SIAM J. SCI. COMPUT.
Vol. 37, No. 1, pp. C122–C141
© 2015 Society for Industrial and Applied Mathematics

EFFICIENT AND ACCURATE PARALLEL INVERSION OF THE GAMMA DISTRIBUTION*

THOMAS LUU†

Abstract. A method for parallel inversion of the gamma distribution is described. This is very desirable for random number generation in Monte Carlo simulations where gamma variates are required. Let α be a fixed but arbitrary positive real number. Explicitly, given a list of uniformly distributed random numbers our algorithm applies the quantile function (inverse CDF) of the gamma distribution with shape parameter α to each element. The result is, therefore, a list of random numbers distributed according to the said distribution. The output of our algorithm has accuracy close to a choice of single- or double-precision machine epsilon. Inversion of the gamma distribution is traditionally accomplished using some form of root finding. This is known to be computationally expensive. Our algorithm departs from this paradigm by using an initialization phase to construct, on the fly, a piecewise Chebyshev polynomial approximation to a transformation function, which can be evaluated very quickly during variate generation. The Chebyshev polynomials are high order, for good accuracy, and generated via recurrence relations derived from nonlinear second order ODEs. A novelty of our approach is that the same change of variable is applied to each uniform random number before evaluating the transformation function. This is particularly amenable to implementation on SIMD architectures, whose performance is sensitive to frequently diverging execution flows due to conditional statements (branch divergence). We show the performance of a CUDA GPU implementation of our algorithm (called Quantus) is within an order of magnitude of the time to compute the normal quantile function.

Key words. gamma distribution, quantile function, inverse CDF, parallel inversion, GPU, CUDA, quantile mechanics, Monte Carlo, copula, Quantus

AMS subject classifications. 65C10, 65C05, 65D25, 65L05, 65Y05

DOI. 10.1137/14095875X

1. Introduction.

The gamma distribution is used in physical and financial modeling. Rainfall [15] and insurance claims [5, p. 43] are two examples. The most direct way to generate a gamma—and indeed any other nonuniform—variate is by inversion. This requires the inverse cumulative distribution function (CDF), also known as the quantile function, of the distribution in question to be computable. Let

$$F_\alpha(x) = \frac{1}{\Gamma(\alpha)} \int_0^x t^{\alpha-1} e^{-t} \, \mathrm{d}t = \frac{\gamma(\alpha, x)}{\Gamma(\alpha)} \tag{1.1}$$

be the CDF of the gamma distribution with shape parameter $\alpha > 0$ and unit scale parameter. Now let $q_\alpha = F_\alpha^{-1}$ be the functional inverse of the gamma CDF. If U is a standard uniform random variable, then

$$X = q_\alpha(U) \tag{1.2}$$

*Submitted to the journal's Software and High-Performance Computing section February 26, 2014; accepted for publication (in revised form) December 3, 2014; published electronically February 24, 2015. The author's work was supported by the Industrial Mathematics Knowledge Transfer Network (KTN), the Engineering and Physical Sciences Research Council (EPSRC), and the Numerical Algorithms Group (NAG) under a KTN–EPSRC CASE award.
http://www.siam.org/journals/sisc/37-1/95875.html
†Department of Mathematics, University College London, Gower Street, London, WC1E 6BT, UK (t.luu@ucl.ac.uk).

has a gamma distribution with shape α and unit scale.[1] The gamma quantile function q_α clearly does not have a closed-form expression, so numerical methods must be used to compute it. Resorting to root finding is very much commonplace. Evaluating F_α one or more times for this is known to be computationally expensive, rendering inversion impractical for applications such as live, real-time Monte Carlo simulation. Moreover, the derivative of F_α, the gamma probability density function (PDF)

$$f_\alpha(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_\alpha(x) = \frac{x^{\alpha-1}e^{-x}}{\Gamma(\alpha)}, \tag{1.3}$$

becomes very small in the right tail. This makes root finding in this region problematic.

In reality, generating gamma variates by inversion is usually forsaken for more computationally feasible methods. The most popular method is probably rejection. A simple and efficient rejection-based algorithm for generating gamma variates is given in [18]. However, the rejection method has several important disadvantages. The inversion method does not have any of these, which is why it is regarded as the best choice for simulation. The advantage of inversion is that uniform variates are monotonically mapped to variates of the nonuniform distribution. This preserves the underlying properties of the uniform variates, which is beneficial for a number of applications, especially in modern computational finance. Copula and quasi-Monte Carlo methods are pertinent examples, and they are significantly easier to use with inversion (see, e.g., [16, p. 46]). Inversion is also well suited to variance reduction techniques, e.g., common random numbers and antithetic variates. Furthermore, with inversion, small distribution parameter perturbations cause small changes in the produced variates. This effect is useful for sensitivity analysis. Contrast this to the rejection method, where small parameter perturbations can cause large changes in the produced variates.[2] The only real disadvantage of gamma variate generation via inversion is the computational speed of existing methods. If this can be significantly improved, the benefits of inversion can be realized in practice.

For some time, the venerable Box–Muller method [7]—or the polar variant [17] of it—was the default choice for generating normal (Gaussian) variates, because it is very computationally efficient. However, nowadays, inversion of the normal CDF,

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}t^2}\,\mathrm{d}t = \frac{1}{2}\mathrm{erfc}\left(-\frac{x}{\sqrt{2}}\right), \tag{1.4}$$

is fast enough and the advantages of doing so far outweigh the negligible loss of speed. The normal quantile function Φ^{-1} is a standard inclusion in most numerical libraries, including Intel's Math Kernel Library (MKL), NAG's numerical libraries, Boost's C++ Math Toolkit, and NVIDIA's CUDA Math Library. Standalone open-source implementations of Φ^{-1} are also freely available (e.g., [30] or [26]).[3] The normal

[1] The gamma distribution has a scale parameter $\beta > 0$, such that if $X \sim \Gamma(\alpha, 1)$, then $\beta X \sim \Gamma(\alpha, \beta)$. The gamma CDF with general scale parameter is as per (1.1) but x is replaced with x/β. We can thus always assume $\beta = 1$.

[2] This is because, even after a slight parameter perturbation, a previously accepted variate could be rejected. When this occurs, the subsequently generated variates will be completely different from before.

[3] The normal quantile function $\Phi^{-1}(u)$ is often given in the form $\sqrt{2}\ \mathrm{erf}^{-1}(2u-1)$. It should be noted that direct implementation of this, using the inverse error function, will result in loss of precision for small u. One should instead use the equivalent form $-\sqrt{2}\ \mathrm{erfc}^{-1}(2u)$ or an algorithm specially designed for computing Φ^{-1}.

distribution does not possess a *shape parameter*—that is, a parameter which neither shifts nor scales the distribution. This means that a one-off polynomial or rational minimax approximation can be constructed and verified offline, so most library and standalone implementations of Φ^{-1} use such approximations. The generation of normal variates by inversion is, therefore, a straightforward matter.

Things are quite different for distributions with one or more shape parameters and no closed-form CDF. For example, out of the aforementioned numerical libraries, only NAG and Boost offer a gamma quantile function implementation. However, they are both based on root finding, so they are more than an order of magnitude slower than evaluating the normal quantile function. Being within an order of magnitude would be a more ideal situation, making inversion tractable in practice. However, the situation is even more challenging on parallel architectures like GPUs.

1.1. GPUs and other many-core architectures. Fast, parallel random number generation is desirable, because simulations are getting larger, and to take advantage of emerging parallel architectures, such as graphics processing units (GPUs) and field-programmable gate arrays (FPGAs). Several popular uniform pseudo- and quasi-random number generators (RNGs) were implemented and evaluated on the GPU in [8]. We build on this work by enabling efficient parallel gamma variate generation by inversion.

