Random Variate Generation by Numerical Inversion When Only the Density Is Known

Derflinger, Gerhard; Hörmann, Wolfgang; Leydold, Josef

DOI:
10.57938/9d128e26-de5a-4a76-b01d-ef936ca97fc2

Published: 01/01/2008

Document Version
Publisher's PDF, also known as Version of record

Random Variate Generation by Numerical Inversion when only the Density Is Known

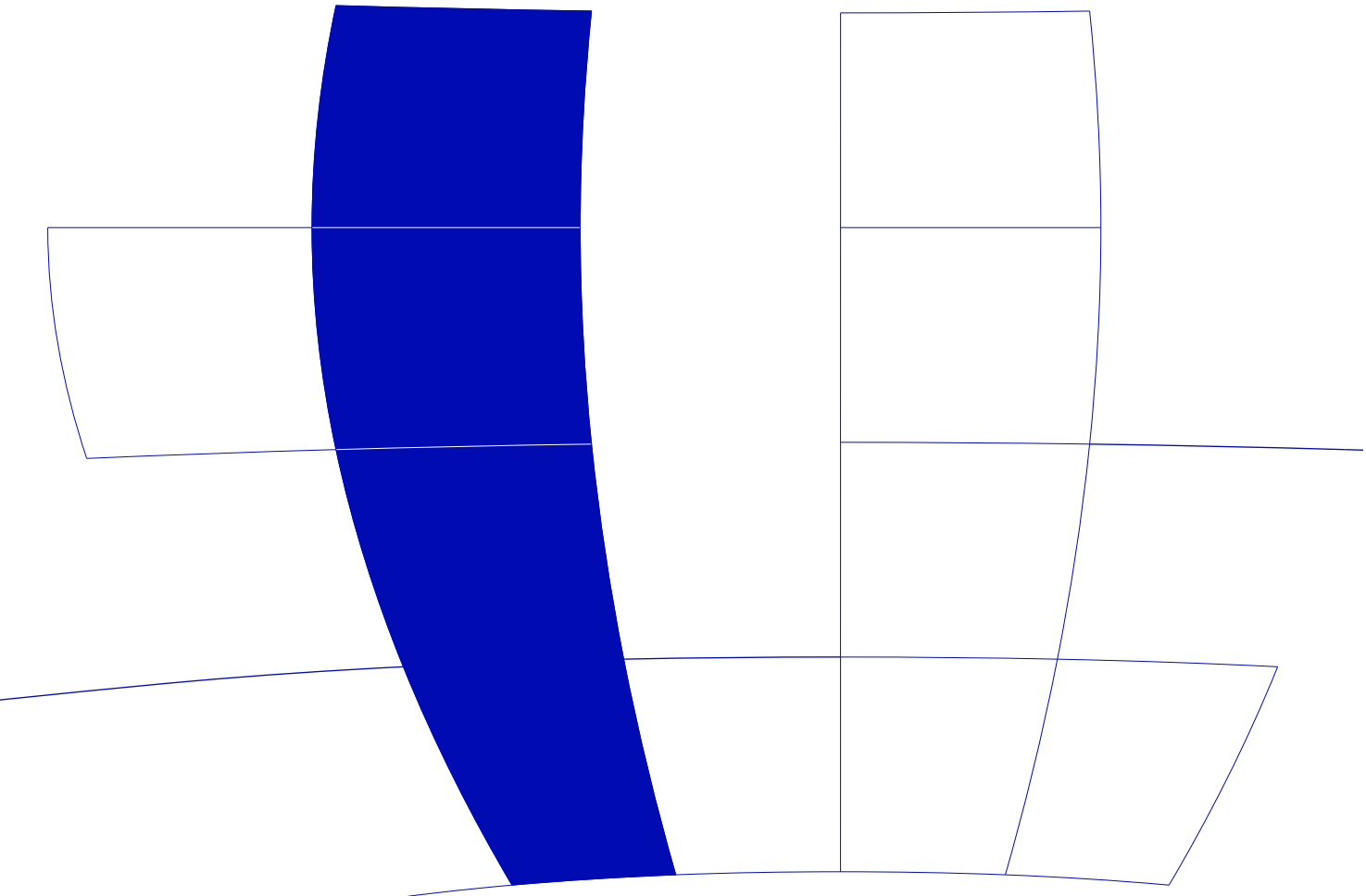
Gerhard Derflinger, Wolfgang Hörmann, Josef Leydold

Department of Statistics and Mathematics
Wirtschaftsuniversität Wien

Research Report Series

Report 78
December 2008

http://statmath.wu-wien.ac.at/



Random Variate Generation by Numerical Inversion when only the Density Is Known

GERHARD DERFLINGER
WU Wien
WOLFGANG HÖRMANN
BU Istanbul
and
JOSEF LEYDOLD
WU Wien

