}, \tag{5.8}$$

and in double-precision this is *much* faster than the logarithm function. This suggests that we explore a sequence in which a uniform random variable is mapped to a suitable sample of the Student t distribution with *two* degrees of freedom, and then this is re-mapped by appropriate solution of the Recycling ODE. Specifically, we will consider an intermediate random variable that is of the form

$$V = \frac{t_2}{\sqrt{2}}\,. \tag{5.9}$$

The quantile function for this random variable is just

$$\frac{u - 1/2}{\sqrt{u(1-u)}}, \tag{5.10}$$

and the mapping from V to a normal random variable is the composition of the CDF of V with the probit function:

$$Q_t(v) = \Phi^{-1}\left(\frac{1}{2}\left[1 + \frac{v}{\sqrt{1+v^2}}\right]\right). \tag{5.11}$$

We have experimented with rational approximations of Q_t, and while it is difficult to give an economical branchless solution, it serves as the basis for a very fast branching solution but with low warp divergence. Specifically, we found a rational approximation of degree $(14, 15)$ that satisfies the double-precision error bound ϵ_{Rd} for $0 \leqslant v \leqslant 15.5$. Furthermore,

$$Q_t(15.5) \approx 3.07933 \tag{5.12}$$

for which $u \approx 0.998963$, giving a low warp divergence probability. This therefore provides the basis for a *hybrid* model in which we put back a breakpoint, but very deep into the tail. In the central region the computation is done by using the scaled t_2 as an intermediate distribution, and in the tail the computation is done using the exponential recycling solution from Section 5.2.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

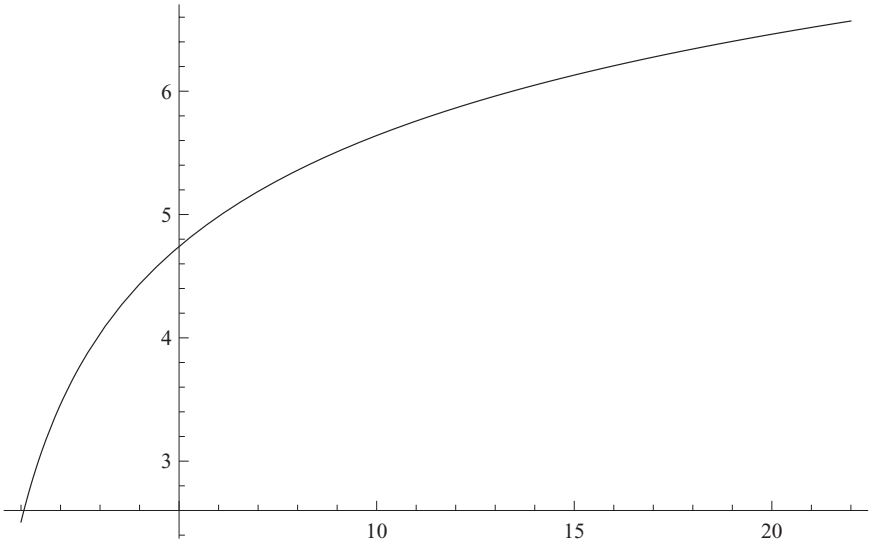


FIGURE 4. The $\tilde{t}_2$ transformation on $[0, 22]$.

We can actually do better and make more economical approximations. Let us consider

$$\Psi(u) = \frac{\Phi^{-1}(u)}{u - \frac{1}{2}} \ . \tag{5.13}$$

Let $v = \tilde{q}(u)$, where $\tilde{q}$ is any invertible function, and write

$$\tilde{Q}(v) = \Psi(\tilde{q}^{-1}(v)) = \Psi(u) \ . \tag{5.14}$$

If

$$\tilde{q}(u) = \frac{1}{2\sqrt{u(1-u)}} \tag{5.15}$$

then

$$\tilde{q}^{-1}(v) = \frac{1}{2}\left(1 + \sqrt{1 - \frac{1}{v^2}}\right) \tag{5.16}$$