GPUs and other single instruction, multiple data (SIMD) architectures are particularly sensitive to divergent flows of execution due to conditional statements in code. When branching occurs, each branch taken is effectively serialized, so the execution time of a conditional statement is roughly equal to the sum of each branch taken. *Branch divergence* is said to occur in this case. If the branches are computationally expensive, valuable parallelism is lost. (Nested conditional statements obviously compound the problem.) Peak performance is hence achieved with straight-line code that has only one execution path. Branch divergence can otherwise significantly affect parallel performance. It is a particular issue for traditional methods of quantile function computation. The tails of distributions are usually managed separately, which causes branch divergence.

There is existing work looking at GPU-optimized algorithms for the normal quantile function in [26]. However, we are not aware of any such research or commercial solutions for the gamma distribution. In this paper, we will concentrate on the case of generating large quantities of identically distributed gamma variates at a time. When the shape parameter α is fixed, it opens the possibility of precomputing a fast approximation to q_α. We remodel the gamma quantile function for optimal evaluation on GPUs and other parallel architectures by using ideas from [26]. More specifically, we leverage *quantile mechanics*, the differential approach to quantile functions, with tailored changes of variable to facilitate fast and accurate computation of the gamma quantile function. An open-source implementation of our new algorithm is publicly available.[4]

2. Quantile mechanics. The quantile function can be viewed as a special function in its own right. A lot of special functions are, of course, solutions to differential equations or integrals. In [27], quantile functions were characterized by differential equations, and analytic series expansions of the quantile functions for the normal, Student t, beta, and gamma distributions were derived. We will review the basic ideas of quantile mechanics now.

[4]https://github.com/thomasluu/quantus

Let $F : \mathbf{R} \to (0, 1)$ be the CDF of a continuous random variable. By the inverse function theorem, for $0 < u < 1$, the corresponding quantile function $q(u) = F^{-1}(u)$ must satisfy the ordinary differential equation (ODE)

$$\frac{\mathrm{d}q(u)}{\mathrm{d}u} = \frac{1}{f(q(u))}, \tag{2.1}$$

where f is the PDF of the random variable. In [27], (2.1) is called the *first order quantile ODE*. This ODE was used in [21] to derive series solutions of the hyperbolic, variance-gamma and generalized inverse Gaussian quantile functions. The first order quantile ODE can actually be found in [29], where it is solved numerically for the normal, exponential, Cauchy, and gamma distributions. However, they encountered accuracy problems with the gamma distribution when its shape parameter was near zero. We will treat this with a change of variable.

Differentiating (2.1) with respect to u results in

$$\frac{\mathrm{d}^2 q}{\mathrm{d}u^2} = H_f(q) \left(\frac{\mathrm{d}q}{\mathrm{d}u} \right)^2, \tag{2.2}$$

where

$$H_f(x) = -\frac{\mathrm{d}}{\mathrm{d}x} \log f(x), \tag{2.3}$$

and the explicit dependence on u is suppressed for brevity. In [27], (2.2) is called the *second order quantile ODE*. $H(x)$ is a simple rational function for the Pearson system of distributions, so analytic series solutions can be found in these cases. Recalling the normal PDF is

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}, \tag{2.4}$$

H for the normal distribution is easily found to be

$$H_\phi(x) = x. \tag{2.5}$$

H for the gamma distribution is

$$H_{f_\alpha}(x) = \frac{1 + x - \alpha}{x}. \tag{2.6}$$

2.1. Variate recycling. The idea of using quantile mechanics to convert samples from one distribution to another was initiated in [26]. Let G be the CDF of an *intermediary* distribution and let q be the quantile function of the *target* distribution (with PDF f). If v is a sample from the intermediary distribution,

$$Q(v) = q(G(v)) \tag{2.7}$$

maps v to the target distribution. The sample v is *recycled* into one from the target distribution. Note this mapping is monotonic, so if v is generated by inversion (using G^{-1}), the underlying properties of the uniform variates are preserved. If Q can be approximated with an approximation that covers a long enough range, branch divergence will be alleviated. In the interest of speed, G^{-1} should preferably be easy to compute.

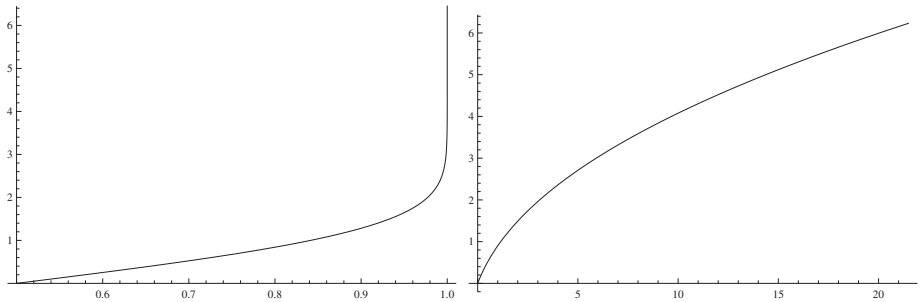

FIG. 1. *Left: the normal quantile function* $\Phi^{-1}(u)$ *for* $1/2 \le u < 1$. *Right: the Laplace to normal transformation* $\Phi^{-1}\left(1 - e^{-v}/2\right)$ *for* $0 \le v \le 22$.

Variate recycling was used in [26] to create GPU-optimized algorithms for Φ^{-1}. A Laplace double-exponential intermediary distribution was used, so the relevant mapping for $1/2 \le u < 1$ is

$$Q(v) = \Phi^{-1}\left(1 - e^{-v}/2\right), \tag{2.8}$$

where

$$v = -\log\left[2\left(1 - u\right)\right]. \tag{2.9}$$

Symmetry of Φ^{-1} allows $0 < u < 1/2$ to be managed trivially. Figure 1 shows a plot of $Q(v)$ for $0 \le v \le 22$. This range is equivalent to $1/2 \le u \le 1 - e^{-22}/2 \approx 1 - 1.39 \times 10^{-10}$. Figure 1 also shows a plot of Φ^{-1} in standard coordinates for the same range. A single minimax rational approximation to Q can feasibly cover $[0, 22]$, due to the exponential change of variable. This led to branch-free algorithms for Φ^{-1} optimized for GPU execution.

Let us consider a change of independent variable in the second order quantile ODE, (2.2). Letting $v = G^{-1}(u)$ and writing $Q(v) = q(u)$, some differentiation and simplification gives

$$\frac{\mathrm{d}^2 Q}{\mathrm{d}v^2} + H_g(v)\frac{\mathrm{d}Q}{\mathrm{d}v} = H_f(Q)\left(\frac{\mathrm{d}Q}{\mathrm{d}v}\right)^2, \tag{2.10}$$

where

$$H_g(x) = -\frac{\mathrm{d}}{\mathrm{d}x}\log g(x) \tag{2.11}$$

with g the PDF of the intermediary distribution. In [26], (2.10) is called the *recycling ODE*.

This work will follow and extend [26] to create a GPU-optimized algorithm for the gamma quantile function. We propose that quantile mechanics with variate recycling is an effective way to approximate this function.

3. Approximating the gamma quantile function. Figure 2 shows the gamma quantile function q_α for various α. By definition, we have $q_\alpha(0) = 0$ and $q_\alpha(1) = \infty$ for all α.

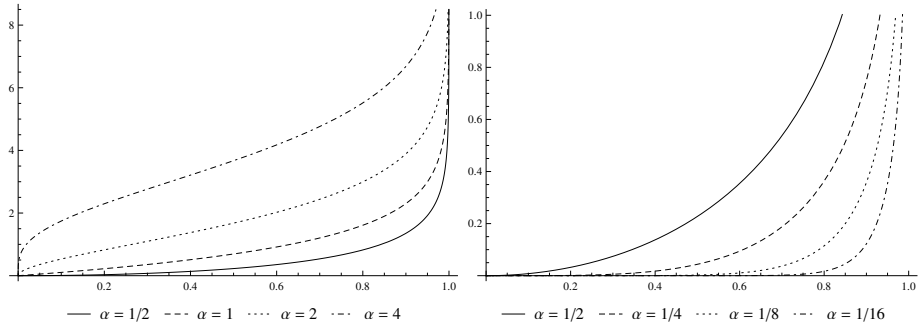

FIG. 2. *The gamma quantile function* $q_\alpha(u)$ *for* $0 \le u < 1$ *and various* α.