We present a numerical inversion method for generating random variates from continuous distributions when only the density function is given. The algorithm is based on polynomial interpolation of the inverse CDF and Gauss-Lobatto integration. The user can select the required precision which may be close to machine precision for smooth, bounded densities; the necessary tables have moderate size. Our computational experiments with the classical standard distributions (normal, beta, gamma, t-distributions) and with the noncentral chi-square, hyperbolic, generalized hyperbolic and stable distributions showed that our algorithm always reaches the required precision. The setup time is moderate and the marginal execution time is very fast and the same for all distributions. Thus for the case that large samples with fixed parameters are required the proposed algorithm is the fastest inversion method known. Speed-up factors up to 1000 are obtained when compared to inversion algorithms developed for the specific distributions. This makes our algorithm especially attractive for the simulation of copulas and for quasi-Monte Carlo applications.

Categories and Subject Descriptors: G.3 [**Probability and Statistics**]: Random number generation

General Terms: Algorithms

Additional Key Words and Phrases: non-uniform random variates, inversion method, universal method, black-box algorithm, Newton interpolation, Gauss-Lobatto integration

1. INTRODUCTION

The *inversion method* is the simplest and most flexible method for drawing samples of non-uniform random variates. For a target distribution with given cumulative distribution function (CDF) F a random variate X is generated by transforming uniform random variates U using

$$X = F^{-1}(U) = \inf\{x: F(x) \geq U\} \ .$$

For *continuous distributions* with *strictly monotone* CDF, $F^{-1}(u)$ simply is the inverse distribution function (quantile function). The *inversion method* is so at-

Author's address: Gerhard Derflinger and Josef Leydold: Department of Statistics and Mathematics, Vienna University of Economics and Business Administration, Augasse 2-6, A-1090 Vienna, Austria, email: Gerhard.Derflinger@wu-wien.ac.at, Josef.Leydold@wu-wien.ac.at
Wolfgang Hörmann: Department of Industrial Engineering, Boğaziçi University, 34342 Bebek-Istanbul, Turkey, email: hormannw@boun.edu.tr

tractive for stochastic simulation due to several important advantages:

—It is the most *general* method for generating non-uniform random variates. It works for all distributions provided that the CDF is given.

—It transforms uniform random numbers U *one-to-one* into non-uniform random variates X.

—It *preserves* the structural properties of the underlying uniform pseudo-random number generator (PRNG).

—It allows easy and efficient sampling from *truncated* distributions.

—It can be used for variance reduction techniques (common or antithetic variates, stratified sampling, ...).

—It is well suited for *quasi-Monte Carlo* methods (QMC).

—It is essential for *copula* methods.

Hence it has long been the method of choice in the simulation community (see, e.g., Bratley et al. [1983]) and it is generally considered as the only possible alternative for QMC and copula methods.

Unfortunately, the inverse CDF is usually not given in closed form and thus one must use numerical methods. Inversion methods based on well-known root finding algorithms such as Newton method, regula falsi or bisection are *slow* and can only be speeded up by the usage of often large tables. Moreover, these methods are *not exact*, i.e., they produce random numbers which are only approximately distributed as the target distribution. Despite of the importance of the inversion method most simulation software packages do not provide any automatic inversion method; others only provide robust but expensive root finding algorithms (e.g., Brent-Dekker method and the bisection method in SSJ [L'Ecuyer 2008]). An alternative approach uses interpolation of tabulated values of the CDF [Hörmann and Leydold 2003; Ahrens and Kohrt 1981]. The tables have to be precomputed in a setup but guarantee fast marginal generation times which are almost independent of the target distribution. Thus such algorithms are well-suited for the *fixed parameter* case where large samples have to be drawn from the same distribution.

However, often we have distributions where (currently) no efficient and accurate implementation of the CDF is available at all, e.g., generalized hyperbolic distributions [Barndorff-Nielsen and Blæsild 1983] and the non-central χ^2-distribution [Johnson et al. 1995]. Then numerical inversion also requires numerical integration of the probability density function (PDF). The first paper describing a (rather crude) version of that approach seems to be Butler [1970]. Ahrens and Kohrt [1981] describe a numerical inversion algorithm that is based on a fixed decomposition of the interval $(0, 1)$. For each of the subintervals the inverse CDF is approximated by a truncated expansion into Chebyshev polynomials. Their highest orders may vary between subintervals and are selected such that single precision for floating point numbers is reached. The algorithm is designed for fast assembler implementation; especially the non-constant order of the interpolation polynomial over different intervals makes an efficient implementation in a high-level language very cumbersome. Ulrich and Watson [1987] compared that algorithm with some other methods where they combined different ready-to-use routines: double precision integration routine embedded in a Newton root finding algorithm, a packaged ordinary differential

equation solver, Runge-Kutta approximation, and polynomial approximation using B-splines.