for $u \geqslant 1/2$. $\tilde{Q}(v)$ is slightly easier to approximate than $Q_t(v)$ as shown in Figure 4. For single-precision, a rational approximation of degree 5 for $\tilde{Q}(v)$ was found for $1 \leqslant v \leqslant 21$. ($\tilde{q}^{-1}(21) \approx 0.999433$.) For double-precision, a rational approximation of degree 11 was found for $1 \leqslant v \leqslant 22$. ($\tilde{q}^{-1}(22) \approx 0.999483$.) These ranges are very good, providing lower probabilities of branch divergence. The aforementioned change of variable will be referred to as $\tilde{t}_2$ henceforth.

5.3.1 *Hybrid approximation for single-precision*

Algorithm 2 gives the pseudocode for our new normal quantile algorithm based on an hybrid approximation, which uses the $\tilde{t}_2$ and exponential changes of variable. Appendix D gives the single-precision CUDA implementation of this.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

Algorithm 2 Hybrid approximation of the normal quantile for single-precision

Input: $u \in (0,1)$
Output: $\Phi^{-1}(u)$
  $v \leftarrow \frac{1}{2}\sqrt{u(1-u)}^{-1}$ ▷ $\tilde{q}(u)$
  if $v \leqslant 21$ **then** ▷ $v > 21$ cannot be handled
    return $\left(u - \frac{1}{2}\right)\tilde{Q}(v)$ ▷ $\tilde{Q}$ rational approximation
  else
    if $u > \frac{1}{2}$ **then**
      $u \leftarrow 1 - u$
    end if
    $v \leftarrow -\log(2u)$ ▷ double-sided exponential quantile
    return $\operatorname{sgn}\left(u - \frac{1}{2}\right) Q(v)$ ▷ Q rational approximation
  end if

5.3.2 *Hybrid approximation for double-precision*

All CUDA-enabled GPUs that support double-precision floating-point numbers also allow the threads within a warp to cooperate. Specifically, the 32 threads in a warp can take a binary vote [11, Section B.13]. In this scheme, each thread simultaneously casts its yes/no (`true`/`false`) vote and then is notified for everyone else's vote. The threads can alternatively receive a summary of the votes in the form of logical AND or inclusive OR of them.

This warp voting functionality can be exploited in the hybrid solution to further alleviate branch divergence. We have the fast $\tilde{t}_2$ transformation that is suitable for most warps for most of the time. We also have a comparatively slow exponential transformation that is suitable for all warps. In the (rare) event of a thread in a warp needing to use this transformation because the $\tilde{t}_2$ transformation is unsuitable, branch divergence occurs, which means that two sets of quantiles and rational functions are computed. Only one set per warp is actually required: if *any* of the threads in the warp strictly needs to use the exponential transformation, force every other thread to also use the transformation (irrespective of their $\tilde{t}_2$ eligibility); otherwise, *all* of the threads can just use the $\tilde{t}_2$ transformation. This all-or-nothing strategy can be implemented using warp voting. The relevant CUDA code is `all(v < X)`, where `X` is upper limit of the $\tilde{t}_2$ range, i.e. `21` or `22` depending on single- or double-precision. The `all` function returns `true` if and only if all `v` in the warp are less than `X`.

The pseudocode for our double-precision hybrid normal quantile algorithm is essentially the same as Algorithm 2, but '$v \leqslant 21$' is substituted for 'all v in the warp are $\leqslant 22$'. Appendix E gives the CUDA implementation of this.

5.4 Performance and accuracy

We benchmarked the performance of our new algorithms against well-established candidates. These are as follows:

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

- Moro [10]
- Wichura (Algorithm AS 241) [19]
- Acklam [1]

(1) Unrefined form, which targets a relative error of no larger than 1.15×10^{-9}.

(2) Refined form, where the level one approximation is fed once through the Newton–Raphson–Halley iteration.

We also considered a normal quantile based on NVIDIA's CUDA Math Library `erfcinv` function.