The exponential and χ^2 distributions are special cases of the gamma distribution.[5] The quantile function of the exponential distribution with rate parameter $\lambda > 0$ is

$$\frac{1}{\lambda} q_1(u) = -\frac{1}{\lambda} \log(1-u), \tag{3.1}$$

so $q_1(u)$ is, therefore, trivial to implement.[6] The quantile function of the χ^2 distribution with $\nu > 0$ degrees of freedom is

$$2\, q_{\nu/2}(u). \tag{3.2}$$

If $\nu = 1$,

$$q_{1/2}(u) = [\mathrm{erf}^{-1}(u)]^2, \tag{3.3}$$

so $q_{1/2}(u)$ is another case that is straightforward to implement.

In general, as mentioned in the introduction, q_α is typically computed by numerical root finding. This is the approach taken by the Boost C++ Math Toolkit [6], which uses [12]. NAG [22] and R [24] instead use [4] to first compute the χ^2 quantile function and then transform the result into the equivalent gamma variate. The most recent (at the time of writing) algorithm for q_α in the literature is in [13]. The authors claim their inversion algorithm, which uses Newton root finding, is more accurate than the one in [12].

3.1. Floating-point considerations. Here we consider only IEEE single- and double-precision[7] floating-point formats henceforth, so there are some important simplifications that can be made when computing q_α. Let us fix some notation. Let, respectively,

$$\begin{aligned} \epsilon_s &= 2^{-24} &&\approx 5.96 \times 10^{-8}, \\ \min_s &= 2^{-126} &&\approx 1.18 \times 10^{-38}, \\ \min'_s &= 2^{-149} &&\approx 1.40 \times 10^{-45} \end{aligned} \tag{3.4}$$

[5]While not a special case, the beta distribution is related to the gamma distribution. If $X_1 \sim \Gamma(\alpha, 1)$ and $X_2 \sim \Gamma(\beta, 1)$ are independent, $X_1/(X_1 + X_2) \sim \mathrm{B}(\alpha, \beta)$. So if one has a fast gamma variate generator, a fast beta variate generator is always available.

[6]Care should be taken to ensure accuracy for u less than machine epsilon. The C function `log1p(x)` computes $\log(1 + x)$ accurately for small x values.

[7]Talking about single-precision may seem quaint. However, single-precision computations are appreciably faster than double-precision on modern parallel architectures. Single-precision is consequently coming back into fashion.

be machine epsilon, the smallest positive (normal) number, and the smallest positive subnormal number for IEEE single-precision. Also let

$$\begin{aligned} \epsilon_d &= 2^{-53} &&\approx 1.11 \times 10^{-16}, \\ \min_d &= 2^{-1022} &&\approx 2.23 \times 10^{-308}, \\ \min'_d &= 2^{-1074} &&\approx 4.94 \times 10^{-324} \end{aligned} \tag{3.5}$$

be the equivalent values for IEEE double-precision. Various schools of thought exist as to how accurate quantile functions—and random numbers in general—need to be. Some may take the position that the number of significant figures obtained is perhaps unimportant, due to the numerous uncertainties elsewhere in problems. We will instead aim to eliminate as much uncertainty as possible from our numerical considerations and characterize the associated errors.

3.1.1. Uniform input. Pseudo- and quasi-RNGs work with unsigned integer data types—typically 32- or, to a lesser extent, 64-bit integers. Most RNGs, therefore, output random integral values between 0 and either 2^{32} or 2^{64}. For example, the Mersenne Twister MT19937 RNG [19] outputs 32-bit integers. A floating-point multiplication by 2^{-32} or 2^{-64} gives a random uniform number between 0 and 1 that can be fed into q_α.

Since we are focusing on simulation applications, we will assume a 32- or 64-bit RNG is the source of randomness for nonuniform variate generation. The smallest (nonzero) uniform number we encounter is thus $2^{-32} \approx 2.33 \times 10^{-10}$ or $2^{-64} \approx 5.42 \times 10^{-20}$. It should be noted that these numbers are larger than both $\min_s$ and $\min_d$. Nevertheless, for completeness, we will also show that our method—very naturally—accommodates for input down to $\min'_d$. 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 sought, the largest number will be limited to $1 - \epsilon_s$ in either case. If double-precision numbers are requested, the largest number will be $1 - 2^{-32}$ or $1 - \epsilon_d$ depending on the RNG.

3.1.2. Nonuniform output. On the other side of the coin, there are occasions where the true result of $q_\alpha(u)$ is outside the range of representable floating-point numbers. The most obvious case is overflow. Here it is logical to return infinity. The other case, which is not so clear-cut, is what to do in the event of gradual underflow. Can we simply ignore this and return zero, or should we return a subnormal number that will be imprecise to some degree?

Gradual underflow might occur for the gamma distribution with a very small α parameter. Let us consider the case of $\alpha = 1/100$. The gamma quantile $q_{1/100}(37/100)$ is $3.741497614 \times 10^{-44}$ to ten significant figures. This value is less than $\min_s$, but greater than $\min'_s$. Therefore, $q_{1/100}(37/100)$ cannot be fully represented in single-precision. The nearest subnormal number is 3.783506×10^{-44}, which has a relative error of 0.0112277. This begs the question of how to measure the accuracy of a subnormal result.

The finite range of floating-point numbers means the uniform input range of concern to us can sometimes be shorter than usual. For example, if $\alpha = 1/100$, the gamma CDF evaluated at $x = \min'_s$ is 0.3580441447 to ten significant figures, so zero can (and should) be returned for all u less than this value when working in single-precision.

TABLE 1

Upper u-limits of the approximation $[u\Gamma(\alpha+1)]^{1/\alpha}$ to the gamma quantile function $q_\alpha(u)$ for various α in single- and double-precision. Asterisked values are not representable in the respective floating-point number formats and would be flushed to zero. This is the same for the double-asterisked value, but it would be rounded to one.

	$u_\alpha(\epsilon)$	
α	Single-precision ($\epsilon = \epsilon_s$)	Double-precision ($\epsilon = \epsilon_d$)
10^{-9}	9.9999998×10^{-1}**	$9.999999638404157 \times 10^{-1}$
10^{-8}	9.9999984×10^{-1}	$9.999996384042162 \times 10^{-1}$
10^{-7}	9.9999839×10^{-1}	$9.999963840480389 \times 10^{-1}$
10^{-6}	9.9998394×10^{-1}	$9.999638410680227 \times 10^{-1}$
10^{-5}	9.9983943×10^{-1}	$9.996384694366355 \times 10^{-1}$
10^{-4}	9.9839545×10^{-1}	$9.963905630200882 \times 10^{-1}$
10^{-3}	9.8406912×10^{-1}	$9.644855708647861 \times 10^{-1}$
10^{-2}	8.5157729×10^{-1}	$6.965068173834265 \times 10^{-1}$
10^{-1}	1.9915322×10^{-1}	$2.668089225353158 \times 10^{-2}$
10^{0}	5.9604647×10^{-8}	$1.110223024625157 \times 10^{-16}$
10^{1}	$1.5596895 \times 10^{-79}$*	$7.840418869435904 \times 10^{-167}$
10^{2}	$3.6141656 \times 10^{-881}$*	$3.724082781223321 \times 10^{-1754}$*

3.2. Analytical estimates. The first order gamma quantile ODE is

$$\frac{\mathrm{d}q_\alpha}{\mathrm{d}u} = e^{q_\alpha}\Gamma(\alpha)\, q_\alpha^{1-\alpha}. \tag{3.6}$$

As $u \to 0$,

$$\frac{\mathrm{d}q_\alpha}{\mathrm{d}u} \sim \Gamma(\alpha)\, q_\alpha^{1-\alpha}. \tag{3.7}$$

Treating (3.7) as an exact ODE, it has the solution $q_\alpha(u) = [u\Gamma(\alpha+1)]^{1/\alpha}$. This motivates the approximation

$$x_\alpha(u) = [u\Gamma(\alpha+1)]^{1/\alpha} \tag{3.8}$$

for $q_\alpha(u)$. The derivative of $F_\alpha(x_\alpha(u))$ is just $\exp(-x_\alpha(u))$. Consequently, the difference of this quantity from one is a measure of how accurate $x_\alpha(u)$ is to $q_\alpha(u)$. This would be useful for q_α algorithms based on root finding, because the expensive iterative process can potentially be short-circuited. Solving $\epsilon = 1 - \exp(-x_\alpha(u))$ for u allows one to find, for a particular α and tolerance ϵ, the range of u that can be managed by $x_\alpha(u)$. Let

$$u_\alpha(\epsilon) = \frac{[-\log(1-\epsilon)]^\alpha}{\Gamma(1+\alpha)} \tag{3.9}$$

be the upper limit of the approximation x_α for ϵ. Table 1 gives u_α for various α in single- and double-precision. An analogous procedure can be used to derive ap-

proximations for certain distributions, such as the noncentral χ^2 distribution,[8] which features in the Cox–Ingersoll–Ross and Heston models in mathematical finance.