Both references given above have a major drawback: They do not allow the user to control the size of the interpolation error. As fast numerical inversion algorithms are of greatest practical importance due to the advantages listed above, we are convinced that there is need for a paper that describes all numerical tools for such an algorithm and explains the non-trivial technical details necessary to reach high precision. The aim of our research in the last five years was therefore to design a robust black-box algorithm to generate continuous random variates by numerical inversion. The user only has to provide the PDF and a "typical point" in the domain of the distribution (e.g., a point near the mode) together with the size of the maximal acceptable error. We arrived at an algorithm that is based on polynomial interpolation of the inverse CDF utilizing Newton's formula together with Gauss-Lobatto integration. Our algorithm is new, compared to the algorithms of [Ahrens and Kohrt 1981; Ulrich and Watson 1987], as it introduces automatically selected subintervals of variable length and a rigorous control of the error. Compared to [Hörmann and Leydold 2003] the new algorithm has the main practical advantage that it does not require the CDF together with the PDF but only the PDF. It also requires a smaller number of intervals and numerical tests show that the error control is much improved.

The paper is organized as follows. In Section 2 we discuss some principles of numerical inversion, in particular the concept of *u-error* that is used to assess the quality of a numerical inversion procedure. Section 3 describes all ingredients of the proposed algorithm, that is, Newton's interpolation formula, Gauss-Lobatto quadrature, estimation of appropriate cut-off points for tails, and a method for finding a partition of the domain. In Section 4 we compile the details of the algorithm and Section 5 summarizes some of our computational experiences.

2. BASIC CONSIDERATIONS ABOUT NUMERICAL INVERSION

2.1 Floating Point Arithmetic and Exact Random Variate Generation

In his book Devroye [1986] assumes that *"our computer can store and manipulate real numbers"* (Assumption 1, p. 1). However, in his model "exact random variate generation by inversion" is only possible if the inverse CDF, $F^{-1}(u)$, is available in closed form (using *"fundamental operations"* that are implemented exactly in our computer, Assumption 3). While these assumptions are fundamental for a mathematically rigorous theory of non-uniform random variate generation, they are far from being realistic for common simulation practice.

Virtually all MC or QMC experiments run on a real-world computer that uses floating point numbers. Usually the *double* format of the IEEE floating point standard is used which takes 64 bits for one number resulting in a machine precision of $\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$ (see Overton [2001] for a survey on the corresponding IEEE 754 standard). Thus a continuous distribution with a very high and narrow peak or pole actually becomes a discrete distribution, in particular when the mode or pole is located far from 0. Thus even for an exact inversion method we cannot avoid round-off errors that are bounded from below by the machine precision.

We therefore think that it is more realistic to call an algorithm *"exact"* if its

precision is close to machine precision, i.e., its relative error is not much larger than 10^{-15}. To be able to use this idea we must thus start with a definition of *error* and a feasible upper bound for the maximal tolerated deviation.

2.2 Approximation Error and u-Resolution

Let F_a^{-1} denote the approximate inverse CDF. Then we define the absolute *x-error* at some point $u \in (0,1)$ by

$$\varepsilon_x(u) = |F^{-1}(u) - F_a^{-1}(u)| \; .$$

However, it requires the close to exact computation of the inverse CDF and thus it could only be applied for testing an inversion algorithm on a small set of test distributions, for which the inverse CDF is computable. Moreover, a bound for $\varepsilon_x(u)$ requires that an algorithm is very accurate in the far tails of the distributions, i.e., for large values of $F^{-1}(u)$. On the other hand, if we replace the absolute error by the *relative* x-error, $\varepsilon_x(u)/|F^{-1}(u)|$ our algorithm must be very accurate near 0.

A better choice is the *u-error* at a point $u \in (0,1)$ given by

$$\varepsilon_u(u) = |u - F(F_a^{-1}(u))| \; . \tag{1}$$

It has some properties that make it a convenient and practical relevant measure of error in the framework of numerical inversion.

—$\varepsilon_u(u)$ can easily be computed provided that we can compute F sufficiently accurate. Thus the maximal u-error can be estimated during the setup.