Both single- and double-precision variants of each algorithm were evaluated where possible. Wichura [19] gave two versions of his algorithm: PPND7, accurate to around seven significant figures; and PPND16, accurate to around 16 significant figures. These map very nicely to single- and double-precision implementations. Unlike Wichura [19], neither Moro [10] nor Acklam [1] give single-precision coefficients for their approximations, so they only feature in our double-precision tests.

Faithful CUDA GPU implementations of the algorithms were written by this author. Wichura's [19] Fortran implementation of his algorithm was ported to CUDA. The implementation of Acklam's algorithm [1] was based on the pseudocode on his website. CUDA's built-in `erfc` function was used for the Halley refinement suggested by Acklam [1]. Moro's algorithm [10] was implemented using the pseudocode in [7]. All of the code written was optimised as far as possible for the GPU architecture, e.g. divisions were factored out of conditional statements.

The performance of our new normal quantile algorithms and their competition were evaluated on three high-end NVIDIA GPUs:

- GeForce GTX Titan;
- GeForce GTX 480;
- Tesla C2050.

The GTX Titan is based on NVIDIA's Kepler architecture, while the remaining two are previous generation Fermi GPUs. We used CUDA 5.5 software, the most recent production release at the time of writing this paper.

5.4.1 *Precision*

How do we assess the quality of the 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 on the *Wikipedia* page on the probit [21], and as a series has been coded up in both *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 agrees with *Mathematica*'s internal `InverseErf` to a precision of better than 10^{-29} on the interval $[e, 1-e]$, with

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

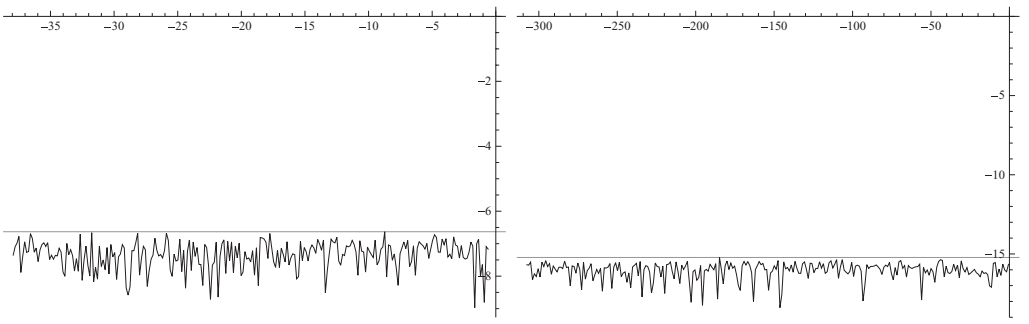


FIGURE 5. Precision of single- (left) and double-precision (right) hybrid approximations in left tail.

$e = 0.0007$. 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.

We will now focus attention on realised precision, and this has been evaluated directly using *Mathematica* 8's capability to call our CUDA functions. We send a controlled sequence of floating-point numbers in $(0, 1)$ to a function of interest, pull the results back and compare with our high-precision benchmark. The plots in this section show $\log_{10}|\text{approx}/\text{exact} - 1|$ as a function of $\log_{10}(u)$.

The new branchless formula preserves a precision better than 4.1×10^{-7} down to about $u = 10^{-10}$. Figure 5 shows that our single-precision hybrid approximation maintains accuracy right down to the smallest non-zero input. Extensive tests suggest its relative error peaks at about 3.91×10^{-7}. The figure also shows the left-tail outcome for the double-precision hybrid model. This confirms that double-precision is realised consistently throughout the region of interest. Between 2^{-32} and 1, the hybrid approximation's peak relative error is about 8.58×10^{-16}.

5.4.2 *Speed*

The speed of each algorithm was evaluated by generating one billion random inputs and measuring the time to apply the quantile function to them. NVIDIA's CUDA Random Number Generation library was used to generate the uniformly distributed random numbers on GPU memory. The numbers were pseudorandom and generated by MRG32k3a. The quantile algorithms were accurately timed using CUDA events as per the CUDA C Programming Guide [11, Section 3.2.5.6].