The form of $x_\alpha(u)$ in (3.8) is unsuitable for practical implementation for very small α, because of the resultant large power, which is problematic for u close to unity. Fortunately, this can be managed:

$$\begin{aligned} x_\alpha(u) &= [u\Gamma(\alpha+1)]^{1/\alpha} \\ &= \exp\left\{\frac{1}{\alpha}[\log u + \log\Gamma(1+\alpha)]\right\}. \end{aligned} \tag{3.10}$$

The power series of $\log\Gamma(1+z)$ for $|z|<2$ (given by equation 6.1.33 in [1, p. 256]) is especially useful here, for α near zero. The series converges rapidly for $|z|<1/2$ and the obvious implementation of $\log\Gamma(1+z)$ is absolutely fine for z outside this range.

We will rely on (3.10) when it will yield a sufficiently accurate result. Section 3.3 will give details of approximations for other inputs.

3.3. Numerical approximations. The gamma quantile function is neither analytic at zero nor easy to approximate without treating the tail separately. We can actually kill two birds with one stone by considering a transformation of the gamma distribution. This indirect route, which turns out to be fruitful, does not appear to have been explored in the literature. Let $X \sim \Gamma(\alpha, 1)$. We will consider $Y \sim \log X$. Since the range of X is $[0,\infty)$, the range of Y is the entire real line. Y follows the exp-gamma distribution with shape α, unit scale, and location zero. The CDF of Y is

$$\hat{F}_\alpha(x) = F_\alpha(e^x), \tag{3.11}$$

which implies its inverse is

$$\hat{q}_\alpha(u) = \log q_\alpha(u), \tag{3.12}$$

so if we can approximate $\hat{q}_\alpha(u)$, then $q_\alpha(u)$ follows immediately. The PDF of Y is

$$\hat{f}_\alpha(x) = \frac{e^{x\alpha - e^x}}{\Gamma(\alpha)}, \tag{3.13}$$

so

$$H_{\hat{f}_\alpha}(x) = e^x - \alpha. \tag{3.14}$$

In contrast to $H_{f_\alpha}(x)$, which diverges as $x \to 0$, $H_{\hat{f}_\alpha}(x)$ clearly converges as $x \to -\infty$.

We observed that the density of Y is remarkably similar to the skew-normal distribution[9] [2] with shape $-\alpha^{-1}$. This suggests that the skew-normal distribution would

[8] The noncentral χ^2 distribution with $\nu > 0$ degrees of freedom and noncentrality parameter $\lambda \geq 0$ has density $f_{\nu,\lambda}(x) = 2^{-\nu/2}e^{-\frac{1}{2}(\lambda+x)}x^{\nu/2-1}{}_0\tilde{F}_1\left(;\frac{\nu}{2};\frac{x\lambda}{4}\right)$ on $x \in [0,\infty)$. There are a handful of closed-form approximations for the noncentral χ^2 quantile function, including one due to Pearson [23]. However, the approximations typically fail for small enough inputs—Pearson's approximation can yield a negative result and the approximation from [25] can give a complex-valued result. These are known issues. They are especially problematic for refinement via root finding, since these invalid values obviously cannot be fed into the noncentral χ^2 CDF. Since the derivative of the corresponding inverse CDF, say, $q_{\nu,\lambda}(u)$ is asymptotic to $2^{\nu/2}e^{\frac{1}{2}\lambda}q_{\nu,\lambda}(u)^{1-\nu/2}\Gamma(\nu/2)$ as $u \to 0$, this suggests the formula $x_{\nu,\lambda}(u) = [2^{\nu/2}e^{\lambda/2}u\Gamma(1+\nu/2)]^{2/\nu}$ would be a more useful guess for when $q_{\nu,\lambda}(u)$ is expected to be very small. Note that when $\lambda = 0$ (the central χ^2 special case), the formula matches (3.8) scaled by two with $\alpha = \nu/2$. $1 - f_{\nu,\lambda}(x_{\nu,\lambda}(u))\frac{2x_{\nu,\lambda}(u)}{u\nu}$ gives the precision of the approximation. Wolfram Mathematica 10.0.0 and Boost 1.55.0 use $x_{\nu,\lambda}$ due to private communication with this author.

[9] The PDF of the skew-normal distribution with shape parameter $\alpha \in \mathbf{R}$ is $2\phi(x)\Phi(\alpha x)$. The quantile function of this distribution is, therefore, easy to compute for $\alpha \in \{-1, 0, 1\}$.

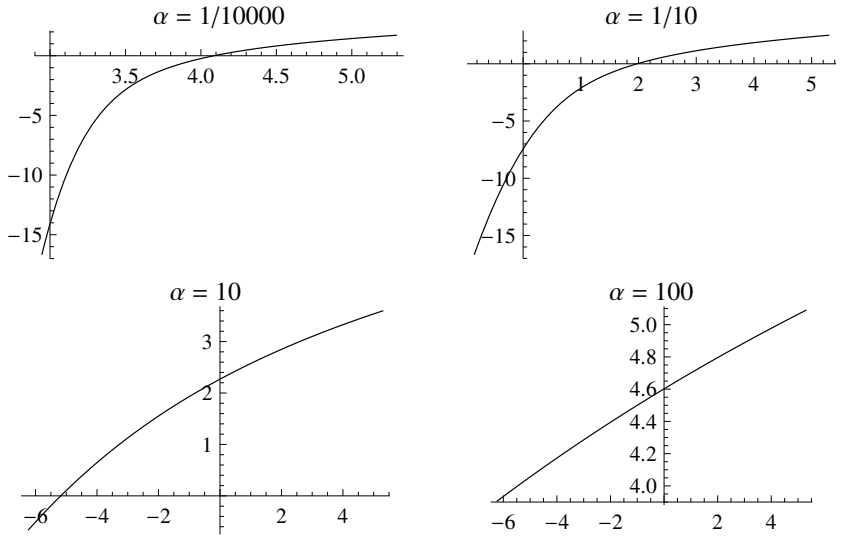

FIG. 3. *The normal to exp-gamma transformation* $\hat{q}_\alpha \circ \Phi$ *for various* α. *The ranges in the top two plots are truncated due to the analytic approximation from section* 3.2.

be useful for variate recycling. Unfortunately, this distribution's quantile function is only straightforward to evaluate for a very limited number of cases—none of which that are of use to us, but all is not lost. The left tail of the skew-normal distribution with negative shape (skew) "decreases at the same rate as the normal distribution tail" [3, pp. 52–53]. This points to the normal distribution as being a good alternative for variate recycling. We have

$$\begin{aligned} v_0 &= \Phi^{-1}(2^{-32}) \approx -6.23 \text{ (for 32-bit RNGs)}, \\ v_0 &= \Phi^{-1}(2^{-64}) \approx -9.08 \text{ (64-bit RNGs)} \end{aligned} \tag{3.15}$$

and

$$\begin{aligned} \Phi^{-1}(1-\epsilon_s) &\approx 5.29 \text{ (for single-precision)}, \\ \Phi^{-1}(1-2^{-32}) &\approx 6.23 \text{ (double-precision 32-bit RNGs)}, \\ \Phi^{-1}(1-\epsilon_d) &\approx 8.21 \text{ (double-precision 64-bit RNGs)}, \end{aligned} \tag{3.16}$$

which gives an idea of the ranges we typically have to approximate over. However, $q_\alpha(u)$ with a very small α parameter skirts the u-axis and suddenly goes to infinity. As shown in section 3.2, the start of the range to approximate over can sometimes be brought in. Figure 3 shows the appropriateness of the normal to exp-gamma transformation

$$Q(v) = \hat{q}_\alpha(\Phi(v)) \tag{3.17}$$

for various α. Note that the transformation becomes more linear as α increases. This is because the exp-gamma distribution converges to the normal distribution as $\alpha \to \infty$.