—Uniform pseudo-random number generators work with integer arithmetic and return points on a grid. Thus these pseudo-random points have a limited resolution, typically $2^{-32} \approx 2.3 \times 10^{-10}$ or (less frequently) machine precision $2^{-52} \approx 2.2 \times 10^{-16}$. Consequently, the positions of pseudo-random numbers U are not random at all at a scale that is not much larger than their resolution. u-errors can be seen as minor deviations of the underlying uniform pseudo-random points U_i from their "correct" positions. We consider this deviation as negligible if it is (much) smaller than the resolution of the pseudo-random variate generator.

—The same holds for QMC experiments where the F-discrepancy [Tuffin 1997] of a point set $\{X_i\}$ is computed as discrepancy of the set $\{F(X_i)\}$. If the X_i are generated by exact inversion their F-discrepancy coincides with the discrepancy of the underlying low-discrepancy set. Thus $\varepsilon_u(u)$ can be used to estimate the maximal change of the F-discrepancy compared to the "exact" points.

—Consider a sequence of approximations F_n^{-1} to the inverse CDF F^{-1} such that $\varepsilon_{u,n}(u) < \frac{1}{n}$ and let F_n be the corresponding CDF. Then $|F(x) - F_n(x)| = |F(F_n^{-1}(u)) - F_n(F_n^{-1}(u))| = |F(F_n^{-1}(u)) - u| = \varepsilon_{u,n}(u) \to 0$ for $n \to \infty$. That is, the CDFs F_n converge weakly to the CDF F of the target distribution and the corresponding random variates $X_n = F_n^{-1}(U)$ converge in distribution to the target distribution [Billingsley 1986].

We are therefore convinced that the u-error is a quite natural concept for the approximation error of numerical inversion. We use the maximal u-error as our criterion for approximation errors when calculating inverse CDFs numerically. We

call the maximal tolerated u-error of an algorithm the *u-resolution* of the algorithm, denoted by ε_u. In the sequel we consider it as a control parameter for a numerical inversion algorithm. It is a design goal of our algorithm to have

$$\sup_{u\in(0,1)} |u - F(F_a^{-1}(u))| \le \varepsilon_u \ . \tag{2}$$

We should also mention two possible drawbacks of the concept of u-resolution. First, it does not work for continuous distributions with high and narrow peaks or poles. Due to the limitations of floating point arithmetic the u-error is at least of the order of the probability of the mass points described in Sect. 2.1 above. However, this just illustrates the severe problems the floating point arithmetic has with such distributions. Secondly, the simple formula for the x-error

$$\varepsilon_x(u) = \varepsilon_u(u)/f(x) + O(\varepsilon_u(u)^2)$$

implies that the x-error of the approximation may be large in the tails of the target distribution. However, we are not considering the problem of calculating exact quantiles in the far tails here as it is (in our opinion) not necessary for the inversion method as the far tails cannot influence the u-error.

2.3 Design of an Automatic Inversion Algorithm

The aim of this paper is to develop an inversion algorithm that can be used to sample from a variety of different distributions. The user only has to provide

—a function that evaluates the PDF of the target distribution,
—a "typical point" of the distribution, that is, a point in the domain of the distribution not too far away from the mode, and
—the desired u-resolution.

We call such algorithms "automatic" or "black box", see Hörmann et al. [2004]. The algorithm uses tables of interpolating polynomials and consists of two parts: (1) the *setup* where all necessary constants for the given distribution are computed and stored in tables; and (2) the *sampling* part where the interpolating polynomials are evaluated for a particular value $u \in (0, 1)$. The setup is the crucial part and we may say that the setup "designs" the algorithm automatically.

The choice of the given u-resolution depends on the particular application and is obviously limited from below by the machine precision. It is important that the maximum u-error can be estimated and controlled in the setup, that is, $\varepsilon_u(u)$ must not exceed the requested u-resolution. Moreover, we wish that a u-resolution close to machine precision (say down to 10^{-13}) can be reached with medium-sized tables storing less than 10^4 double constants. We also hope that the sampling part of the algorithm is (very) fast, about as fast as generating exponential random variates by inversion.

It should be clear that for reaching such high aims we also have to accept down sides. For an accurate algorithm we have to accept a slow setup. Of course, we also cannot expect that an automatic algorithm works for all continuous distributions. For our numerical inversion algorithm we have to assume that the density of the distribution is bounded, sufficiently smooth and positive on its relevant domain. The necessary mathematical conditions for the smoothness can be seen from the

error bounds for numerical integration and interpolation. Moreover, if the density is multimodal, then it must not vanish between its modes. (In practice this means that the density must not be close to zero in a region around a local minimum.) If there are isolated points of discontinuity or where other assumptions do not hold, it is often possible to decompose the domain of the distribution. We thus obtain intervals with smooth PDF and can apply the approximation procedure.