The timings for single-precision are given in Table 1. The hybrid algorithm is the fastest, having a slight edge over NVIDIA. Our branchless method comes third behind the hybrid and NVIDIA algorithms, but has the merit of robustness with respect to any future increase in the number of threads per warp. It may also be of value for other vector/parallel architectures such as Intel's Xeon Phi. 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.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

Table 1. *Single-precision timings in ms for each algorithm to compute the normal quantile for 1 billion pseudorandom uniform variates on the test* NVIDIA *GPUs*

Single-precision algorithm	GTX Titan	GTX 480	C2050
Wichura	54.89	135.62	178.62
NVIDIA	45.38	76.56	102.65
Branchless	50.74	118.02	155.04
Branched	52.58	124.63	164.08
Hybrid	43.52	72.40	97.12

Table 2. *Double-precision timings in ms for each algorithm to compute the normal quantile for 1 billion pseudorandom uniform variates on the test* NVIDIA *GPUs*

Double-precision algorithm	GTX Titan	GTX 480	C2050
Moro	907.58	1207.81	697.55
Wichura	853.20	1090.50	661.39
Acklam (unrefined)	665.94	877.76	567.18
Acklam (refined)	2034.90	2662.70	1585.58
NVIDIA	553.13	703.95	431.19
Branched	656.52	879.90	540.58
Hybrid	412.65	519.28	378.94

The timings for double-precision, reported in Table 2, are very interesting. The use of the $\tilde{t}_2$ distribution as an intermediate distribution results in a clear win for this method on performance grounds, and given that it also provides clear double-precision accuracy over the range of interest for Monte Carlo work, we argue that it is a good candidate for the preferred method on GPUs. The advantage arises from the relatively fast speed of the inverse square root operation compared with the logarithm in double-precision on NVIDIA GPUs. (We do not see such a pronounced advantage in single-precision.) The improvement over other algorithms can make a difference, especially if one is solving a stochastic differential equation via many calls to a normal random variable prior to evaluating a payoff.

6 Conclusions

This paper has developed two related threads of thought. First, 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 to the samples of the solution of a recycling differential equation. The recycling differential equation is the ODE for transforming samples from a density f_0 to a density f_1, and is

$$\frac{d^2Q(v)}{dv^2} + H_0(v)\frac{dQ(v)}{dv} = H_1(Q(v))\left(\frac{dQ(v)}{dv}\right)^2,$$

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

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 variables 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 Recycling ODE with 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 Recycling ODE 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 its characteristic function is known, other methods must be used. Investigations of the resulting integro-differential equations will be reported elsewhere.

The second thread of thought concerns the detailed practicalities of building good quantile implementations for Monte Carlo use. We have set out the numerical and computer science issues that are relevant to this in general. Then, as a particular application of these methods for recycling, we have carried out an extensive re-evaluation of the methods used for computing the normal quantile, especially in a GPU environment. We have reported new formulae for the normal quantile in both single- and double-precisions. They outperform all other methods known to us on GPU speed while preserving precision on a range more than enough for Monte Carlo simulations.

Acknowledgements

Thomas Luu wishes to thank the Industrial Mathematics Knowledge Transfer Network, the Engineering and Physical Sciences Research Council and the Numerical Algorithms Group for support under a KTN–EPSRC Case award. William T. Shaw wishes to thank I. Buckley, W. Gilchrist, P. Jäckel, D. Scott, G. Steinbrecher, A. Munir 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 a London workshop on GPU computing [3] stimulated the development of the essentially "if-less" normal quantile. A. Munir provided assistance in producing further Windows binaries for the Cuda architecture. Clarification of precision issues from several members of the Numerical Algorithms Group were invaluable, and we are also grateful to the company for access to their Nvidia Tesla system. We are also grateful to Wolfram Research for their creation of the Cuda embedding tools in *Mathematica* 8 that allowed the precision studies here to be made so easily, and to Tom Wickham-Jones for clarifying some points on double-precision analysis in that environment.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

Availability of software

An implementation of the GPU-optimised normal quantile approximations will be available in the NAG Numerical Routines for GPUs.[4] Anyone who wishes to obtain this should contact NAG either through the website `www.nag.co.uk`, or via email at `infodesk@nag.co.uk`.