Combining (2.10) with (2.5) and (3.14), the normal to exp-gamma quantile recy-

cling ODE and initial conditions are

$$\begin{aligned} \frac{\mathrm{d}^2 Q}{\mathrm{d}v^2} &= \frac{\mathrm{d}Q}{\mathrm{d}v}\left[\left(e^Q - \alpha\right)\frac{\mathrm{d}Q}{\mathrm{d}v} - v\right], \\ Q(v_i) &= \hat{q}_\alpha(\Phi(v_i)), \\ Q'(v_i) &= \frac{\phi(v_i)}{\hat{f}_\alpha(Q(v_i))}, \end{aligned} \tag{3.18}$$

or equivalently

$$\begin{aligned} Q_0' &= Q_1, \\ Q_1' &= Q_1\left[\left(e^{Q_0} - \alpha\right)Q_1 - v\right], \end{aligned} \tag{3.19}$$

with $Q_0(v_i) = Q(v_i)$ and $Q_1(v_i) = Q'(v_i)$. The first order system can be represented as

$$\begin{aligned} V_1 &= e^{Q_0}, \\ V_2 &= V_1 - \alpha, \\ V_3 &= V_2 \cdot Q_1, \\ V_4 &= V_3 - v, \\ V_5 &= Q_1 \cdot V_4, \\ (Q_0)_1 &= Q_1, \\ (Q_1)_1 &= V_5, \end{aligned} \tag{3.20}$$

where $(X)_k$ denotes the kth coefficient in the Taylor series of $X(v)$. So

$$\begin{aligned} (V_1)_k &= \begin{cases} e^{(Q_0)_0} & \text{if } k = 0, \\ \sum_{j=0}^{k-1}(1 - j/k)(V_1)_j(Q_0)_{k-j} & \text{otherwise,} \end{cases} \\ (V_2)_k &= (V_1)_k - \delta_k\alpha, \\ (V_3)_k &= \sum_{j=0}^{k}(V_2)_j(Q_1)_{k-j}, \\ (V_4)_k &= (V_3)_k - (v)_k, \\ (V_5)_k &= \sum_{j=0}^{k}(Q_1)_j(V_4)_{k-j}, \\ (Q_0)_{k>0} &= \frac{1}{k}(Q_1)_{k-1}, \\ (Q_1)_{k>0} &= \frac{1}{k}(V_5)_{k-1}, \end{aligned} \tag{3.21}$$

with

$$\begin{aligned} (v)_k &= \begin{cases} v & \text{if } k = 0, \\ 1 & \text{if } k = 1, \\ 0 & \text{otherwise,} \end{cases} \\ (Q_0)_0 &= Q(v), \\ (Q_1)_0 &= Q'(v), \end{aligned} \tag{3.22}$$

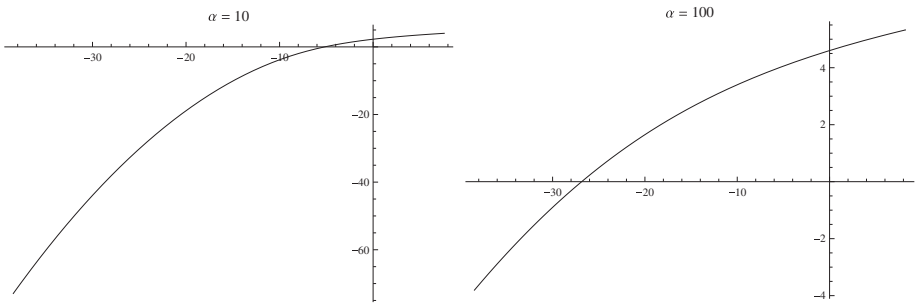

FIG. 4. *The normal to exp-gamma transformation* $\hat{q}_\alpha \circ \Phi$ *for* $\alpha = 10$ *and* $\alpha = 100$*, applicable to the full complement of double-precision numbers.*

where we have employed the technique from [20, section 3.4] to derive the recurrence relations for the Taylor coefficients of Q about any point. (See also [10] and [21]. The latter novelly extends the method to several non-Pearson distributions, which have relatively complicated H-functions.) These recurrence relations are straightforward to implement on a computer. Moreover, for $1, 2, \ldots, n$ the total number of elementary operations required to compute $(Q_0)_n$ is of order n^2 (assuming $(Q_0)_1, (Q_0)_2, \ldots, (Q_0)_{n-1}$, and the previously computed auxiliary variables are cached so they can be reused). The complexity of evaluating $(Q_0)_k$ is thus relatively low. The accuracy of the summations corresponding to V_1 and the product auxiliary variables (V_3 and V_5) can be improved by Kahan/compensated summation (see, e.g., [14]).

Given a suitably accurate numerical approximation of $Q(v_i)$ and setting $v = v_i$ in the recurrence relations above, we can compute the coefficients of the Taylor expansion of $Q(v)$ about $v = v_i$, $(Q_0)_k$, up to an *arbitrary order*. The truncated Taylor series of $Q(v)$ about $v = v_i$ is thus

$$\sum_{k=0}^{n} (Q_0)_k (v - v_i)^k. \tag{3.23}$$

We have shown that the normal to exp-gamma transformation regularizes q_α well for uniform output from 32- and 64-bit RNGs. We have also shown that the analytic approximation in section 3.2 can handle input close to zero for small to moderate α. For other α, the normal to exp-gamma transformation can be extended right down to $\min'_d$. Figure 4 shows that this transformation is perfectly serviceable for this case.

3.3.1. Very large shape parameter values. It is well known that the gamma distribution tends to the normal distribution as $\alpha \to \infty$. Since the normal distribution approximates the gamma distribution well for large α, it is a natural candidate for variate recycling in this case. While the normal to exp-gamma transformation is theoretically suitable for very big α values, directly transforming normal variates to gamma ones is more efficient. Details for this are given here for completeness. The ranges we have to approximate over are as per (3.15) and (3.16). Figure 5 shows the appropriateness of the normal to gamma transformation

$$Q(v) = q_\alpha(\Phi(v)) \tag{3.24}$$

for various large α.