3. BUILDING BLOCKS OF NUMERICAL INVERSION

As we start with density $f(x)$ we have to solve the following problems:

(1) *Computational domain:* Find the computationally relevant domain $[b_l, b_r]$ of the distribution, that is, the region where the construction of our approximation of the inverse CDF is numerically stable, and where the probability of falling into this region is sufficiently close to 1.

(2) *Subintervals:* Divide the domain of the distribution into intervals $[a_{i-1}, a_i]$, $i = 1, \dots k$, with $b_l = a_0 < a_1 < \ldots < a_k = b_r$.

(3) *Interpolation:* For each interval we approximate the inverse CDF F^{-1} on the interval $[F(a_{i-1}), F(a_i)]$ by interpolating the construction points $(F(x_j), x_j)$ for some points $a_{i-1} = x_0 < x_1 < \ldots x_n = a_i$. Notice that there is no need to evaluate $F^{-1}(u)$.

(4) *Numerical integration:* Compute the approximate CDF, $F_a(x)$, by iteratively computing $\int_{x_{j-1}}^{x_j} f(x)\,dx$ for $j = 1, \dots, n$ on each interval.

For Task (4) we use (adaptive) Gauss-Lobatto quadrature (also known as Radau integration). For Task (3) we found that Newton's recursion for the interpolating polynomial ("Newton's interpolation formula") with a fixed number of points is well-suited. Both, numerical integration and interpolation lead to small errors when applied on short intervals and they allow the estimation of u-errors on each interval. Thus we can accomplish Task (2) by selecting proper intervals which are short enough to gain sufficient accuracy but not too short thus avoiding needless large tables. Notice that by this approach the computation of the approximation is carried out on each interval independently from the others. We will see that using the same intervals for integration and for interpolation leads to significant synergies. Task (1) is important as our approach only works for distributions with bounded domains. Moreover, regions where the CDF is extremely flat (as in the tails of the distributions) result in an extremely steep inverse CDF and its polynomial interpolation becomes numerically unstable.

The sampling part of the algorithm is straightforward. We use indexed search [Chen and Asau 1974] to find the correct interval together with evaluation of the interpolation polynomial.

3.1 Newton's Interpolation Formula

Polynomial interpolation for approximating a function $g(x)$ on some interval $[x_0, x_n]$ is based on the idea to use a polynomial $P_n(x)$ of order n such that

$$g(x_i) = P_n(x_i), \quad \text{for } i = 0, \dots, n,$$

where $x_0 < x_1 < \ldots < x_n$ are some fixed points. Note, that we use the borders of the interval, x_0 and x_n, as interpolation points to avoid discontinuities of the approximate polynomials at successive intervals. For smooth functions the approximation error at a point $x \in (x_0, x_n)$ is given by

$$|g(x) - P_n(x)| = \frac{g^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \quad \text{for some } \xi \in (x_0, x_n).$$

Thus for a function with bounded $(n+1)$-st derivative the approximation error is of order $O((x_n - x_0)^{n+1})$ and can be made arbitrarily small by using short intervals.

Newton's interpolation formula uses a numerically stable representation of the polynomial $P_n(x)$ and a simple and elegant recursion to calculate the coefficients of that representation. Using the ansatz (see, e.g., Schwarz [1997, Sect. 3.4] or Dahlquist and Björck [2008, Sect. 4.2.1])

$$P_n(x) = c_0 + \sum_{k=1}^{n} c_k \prod_{i=1}^{k} (x - x_i)$$

we find that

$$c_k = g[x_0, x_1, \ldots, x_k] \qquad \text{for } k = 0, \ldots, n,$$

where the *divided differences* are recursively defined by

$$g[x_0, x_1, \ldots, x_k] = \frac{g[x_1, \ldots, x_k] - g[x_0, \ldots, x_{k-1}]}{x_k - x_0} \quad \text{and} \quad g[x_i] = g(x_i).$$

This formula allows to compute the coefficients c_k using Routine 1. The polynomial can be evaluated at some point $x \in [x_0, x_n]$ using the Horner like scheme in Routine 2.

Routine 1 `NCoef` (Newton-Coefficients)

Input: Nodes $x_0 < \ldots < x_n$, values $g(x_0), \ldots, g(x_n)$.
Output: Coefficients $c_0, \ldots, c_n$ for interpolating polynomial P_n.

1: **for** $i = 0, \ldots, n$ **do**
2: $\quad c_i \leftarrow g(x_i)$.
3: **for** $k = 1, \ldots, n$ **do**
4: $\quad$ **for** $i = n, n-1, \ldots, k$ **do**
5: $\quad\quad c_i \leftarrow (c_i - c_{i-1})/(x_i - x_{i-k})$.
6: **return** $c_0, \ldots, c_n$.