Appendix A Branchless approximation for single-precision

```
__device__ float branchless(float u)
{
    float ushift = u - 0.5f;
    if (ushift > 0.0f) u = 1.0f - u;
    float v = -logf(u + u);

    float p =   1.68267776058639e-6f;
    p = p * v + 0.0007404314351202936f;
    p = p * v + 0.03602364419560667f;
    p = p * v + 0.4500443083534446f;
    p = p * v + 1.861100468283588f;
    p = p * v + 2.748475794390544f;
    p = p * v + 1.253314132218524f;

    float q =   0.00003709787159774307f;
    q = q * v + 0.004513659269519104f;
    q = q * v + 0.1101701640048184f;
    q = q * v + 0.8410203004476538f;
    q = q * v + 2.402969434512837f;
    q = q * v + 2.692965915568952f;
    q = q * v + 1.0f;

    return __fdividef(p, q) * copysignf(v, ushift);
}
```

Appendix B Branched approximation for single-precision

```
__device__ float branched(float u)
{
    float ushift = u - 0.5f;
    if (ushift > 0.0f) u = 1.0f - u;
    float v = -logf(u + u);
```

[4] `http://www.nag.co.uk/numeric/GPUs/index.asp`

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

```
    float p, q;
    if (v < 22.0f) {
        p =         1.68267776058639e-6f;
        p = p * v + 0.0007404314351202936f;
        p = p * v + 0.03602364419560667f;
        p = p * v + 0.4500443083534446f;
        p = p * v + 1.861100468283588f;
        p = p * v + 2.748475794390544f;
        p = p * v + 1.253314132218524f;

        q =         0.00003709787159774307f;
        q = q * v + 0.004513659269519104f;
        q = q * v + 0.1101701640048184f;
        q = q * v + 0.8410203004476538f;
        q = q * v + 2.402969434512837f;
        q = q * v + 2.692965915568952f;
    } else {
        p =         0.00001016962895771568f;
        p = p * v + 0.003330096951634844f;
        p = p * v + 0.1540146885433827f;
        p = p * v + 1.045480394868638f;

        q =         1.303450553973082e-7f;
        q = q * v + 0.0001728926914526662f;
        q = q * v + 0.02031866871146244f;
        q = q * v + 0.3977137974626933f;
    }
    p *= copysignf(v, ushift);
    q = q * v + 1.0f;

    return __fdividef(p, q);
}
```

Appendix C Branched approximation for double-precision

```
__device__ double branched(double u)
{
    double ushift = u - 0.5;
    if (ushift > 0.0) u = 1.0 - u;
    double v = -log(u + u);
```