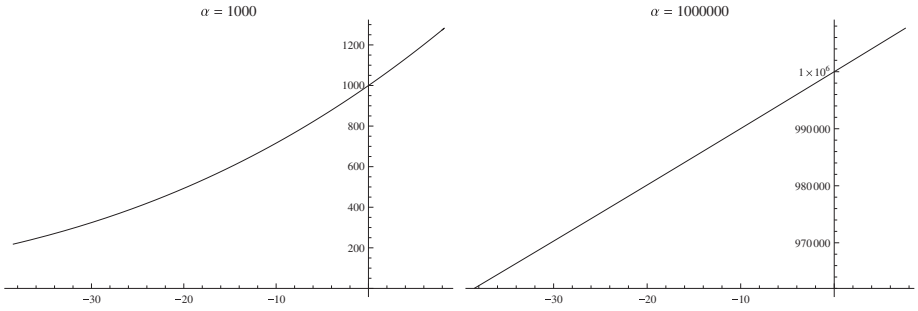

FIG. 5. *The normal to gamma transformation $q_\alpha \circ \Phi$ for $\alpha = 10^3$ and $\alpha = 10^6$, applicable to the full complement of double-precision numbers.*

The normal to gamma quantile recycling ODE and initial conditions are

$$\begin{aligned} \frac{\mathrm{d}^2 Q}{\mathrm{d}v^2} &= \frac{\mathrm{d}Q}{\mathrm{d}v}\left(\frac{1-\alpha+Q}{Q}\frac{\mathrm{d}Q}{\mathrm{d}v} - v\right), \\ Q(v_i) &= q_\alpha(\Phi(v_i)), \\ Q'(v_i) &= \frac{\phi(v_i)}{f_\alpha(Q(v_i))}, \end{aligned} \tag{3.25}$$

or equivalently

$$\begin{aligned} Q_0' &= Q_1, \\ Q_1' &= Q_1\left(\frac{1-\alpha+Q_0}{Q_0}Q_1 - v\right), \end{aligned} \tag{3.26}$$

with $Q_0(v_i)$ and $Q_1(v_i)$ as appropriate. This leads to

$$\begin{aligned} V_1 &= 1-\alpha+Q_0, & (V_1)_k &= \delta_k(1-\alpha) + (Q_0)_k, \\ V_2 &= V_1 \cdot Q_1, & (V_2)_k &= \sum_{j=0}^{k} (V_1)_j (Q_1)_{k-j}, \\ V_3 &= \frac{V_2}{Q_0}, & (V_3)_k &= \frac{1}{(Q_0)_0}\left[(V_2)_k - \sum_{j=1}^{k} (Q_0)_j (V_3)_{k-j}\right], \\ V_4 &= V_3 - v, & (V_4)_k &= (V_3)_k - (v)_k, \\ V_5 &= Q_1 \cdot V_4, & (V_5)_k &= \sum_{j=0}^{k} (Q_1)_j (V_4)_{k-j}, \\ (Q_0)_1 &= Q_1, & (Q_0)_{k>0} &= \frac{1}{k}(Q_1)_{k-1}, \\ (Q_1)_1 &= V_5, & (Q_1)_{k>0} &= \frac{1}{k}(V_5)_{k-1}, \end{aligned} \tag{3.27}$$

where $(v)_k$, $(Q_0)_0$, and $(Q_1)_0$ are as per (3.22).

3.4. Algorithm design. Our algorithm for q_α is split into two phases:

(i) *initialization* for a given shape parameter α; and

(ii) *generation* of gamma variates with shape α.

For the sake of simplicity, this section will ignore the normal to gamma transformation (3.24). If the transformation is used for large α values, the algorithm steps are broadly identical.

Routine 1. Initialization.

Input: The value of α, requested precision ϵ (either ϵ_s or ϵ_d), target RNG (either 32- or 64-bit).

▷ Determine relevant u-range (section 3.1.1)

1: $u_{\min} \leftarrow$ the smallest (nonzero) uniform number produced by the target RNG
2: $u_{\max} \leftarrow$ the largest (excluding 1) uniform number produced by the target RNG
3: **if** $u_{\min} < u_\alpha(\epsilon)$ **then**
4: $\quad u_{\min} \leftarrow u_\alpha(\epsilon)$ ▷ See section 3.2
5: **end if**
6: **if** $u_{\min} \geq u_{\max}$ **then**
7: $\quad x_\alpha(u)$ can be used for all $0 \leq u \leq u_{\max}$ so no further initialization is necessary.
8: **end if**

▷ Compute the smallest and largest expected inputs into the recycling function Q (3.17)

9: $v_0 \leftarrow \Phi^{-1}(u_{\min})$
10: $v_m \leftarrow \Phi^{-1}(u_{\max})$
11: An instance of the recycling ODE (3.18) is solved over $[v_0, v_m]$.

3.4.1. Initialization. Routine 1 gives the pseudocode for this phase. Our ODE solution is an approximation, which is in the form of a piecewise Taylor polynomial approximation. Supposing $[v_0, v_m]$ is partitioned as $v_0 < v_1 < \cdots < v_m$, Q is thus approximated by polynomials of the form

$$\sum_{k=0}^{n} \frac{Q^{(k)}(v_i)}{k!}(v - v_i)^k, \tag{3.28}$$

where we compute the Taylor coefficients using the recurrence relations developed in section 3.3. The task of computing the piecewise polynomial approximations can be done sequentially or in parallel, by concurrently expanding about different points. The ODE solution is basically an $(m+1) \times (n+1)$ matrix with the coefficients of the polynomial expansion about v_i in the ith row. For an arbitrary $v \in [v_0, v_m]$ the relevant row for evaluating $Q(v)$ is easily found. If $v_0 < v_1 < \cdots < v_m$ is an equidistant partition of $[v_0, v_m]$, the calculation is even simpler: the correct row is the integer part of $(v - v_0)/h$, assuming h is the distance between any two partition points. We will actually use equidistant partitions, because they are feasible due to our changes of variable. We can estimate the accuracy of an approximant about v_i by evaluating it at v_{i+1} and comparing the result with $Q(v_{i+1})$, since this is where the error will peak. The partition of $[v_0, v_m]$ and order n can be refined and adjusted so as to meet a desired tolerance.

A simple yet reliable procedure therefore exists for solving the recycling ODEs. It works by successively refining a mesh. Given an initial step size, a piecewise Taylor polynomial approximation to Q is generated. Derivatives of up to a specified maximum order can be used to iteratively form piecewise polynomials of increasing accuracy. If an approximation is found that is accurate to some tolerance, initialization is complete. If such an approximation is not found, the search process is repeated

TABLE 2
The recommended parameters for the initialization phase of our algorithm.

	Single-precision	Double-precision
Maximum order	10th	20th
Initial step size	1/4	1/8
Tolerance target	$50\epsilon_s$	$50\epsilon_d$

with the step size halved. Table 2 gives the initialization parameters that we recommend. They were found to give a good balance of performance and accuracy. Starting with step sizes with an integral power of two and expanding about points that are a whole multiple of the step size allows for more efficient polynomial selection and evaluation.

In the interest of numerical stability we would prefer to evaluate Chebyshev polynomials instead of Taylor ones. Armed with our Taylor coefficients, an efficient method to accomplish this is given in [28]. (See [21, section 8.5.2] for an application to the hyperbolic distribution.) Our Taylor polynomials can be recast into the Chebyshev form

$$\frac{c_0}{2} + \sum_{k=1}^{n} c_k T_k\left(\frac{v - v_i}{h}\right), \tag{3.29}$$

where $T_k(x)$ are the Chebyshev polynomials of the first kind, defined as

$$T_k(x) = \cos(k \arccos x), \tag{3.30}$$

and c_k are Chebyshev coefficients. These coefficients are computed as

$$c_k = \sum_{r=k}^{n} a_r \theta_{r,k}, \tag{3.31}$$

where

$$a_r = \frac{Q^{(r)}(v_i)}{r!} h^r \tag{3.32}$$

and

$$\theta_{r,k} = \begin{cases} 2^{1-r}\binom{r}{(r-k)/2} & \text{if } r-k \text{ is even,} \\ 0 & \text{otherwise,} \end{cases} \tag{3.33}$$

which can be evaluated recursively.

While our approximants are not guaranteed to preserve complete monotonicity, it is not unreasonable to expect them by and large to be monotonic. We found this to be true in practice. Also, whenever monotonicity between two consecutive variates was violated, the deviation from monotonicity was very close to machine epsilon. Such deviations are irrelevant in practice for most simulations.

3.4.2. Generation. Routine 2 gives the pseudocode for the variate generation phase. The recycling function $Q(v)$ is computed by looking up the appropriate Chebyshev polynomial and evaluating it using Clenshaw's formula [9]. It should

Routine 2. Generation.
Input: A uniform variate $u \in (0, 1)$
Output: $q_\alpha(u)$
1: **if** $u \leq u_\alpha(\epsilon)$ **then**
2: **return** $x_\alpha(u)$
3: **end if**
4: $v \leftarrow \Phi^{-1}(u)$ ▷ Evaluate change of variable
5: $y \leftarrow Q(v)$ ▷ The recycling function is computed
6: **return** $\exp(y)$ ▷ Since $Q(v) = \hat{q}_\alpha(u)$

be noted that this *does not* introduce additional branch divergence. Once the correct polynomial index is determined by each thread, the polynomials are evaluated synchronously—the instructions to do this are identical, but the coefficients may be different. The variate generation portion of the algorithm is incredibly simple, so the barrier to execution on present and future computer architectures is low.