We used *Chebyshev points* of $(n+1)$-st order as nodes for P_n, i.e., the roots of the Chebyshev polynomial of order $n+1$, given by

$$\cos\left(\frac{2k+1}{n+1} \cdot \frac{\pi}{2}\right), \quad \text{for } k = 0, 1, \ldots, n,$$

rescaled to our intervals $[x_0, x_n]$ such that the smallest and the largest root are mapped on the boundary points x_0 and x_n. These points lead to a minimal upper

Routine 2 `NEval` (Newton-Evaluate)

Input: Coefficients c_k of P_n, nodes $x_0, \ldots, x_n$, point $x \in [x_0, x_n]$.
Output: Value of $P_n(x)$.

```
1: p ← c_n.
2: for k = n − 1, n − 2, ..., 0 do
3:     p ← c_k + (x − x_k) p.
4: return p.
```

bound for the maximal approximation error. We call these points the *rescaled Chebyshev points* in the following. For the interval $[0, 1]$ we find

$$x_k = \frac{\sin(k\,\phi)\,\sin((k+1)\,\phi)}{\cos(\phi)} \quad \text{for } k = 0, 1, \ldots, n, \quad \text{where } \phi = \frac{1}{(n+1)}\,\frac{\pi}{2} \tag{3}$$

and for the differences

$$x_{k-1} - x_k = \sin(2k\phi)\,\tan(\phi) \quad \text{for } k = 1, \ldots, n\,.$$

3.2 Inverse Interpolation

For an interval $[F(a_{k-1}), F(a_k)]$ we use Newton's interpolating formula to construct an approximating polynomial F_a^{-1} for the inverse CDF F^{-1}. As nodes we use $u_i = F(x_i)$, where x_i are the rescaled Chebyshev points on the interval $[a_{k-1}, a_k]$. Chebyshev interpolation, i.e., Newton interpolation using the Chebyshev points, is that special case of Newton interpolation for which the smallest error bounds can be proven. The transformation F is nearly linear within an interval $[a_{k-1}, a_k]$ (except for far tails). Thus this approach avoids the evaluation of the inverse CDF and leads to close to minimal interpolation errors in the center of the distribution. It may lead to larger than optimal errors in the tails, but it is still better than using equidistant u_i. A disadvantage is that we have to store the values of u_i in a table. We finally decided to use this simple approach as it leads to a stable setup.

For the x-error we find

$$|F^{-1}(u) - F_a^{-1}(u)| \le \max_{v \in [F(a_{k-1}), F(a_k)]} \frac{\left(\frac{d}{dv}\right)^{n+1} F^{-1}(v)}{(n+1)!} \prod_{i=0}^{n} (u - u_i)\,,$$

and thus we get the first approximation of the u-error by

$$\varepsilon_u(u) \approx \varepsilon_x(u)\, f(F^{-1}(u)) \approx f(F^{-1}(u))\,(F^{-1})^{(n+1)}(u) \frac{1}{(n+1)!} \prod_{i=0}^{n} (u - u_i)\,,$$

where $(F^{-1})^{(n+1)}(u)$ denotes the $n+1$st derivate of F^{-1}. Notice that it can be computed using only derivatives of $f(x)$ by means of Faá di Bruno's formula. Of course the u-error is much smaller than the x-error in the tails. For short intervals the term $f(F^{-1}(u))\,(F^{-1})^{(n+1)}(u)$ remains almost constant and the maximal u-error in an interval is attained close to the extrema of $\prod_{i=0}^{n}(u - u_i)$ that we denote by t_i, $i = 1, \ldots, n$. We therefore use

$$u\text{-error} \approx \max_{i=1,\ldots,n} |t_i - F_i(F_a^{-1}(t_i))| \tag{4}$$

as a simple estimate for the u-error of the interpolation, where F_i is the approximate CDF computed using numerical integration of the PDF. For the polynomial $p(u) = \prod_{i=0}^{n}(u-u_i)$ we find

$$p'(u) = \sum_{k=0}^{n}\prod_{\substack{i=0\\ i\neq k}}^{n}(u-u_i) = p(u)\sum_{k=0}^{n}\frac{1}{u-u_k}\ .$$

Since the points $u_0 < \ldots < u_n$ are distinct, $p'(u)$ cannot be zero at u_k, $k = 0, \ldots n$. Thus we only need to find the roots of $\sum_{k=0}^{n}\frac{1}{u-u_k}$ (approximately). We found out that this can be done efficiently using only two iterations of Newton's root finding method, where we use the recursion

$$\hat{u}_{new} = \hat{u} + \frac{\sum_{k=0}^{n}(1/(\hat{u}-u_k))}{\sum_{k=0}^{n}(1/(\hat{u}-u_k)^2)}\ .$$

The entire algorithm for computing the test points t_k is compiled in Routine 3.