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

```
double p, q;
if (v < 8.0) {
    p =         1.34951886838167805875324 9e-8;
    p = p * v + 8.52746651977185243009898e-6;
    p = p * v + 0.00074063772735021957133007 82;
    p = p * v + 0.02110410727013085360335842;
    p = p * v + 0.26237384945136259273579 95;
    p = p * v + 1.63640952569483930879691 2;
    p = p * v + 5.51880861798760045711393 2;
    p = p * v + 10.40118297266547564032922;
    p = p * v + 10.86112393025250379368 94;
    p = p * v + 5.83596552394336649440944 2;
    p = p * v + 1.25331413731550021884663 8;

    q =         3.47539058439584852352887 9e-7;
    q = q * v + 0.00006713843966407750619673244;
    q = q * v + 0.00323473213770173088103963 8;
    q = q * v + 0.06142247251750825316341193;
    q = q * v + 0.55989835613022535055107 79;
    q = q * v + 2.72393321132616879584782 5;
    q = q * v + 7.49538065102905808981051 4;
    q = q * v + 11.93179043337747424406973;
    q = q * v + 10.81567043691618587425845;
    q = q * v + 5.15642678893220502741624 9;
} else if (v < 22.0) {
    p =         5.90554708112176250651658 9e-8;
    p = p * v + 0.00002242239919190552389396371;
    p = p * v + 0.00182337296360977179677029 7;
    p = p * v + 0.04741412290058875930061007;
    p = p * v + 0.44149129800327179905490 48;
    p = p * v + 1.42498375920143834414840 4;
    p = p * v + 1.25042974670753215556787 7;

    q =         7.82071503252677790408375 2e-10;
    q = q * v + 1.03824580342612109924183 5e-6;
    q = q * v + 0.00016911529561368535771539 63;
    q = q * v + 0.00773458514233098898794149 47;
    q = q * v + 0.12482308775876290069799 56;
    q = q * v + 0.75469086137062846502432 54;
    q = q * v + 1.62895319823209931637785 9;
} else if (v < 44.0) {
    p =         5.61039042686373150685202 6e-11;
    p = p * v + 1.42117107484504824418430 2e-7;
    p = p * v + 0.00004247705115079861566951468;
```

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

```
        p = p * v + 0.00335672340290416480538209 6;
        p = p * v + 0.08468194876191724948853;
        p = p * v + 0.66457017357550759919188 87;
        p = p * v + 1.21805414289807320927963 9;

        q =         2.93853660559288451170975 7e-9;
        q = q * v + 2.10886776680013426652130 3e-6;
        q = q * v + 0.00031329395179411783976 87649;
        q = q * v + 0.01414790841185618108831795;
        q = q * v + 0.21163134088065315490230 63;
        q = q * v + 0.96974786411413743833596 39;
    } else {
        return -CUDART_SQRT_TWO * erfcinv(u + u);
    }
    p *= copysign(v, ushift);
    q = q * v + 1.0;
    return p / q;
}
```

Appendix D Hybrid approximation for single-precision

```
__device__ float hybrid(float u)
{
    float v, p, q, ushift;

    ushift = u - 0.5f;

    v = copysignf(ushift, 0.0f);

    if (v < 0.499433f) {
        asm("rsqrt.approx.ftz.f32 %0,%1;" : "=f"(v) : "f"(u - u * u));
        v *= 0.5f;

        p =         0.001732781974270904f;
        p = p * v + 0.1788417306083325f;
        p = p * v + 2.804338363421083f;
        p = p * v + 9.35716893191325f;
        p = p * v + 5.283080058166861f;
        p = p * v + 0.07885390444279965f;
        p *= ushift;
        q =         0.0001796248328874524f;
        q = q * v + 0.02398533988976253f;
        q = q * v + 0.4893072798067982f;
        q = q * v + 2.406460595830034f;
        q = q * v + 3.142947488363618f;
```

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

```
    } else {
        if (ushift > 0.0f) u = 1.0f - u;
        asm("lg2.approx.ftz.f32 %0,%1;" : "=f"(v) : "f"(u + u));
        v *= -0.6931471805599453f;

        if (v < 22.0f) {
            p =         0.000382438382914666f;
            p = p * v + 0.03679041341785685f;
            p = p * v + 0.5242351532484291f;
            p = p * v + 1.21642047402659f;
            q =         9.14019972725528e-6f;
            q = q * v + 0.003523083799369908f;
            q = q * v + 0.126802543865968f;
            q = q * v + 0.8502031783957995f;
        } else {
            p =         0.00001016962895771568f;
            p = p * v + 0.003330096951634844f;
            p = p * v + 0.1540146885433827f;
            p = p * v + 1.045480394868638f;
            q =         1.303450553973082e-7f;
            q = q * v + 0.0001728926914526662f;
            q = q * v + 0.02031866871146244f;
            q = q * v + 0.3977137974626933f;
        }
        p *= copysignf(v, ushift);
    }
    q = q * v + 1.0f;
    asm("rcp.approx.ftz.f32 %0,%1;" : "=f"(v) : "f"(q));
    return p * v;
}
```