3.5. Computational experiments. We will now demonstrate the parallel performance of our gamma quantile function algorithm. The Oxford English Dictionary says *quantile* originates from the Latin word *quantus*, which means *how great, how much*. Quantus, therefore, seems an apt name for our algorithm implementation. The performance of Quantus was evaluated on two high-end NVIDIA GPUs:

(i) a Kepler-class GeForce GTX Titan; and
(ii) a Fermi-class Tesla C2050.

The test GPUs were hosted in a system with

(i) an Intel Core i5-4670K (overclocked to 4.2 GHz); and
(ii) 8 GB of RAM.

The system was running

(i) Ubuntu Server 12.04.2 LTS with GCC 4.6.3;
(ii) NVIDIA CUDA 6.5.14; and
(iii) Boost 1.56.0.

The freely available Boost C++ Math Toolkit provides a high-quality quantile function implementation for the gamma distribution (along with several other distributions). Suitably precise initial conditions can hence be computed via Boost's quantile function implementation.

The Quantus initialization code was parallelized with OpenMP and compiled using GCC with the –O2 optimization flag. For both single- and double-precision initialization, double working precision was used. This yields recycling function approximations with accuracy close to machine epsilon. A caveat is that the initial conditions have to be accurate to target precision. This is easily achieved for single target precision. However, for double target precision this is problematic for values in the right tail. We took a brute force approach, simply computing these initial conditions with software-simulated arbitrary precision for $u > 9/10$. We used the normal recycling scheme from section 3.3.1 for all $\alpha \geq 1000$. The normal quantile function was computed using the "hybrid" GPU-optimized algorithm from [26].

3.5.1. Speed. We will consider only the timing of code directly related to computation of the gamma quantile function. Anything else, such as uniform random number generation, is not our concern.

The speed of Quantus was evaluated by generating 10^7 random inputs and measuring the time to apply the gamma quantile function to them. This experiment was

TABLE 3

Timings in ms to compute the gamma quantile function q_α for 10^7 pseudorandom uniform variates using an implementation of the algorithm described herein averaged over 100 runs on the two test NVIDIA GPUs. All standard deviations were negligible. Initialization times for the implementation on an Intel Core i5-4670K system are also shown. Timings for the normal quantile function Φ^{-1} using the "hybrid" GPU-optimized algorithm from [26] are given on the bottom row.

	Single-precision			Double-precision		
		Generation			Generation	
α	Init.	Titan	C2050	Init.	Titan	C2050
10^{-9}	0.06	0.70	2.44	10.39	4.32	3.73
10^{-8}	0.37	0.71	2.32	19.84	4.31	3.73
10^{-7}	0.68	0.69	2.32	16.27	4.32	3.74
10^{-6}	0.70	0.72	2.32	17.25	4.38	3.81
10^{-5}	0.83	0.75	2.35	16.88	4.81	4.57
10^{-4}	0.85	1.01	2.65	10.75	7.42	6.63
10^{-3}	0.83	1.50	4.04	9.83	13.72	11.94
10^{-2}	0.79	1.75	5.70	9.47	15.77	13.03
10^{-1}	0.80	1.72	5.47	10.98	13.20	11.36
10^{1}	0.71	1.25	3.69	6.11	9.09	8.21
10^{2}	0.60	1.17	3.44	5.30	8.51	7.65
10^{3}	0.60	1.10	3.08	5.54	6.93	6.51
10^{4}	0.60	0.96	2.59	5.12	6.64	6.23
10^{5}	0.58	0.97	2.59	5.06	6.63	6.23
10^{6}	0.58	0.96	2.59	4.96	6.63	6.23
10^{7}	0.77	0.97	2.59	4.66	6.34	5.96
10^{8}	1.52	0.96	2.59	4.61	6.34	5.96
10^{9}	3.67	0.96	2.59	7.12	6.33	5.96
Φ^{-1}	n/a	0.44	0.97	n/a	4.00	3.61

repeated, with fresh random numbers, 100 times to get an average. NVIDIA's CUDA Random Number Generation library was used to generate the uniformly distributed random numbers. The numbers were pseudorandom and generated by MRG32k3a, and 19,532 blocks of 512 threads were used, with each thread processing a single uniform input. Quantus was accurately timed using CUDA events as per the CUDA C Programming Guide.[10] Using the parameters in Table 2, the coefficients table for the recycling function approximation always comfortably fits[11] in the level 1 (L1) cache on NVIDIA GPUs, so evaluation of the function is very fast.

Table 3 shows the performance of Quantus for a wide range of shape parameters. The performance is always within an order of magnitude of the time to compute the normal quantile function, which is what we were aiming for. Table 3 also shows the initialization times of Quantus. They are relatively high compared to generation, but bear in mind initialization is a fixed cost.

3.5.2. Precision. The precision of Quantus was assessed by inspecting the output from ten out of the 100 runs of each speed test. The gamma output and corresponding uniform input were copied from the GPU to the CPU. 80-bit extended

[10] http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#events

[11] The average table size for single- and double-precision was about 0.4 and 4 KB, respectively.

TABLE 4

Peak relative error statistics for an implementation of the algorithm described herein, over 10^8 *pseudorandom uniform variates for each* α.

	Single-precision		Double-precision	
α	E_1	E_2	E_1	E_2
10^{-9}	nil	nil	2.42×10^{-13}	5.42×10^{-20}
10^{-8}	4.13×10^{-5}	4.13×10^{-13}	2.43×10^{-13}	1.08×10^{-19}
10^{-7}	7.44×10^{-5}	7.44×10^{-12}	2.58×10^{-13}	1.63×10^{-19}
10^{-6}	5.03×10^{-5}	5.03×10^{-11}	2.73×10^{-13}	2.71×10^{-19}
10^{-5}	6.29×10^{-5}	6.29×10^{-10}	3.26×10^{-13}	3.25×10^{-18}
10^{-4}	4.14×10^{-5}	4.14×10^{-09}	2.15×10^{-13}	2.15×10^{-17}
10^{-3}	2.77×10^{-5}	2.77×10^{-08}	1.62×10^{-13}	1.62×10^{-16}
10^{-2}	1.28×10^{-5}	1.28×10^{-07}	1.32×10^{-13}	1.32×10^{-15}
10^{-1}	8.76×10^{-6}	8.76×10^{-07}	4.88×10^{-14}	4.88×10^{-15}
10^{1}	8.15×10^{-7}	7.20×10^{-06}	1.92×10^{-15}	1.45×10^{-14}
10^{2}	1.23×10^{-6}	3.87×10^{-05}	3.01×10^{-15}	6.96×10^{-14}
10^{3}	1.81×10^{-7}	1.49×10^{-05}	6.34×10^{-16}	5.07×10^{-14}
10^{4}	2.23×10^{-6}	1.10×10^{-03}	9.70×10^{-15}	4.94×10^{-12}
10^{5}	2.84×10^{-7}	3.99×10^{-04}	3.27×10^{-16}	4.50×10^{-13}
10^{6}	5.44×10^{-8}	2.66×10^{-04}	2.19×10^{-16}	8.35×10^{-13}
10^{7}	1.02×10^{-7}	1.43×10^{-03}	1.90×10^{-15}	2.90×10^{-11}
10^{8}	7.88×10^{-8}	3.67×10^{-03}	1.99×10^{-16}	7.25×10^{-12}
10^{9}	6.34×10^{-8}	9.71×10^{-03}	1.19×10^{-16}	1.63×10^{-11}

precision references were computed using Boost and compared to the copied values.