Routine 3 `NTest` (Newton-Testpoints)

Input: Nodes $u_0 < \ldots < u_n$.
Output: Test points $t_1 < \ldots < t_n$.

1: **for** $k = 1, \ldots, n$ **do**
2: $\quad t_k \leftarrow (u_{k-1} + u_k)/2.$
3: $\quad$ **for** $j = 1, 2$ **do** ▷ 2 Newton steps
4: $\quad\quad s \leftarrow 0, \quad sq \leftarrow 0.$
5: $\quad\quad$ **for** $i = 0, \ldots, n$ **do**
6: $\quad\quad\quad s \leftarrow s + 1/(t_k - u_i), \quad sq \leftarrow sq + 1/(t_k - u_i)^2.$
7: $\quad\quad t_k \leftarrow t_k + s/sq.$
8: **return** $t_1, \ldots, t_n$.

Monotonicity of the approximated inverse CDF, F_a^{-1}, might also be an issue for some applications. There exist a lot of literature on shape-preserving (mostly cubic) spline interpolation. However, methods for constructing monotone polynomials that interpolate monotone points are rather complicated (e.g., Costantini [1986]). Thus we have decided to use a rather simple technique and only check the monotonicity at the points $F_a^{-1}(t_i)$ that we already use in (4) for estimating the u-error.

To increase the numeric stability of the interpolation of the inverse CDF it is very useful to shift the x and the u coordinate of the CDF for every interval into the origin. Therefore we define $F_k(x) = F(x + a_{k-1}) - F(a_{k-1}) = \int_{a_k}^{a_{k-1}+x} f(t)\,dt$ for $x \in [0, a_k - a_{k-1}]$. Then the problem of inverting $F(x)$ on $[a_{k-1}, a_k]$ is equivalent to inverting $F_k(x)$ on $[0, a_k - a_{k-1}]$: for a $u \in [F(a_{k-1}), F(a_k)]$ we get $F^{-1}(u)$ by

$$F^{-1}(u) = a_{k-1} + F_k^{-1}(u - F(a_{k-1}))\ .$$

This transformation has two advantages. The computation of $F_k(x)$ by integrating $\int_{a_k}^{a_{k-1}+x} f(t)\,dt$ is numerically more stable as the subtraction $F(x+a_{k-1}) - F(a_{k-1})$, that might cause loss of accuracy, is no longer necessary. This improves in particular the numeric precision in the right tail of the distribution. Moreover, the first node of our interpolation problem is always $(0, 0)$ which saves memory and computations.

Remark 1. We use linear interpolation as a fallback when Newton's interpolation formula fails due to numerical errors.

Remark 2. An alternative approach is to use of the optimal interpolation points, i.e., the Chebyshev points rescaled for $[F(a_{k-1}), F(a_k)]$. This is convenient as, besides a linear transformation, these are the same for all intervals and we need not store them in a table. Moreover, this also holds for the points t_i for the error estimate (4) and, therefore, we would not need Routine `NTest`. However, the main drawback of this alternative is that we have to invert the CDF to calculate $x_i = F^{-1}(u_i)$ which may lead to numerical problems due to rounding errors especially in the tails.

Remark 3. We also made experiments with other interpolations. However, we have decided against these alternatives. For example, continued fractions have the advantage that we can extrapolate towards poles but we were not able to estimate interpolation errors during the setup.

3.3 Gauss-Lobatto Quadrature

There exist many different quadrature rules. We are interested in very precise results for smooth f on short intervals. In our experiments with several standard distributions we noticed that Gauss-Lobatto quadrature with 5 nodes resulted in very small errors for all intervals that we obtained from the interpolation algorithm. For the interval $[0, 1]$ it uses the nodes $c_1 = 0$, $c_2 = \frac{1}{2} - \sqrt{3/28}$, $c_3 = \frac{1}{2}$, $c_4 = \frac{1}{2} + \sqrt{3/28}$, and $c_5 = 1$ with corresponding weights $w_1 = w_5 = \frac{9}{180}$, $w_2 = w_4 = \frac{49}{180}$, and $w_3 = \frac{64}{180}$. Then for a density f on $[a, a + h]$ we have

$$\int_a^{a+h} f(x)\,dx \approx \hat{I}_{(a,a+h)}[f] = \sum_{i=1}^{5} w_i\, h\, f(a + c_i\, h)\ . \tag{5}$$

The integration error for an eight times continuously differentiable density f is given by [Abramowitz and Stegun 1972]

$$\left| \int_a^{a+h} f(x)\,dx - \hat{I}_{(a,a+h)}[f] \right| \le 7.03 \cdot 10^{-10}\, h^9 \max_{\xi \in [a,a+h]} f^{(8)}(\xi)\ . \tag{6}$$

Thus numerical integration works for smooth $f(x)$ with bounded 8th derivative.