Appendix E Hybrid approximation for double-precision

```
__device__ double hybrid(double u)
{
    double v, p, q, ushift;

    ushift = u - 0.5;

    v = copysign(ushift, 0.0);

    if (all(v < 0.499483203996)) {
        v = rsqrt(u - u * u);
        v *= 0.5;
```

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

```
        p =          7.27442282797733047104018730 82e-8;
        p = p * v + 0.00004217438718138828581153268755947;
        p = p * v + 0.00520379665440516934032831487884;
        p = p * v + 0.22526456084447032749113748 6146;
        p = p * v + 4.14932318033989880708071080023;
        p = p * v + 35.47676305041613799971236107 15;
        p = p * v + 145.07337603813063126351814748 7;
        p = p * v + 279.37531211744014368336302233 3;
        p = p * v + 236.36549858670068022124322594 4;
        p = p * v + 75.03054485443986941838063326 34;
        p = p * v + 6.39207201315300014457052458023;
        p = p * v + 0.01679258875150793314602304791 24;
        p *= ushift;
        q =          6.75247954960365765875437842188e-9;
        q = q * v + 4.65945444291242666572785117414e-6;
        q = q * v + 0.00065520075255587976474589504 9551;
        q = q * v + 0.03211851764177402202943745747 23;
        q = q * v + 0.67590678908187778768942583445 13;
        q = q * v + 6.75231637016836002442853621662;
        q = q * v + 33.61825107009507893259817398 77;
        q = q * v + 84.73008513522549816433073751 48;
        q = q * v + 106.64259434339609057760907922 3;
        q = q * v + 63.36259362037864446009880735 68;
        q = q * v + 15.20229822124271666863412783 83;
    } else {
        if (ushift > 0.0) u = 1.0 - u;
        v = -log(u + u);

        if (all(v < 44.0)) {
            if (v < 8.0) {
                p =          1.34951886838167805875324 9e-8;
                p = p * v + 8.52746651977185243009898e-6;
                p = p * v + 0.00074063772735021957133007 82;
                p = p * v + 0.02110410727013085360335842;
                p = p * v + 0.26237384945136259273579 95;
                p = p * v + 1.63640952569483930879691 2;
                p = p * v + 5.51880861798760045711393 2;
                p = p * v + 10.40118297266547564032922;
                p = p * v + 10.86112393025250379368 94;
                p = p * v + 5.83596552394336649440944 2;
                p = p * v + 1.25331413731550021884663 8;
                q =          3.47539058439584852352887 9e-7;
                q = q * v + 0.00006713843966407750619673244;
                q = q * v + 0.00323473213770173088103963 8;
```

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

```
            q = q * v + 0.06142247251750825316341193;
            q = q * v + 0.55989835613022535055107790;
            q = q * v + 2.72393321132616879584782500;
            q = q * v + 7.49538065102905808981051400;
            q = q * v + 11.93179043337747424406973000;
            q = q * v + 10.81567043691618587425845000;
            q = q * v + 5.15642678893220502741624900;
        } else {
            p =         3.52031351611690210471836400e-14;
            p = p * v + 1.29015046760974146943988600e-10;
            p = p * v + 6.20558350620784271727934500e-8;
            p = p * v + 9.15893278788214436447190500e-6;
            p = p * v + 0.00053744013696382084534830020;
            p = p * v + 0.01394912091056901348391011000;
            p = p * v + 0.16549242452273177206985010000;
            p = p * v + 0.87667479370384158573122230000;
            p = p * v + 1.88468136100870139998674800000;
            p = p * v + 1.25236124398308722407455500000;
            q =         2.19276488942086051209795200e-12;
            q = q * v + 2.38894462713794020396471400e-9;
            q = q * v + 6.09142515484072031801978200e-7;
            q = q * v + 0.00005611957534780981729681127;
            q = q * v + 0.00221028996511963578495379800;
            q = q * v + 0.03993946264733258917755805000;
            q = q * v + 0.33620298562326883578626470000;
            q = q * v + 1.27819116349780849070965200000;
            q = q * v + 2.00077171403615841498062800000;
        }
        p *= copysign(v, ushift);
    } else {
        return -CUDART_SQRT_TWO * erfcinv(u + u);
    }
}
q = q * v + 1.0;
return p / q;
}
```