There are several error measures we could have used. However, since we are working with floating-point numbers, it makes sense to restrict our attention to relative error measures. The most obvious relative error measure is probably

$$E_1 = \max_u \left| \frac{\tilde{q}_\alpha(u)}{q_\alpha(u)} - 1 \right|, \tag{3.34}$$

where u is a uniform input and $\tilde{q}_\alpha(u)$ denotes the approximation of $q_\alpha(u)$. If $\tilde{q}_\alpha(u)$ and $q_\alpha(u)$ are both ∞, then this measure is taken as zero. This is the same if $\tilde{q}_\alpha(u)$ and $q_\alpha(u)$ are both less than $\min_s$ or $\min_d$ (effectively regarding all subnormal numbers as zero). We also considered the "roundtrip"/backward relative error measure

$$E_2 = \max_u \left| \frac{F_\alpha(\tilde{q}_\alpha(u))}{u} - 1 \right|, \tag{3.35}$$

which we define as zero if u is less than $F_\alpha(\min_s)$ or $F_\alpha(\min_d)$ and $\tilde{q}_\alpha(u)$ is less than either $\min_s$ or $\min_d$. The smallest possible relative error achievable is dependent on machine epsilon (see section 3.1). It should be noted that machine epsilon accuracy is usually overkill in practical simulation applications. For example, extreme accuracy can be traded off against computational speed in many Monte Carlo applications in finance.[12]

Table 4 gives peak relative error statistics for Quantus with the same shape parameters from the speed test. We found Quantus achieves better or comparable accuracy

[12]Senior quantitative analyst at a Tier 1 investment bank, private communication.

with respect to Boost's gamma quantile function in both single- and double-precision, over all distribution parameters and uniform inputs for both relative error measures. The results suggest our algorithm is stable for $\alpha \leq 10^3$. The peak backward errors for $\alpha \geq 10^4$ deteriorate, because of the large magnitude of the variates, but the forward errors are excellent.

4. Conclusions. We have described a method for the efficient and accurate parallel inversion of the gamma distribution. Quantile mechanics—the differential approach to quantile functions—was used with select changes of variable to accomplish this. We showed that the performance of a CUDA GPU implementation of our algorithm is similar to the time to compute the normal quantile function. The underlying algorithmic ideas should translate well to other parallel architectures, e.g., Intel Xeon Phi.

Devroye in [11, p. 404] said a good gamma variate generator should

(i) have uniform generation speed over all shape parameters α;
(ii) be simple to implement; and
(iii) have small or nonexistent initialization times.

We believe our algorithm meets the first and last of these goals. While our algorithm is simple in principle, it is certainly not one of the easiest algorithms to implement. Much of the effort is in efficiently implementing the recurrence relations and automating the generation of the gamma quantile function approximation. However, our generation times are more or less uniformly bounded, and our initialization times are relatively small and get amortized for practical simulation sizes.

Availability of software. An open-source implementation of the gamma quantile function algorithm described herein is available from a public GitHub repository https://github.com/thomasluu/quantus. A production grade GPU and multithreaded CPU implementation of the algorithm is available from NAG. Please see http://www.nag.co.uk or contact the author for more details.

Acknowledgments. The author wishes to thank Prof. William Shaw and Dr. Robert Tong for imparting invaluable knowledge and advice. Dr. Asad Munir clarified various aspects of quantile function theory, for which the author is very grateful. The author also thanks the anonymous referees and various members of NAG for the helpful comments on the initial version of this paper.

REFERENCES

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*, National Bureau of Standards Applied Mathematics Series 55, Washington, D.C., 1964.
[2] A. Azzalini, *A class of distributions which includes the normal ones*, Scand. J. Stat., 12 (1985), pp. 171–178.
[3] A. Azzalini, *The Skew-Normal and Related Families*, Vol. 3, Cambridge University Press, Cambridge, UK, 2013.
[4] D. J. Best and D. E. Roberts, *Algorithm AS* 91: *The percentage points of the* χ^2 *distribution*, J. R. Stat. Soc. Ser. C. Appl. Stat., 24 (1975), pp. 385–388.
[5] P. J. Boland, *Statistical and Probabilistic Methods in Actuarial Science*, Chapman & Hall/CRC, Boca Raton, FL, 2007.
[6] Boost, *Boost C++ Math Toolkit*, http://www.boost.org/users/history/version_1_55_0.html, 2014.
[7] G. E. P. Box and M. E. Muller, *A note on the generation of random normal deviates*, Ann. Math. Statist., 29 (1958), pp. 610–611.

[8] T. Bradley, J. du Toit, M. Giles, R. Tong, and P. Woodhams, *Parallelization techniques for random number generators*, in GPU Computing Gems Emerald Edition, W.-m. W. Hwu, ed., Morgan Kaufmann, Burlington, MA, 2011, pp. 231–246.
[9] C. W. Clenshaw, *A note on the summation of Chebyshev series*, Math. Tables Aids Comput., 9 (1955), pp. 118–120.
[10] G. Corliss and Y. F. Chang, *Solving ordinary differential equations using Taylor series*, ACM Trans. Math. Software, 8 (1982), pp. 114–144.
[11] L. Devroye, *Nonuniform Random Variate Generation*, Springer-Verlag, New York, 1986.
[12] A. R. DiDonato and A. H. Morris, Jr., *Computation of the incomplete gamma function ratios and their inverse*, ACM Trans. Math. Software, 12 (1986), pp. 377–393.
[13] A. Gil, J. Segura, and N. M. Temme, *Efficient and accurate algorithms for the computation and inversion of the incomplete gamma function ratios*, SIAM J. Sci. Comput., 34 (2012), pp. A2965–A2981.
[14] N. J. Higham, *The accuracy of floating point summation*, SIAM J. Sci. Comput., 14 (1993), pp. 783–799.
[15] G. J. Husak, J. Michaelsen, and C. Funk, *Use of the gamma distribution to represent monthly rainfall in Africa for drought monitoring applications*, Int. J. Climatol., 27 (2007), pp. 935–944.
[16] C. Lemieux, *Monte Carlo and Quasi-Monte Carlo Sampling*, Springer Ser. Statist. 20, Springer, New York, 2009.
[17] G. Marsaglia and T. A. Bray, *A convenient method for generating normal variables*, SIAM Rev., 6 (1964), pp. 260–264.
[18] G. Marsaglia and W. W. Tsang, *A simple method for generating gamma variables*, ACM Trans. Math. Software, 26 (2000), pp. 363–372.
[19] M. Matsumoto and T. Nishimura, *Mersenne Twister: A* 623*-dimensionally equidistributed uniform pseudo-random number generator*, ACM Trans. Model. Comput. Simul., 8 (1998), pp. 3–30.
[20] R. E. Moore, *Methods and Applications of Interval Analysis*, SIAM Stud. Appl. Math. 2, SIAM, Philadelphia, 1979.
[21] A. U. K. Munir, *Series Representations and Approximation of Some Quantile Functions Appearing in Finance*, Ph.D. thesis, University College London, London, UK, 2012.
[22] NAG, *The NAG Library*, The Numerical Algorithms Group (NAG), Oxford, UK, 2014.
[23] E. S. Pearson, *Note on an approximation to the distribution of non-central* $\chi 2$, Biometrika, 46 (1959), p. 364.
[24] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2014.
[25] M. Sankaran, *On the non-central chi-square distribution*, Biometrika, 46 (1959), pp. 235–237.
[26] W. T. Shaw, T. Luu, and N. Brickman, *Quantile mechanics* II: *Changes of variables in Monte Carlo methods and GPU-optimised normal quantiles*, European J. Appl. Math., 25 (2014), pp. 177–212.
[27] G. Steinbrecher and W. T. Shaw, *Quantile mechanics*, European J. Appl. Math., 19 (2008), pp. 87–112.
[28] H. C. Thacher, Jr., *Conversion of a power to a series of Chebyshev polynomials*, Commun. ACM, 7 (1964), pp. 181–182.
[29] G. Ulrich and L. T. Watson, *A method for computer generation of variates from arbitrary continuous distributions*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 185–197.
[30] M. J. Wichura, *Algorithm AS* 241: *The percentage points of the normal distribution*, Appl. Stat., 37 (1988), pp. 477–484.