References

[1] Acklam, P. J. (2003) An algorithm for computing the inverse normal cumulative distribution function [online]. URL: http://home.online.no/~pjacklam/notes/invnorm/. Accessed on 7 January 2009.

[2] Abramowitz, M. & Stegun, I. A. (1975) *Handbook of Mathematical Functions*, Dover, New York.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press

[3] Albanese, C., Ziegler, G., Sayers, D. & Giles, M. (November 2008) Presentations of workshop on GPU computing in finance, King's College London [online]. Workshop home page URL: http://www.level3finance.com/gpuworkshop.html. Accessed on 8 January 2009.
[4] Fergusson, K. & Platen, E. (March 2006) On the distributional characterization of daily log-returns of a world stock index. *Appl. Math. Finance* **13**(1), 19–38.
[5] Gilchrist, W. (2000) *Statistical Modelling with Quantile Functions*, CRC Press, Boca Raton, FL.
[6] Giles, M. B. (2011) Approximating the erfinv function. In: Wen-mei W. Hwu (editor), *GPU Computing Gems*, Vol. 2, Morgan Kaufmann, Burlington, MA, pp. 109–116.
[7] Glasserman, P. (2004) *Monte Carlo Methods in Financial Engineering*, Springer, New York, NY.
[8] Hill, G. W. & Davis, A. W. (1968) Generalized asymptotic expansions of Cornish-Fisher type. *Ann. Math. Stat*. **39**(4), 1264–1273.
[9] Mikosch, T. (2006) Copulas: Tales and facts. *Extremes* **9**, 3–20.
[10] Moro, B. (1995) The full monte. *RISK* **8**(February), 57–58.
[11] Nvidia C. (2013) *CUDA C Programming Guide* [online]. URL: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Accessed on 7 October 2013.
[12] Nvidia C. (2013) *CUDA Parallel Computing* [online]. URL: http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html. Accessed on 7 October 2013.
[13] Ralston, A. & Rabinowitz, P. (1987) *First Course in Numerical Analysis*, McGraw-Hill, New York, NY.
[14] Shaw, W. T. (2006) Sampling Student's T distribution – use of the inverse cumulative distribution function. *J. Comput. Finance* **9**(4), 37–73.
[15] Shaw, W. T. (2007) 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 Wichura's AS241 [online]. URL: http://www.mth.kcl.ac.uk/~shaww/web_page/papers/NormalQuantile1.pdf. Accessed on 15 February 2009.
[16] Shaw, W. T. & Brickman, N. (2009) [online]. URL: http://arxiv.org/abs/0901.0638v3. Accessed on 2 December 2011.
[17] Shaw, W. T. & Buckley, I. R. C. (March 2007) 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* [online]. URL: http://arxiv.org/abs/0901.0434v1. Accessed on 10 December 2008.
[18] Steinbrecher, G. & Shaw, W. T. (2008) Quantile mechanics. *Eur. J. Appl. Math.* **19**(2), 87–112.
[19] Wichura, M. J. (1988) Algorithm AS 241: The percentage points of the normal distribution. *Appl. Stat*. **37**, 477–484.
[20] Wikipedia. (2008) "Quantile function" [online]. URL: http://en.wikipedia.org/wiki/Quantile_function. Accessed on 5 November 2008.
[21] Wikipedia. (2009) "Probit" [online]. URL: http://en.wikipedia.org/wiki/Probit. Accessed on 6 January 2009.

https://doi.org/10.1017/S0956792513000417 Published online by Cambridge University Press