Remark 4. We found that errors for Gauss-Lobatto quadrature are usually much smaller than those of interpolation. Note, however, that (e.g.) for the Gamma distribution with shape parameter $\alpha < 2$ we have unbounded derivatives at zero. This leads to an integration error for the interval $(0, h)$ which increases linearly with h. But this difficulty can be overcome by using adaptive integration.

Adaptive integration is a standard method for numerically integrating $f(x)$ on long intervals. In addition, it allows to (roughly) estimate the integration error. We use a variant where we halve the intervals, till the total integral does not change any more, see Routine 4 (`AGL`).

In practice evaluations of f and the two subintegrals for I_1 are passed to `AGL` in each recursion step. Thus a single call to this routine requires 11 evaluations of f, and each additional step 6 evaluations. We also need a rough estimate for

Routine 4 `AGL` (Adaptive-Gauss-Lobatto)

Input: Density $f(x)$, domain $[a, a+h]$, tolerance tol.

Output: $\int_a^{a+h} f(x)\,dx$ with estimated maximal error less than tol.

```
1: I_0 ← Î_(a,a+h)[f].
2: I_1 ← Î_(a,a+h/2)[f] + Î_(a+h/2,a+h)[f].
3: if |I_0 − I_1| < tol then
4:    return I_1.
5: else
6:    return (AGL(f, (a, a + h/2), tol) + AGL(f, (a + h/2, a + h), tol)).
```

$\int_{b_l}^{b_r} f(x)\,dx$ in order to set tolerance tol as the function f can be any positive multiple of a density function.

Notice that $|I_0 - I_1|$ is just an estimate for the integration error and does not provide any upper bound. Many papers suggest to replace tol by $tol/2$ when recursively calling the adaptive quadrature algorithm in order to obtain an upper bound for the estimated error. However, we observed in our experiments that halving the tolerance leads to severe problems for distributions with heavy tails (e.g., the Cauchy distribution) where it never reaches the precision goal. Thus we followed a version described by Gander and Gautschi [2000]. They argue[1] that halving the tolerance for every subdivision leads to lots of unnecessary function evaluations without providing an upper bound for the (true) integration error, that is not available with that type of error bound anyway. In all our experiments, the errors when using Routine 4 where smaller than required. We have even observed in our experiments that for nearly all cases the intervals for Newton's interpolation formula are adequately short for simple (non-adaptive) Gauss-Lobatto quadrature (5) to obtain sufficient accuracy. However, we found that for distributions with high tails (e.g., Cauchy) or unbounded derivatives (e.g., Gamma with shape parameter α close to one) we need adaptive integration. The following procedure was quite efficient then:

0. Roughly compute $I_0 = \hat{I}_{(b_l,b_r)}[f]$ to get maximal tolerated error tol.
1. Compute $\hat{I}_{(b_l,b_r)}[f]$ with required accuracy by adaptive Gauss-Lobatto quadrature using Routine `AGL` and store subinterval boundaries and CDF values. We used $tol = 0.05\, I_0\, \varepsilon_u$.
2. When integrals $\hat{I}_{(x_{j-1},x_j)}[f]$ have to be computed we use the intervals and the density values from above for simple Gauss-Lobatto quadrature.

Remark 5. There exist many other quadrature rules as well. An alternative would be Gauss-Legendre quadrature with 4 nodes. It has an integration error similar to Gauss-Lobatto quadrature with 5 nodes but as the latter uses the interval endpoints as nodes it is especially suited for recursive subdivisions of the intervals. Then it requires only 6 additional evaluations of f in opposition to Gauss-Legendre where 8 evaluations are necessary. When we have n successive

[1] Private communication from Walter Gander.

sub-intervals then Gauss-Lobatto requires $4\,n+1$ evaluations of f compared with $4\,n$ for Gauss-Legendre.

3.4 Cut-off Points for the Computational Domain

Newton's interpolation formula becomes numerically unstable when the inverse CDF is very steep. Thus numerical problems arise for the case where the density f is close to zero over some subinterval. This leads to cancellation errors and overflows when calculating the coefficients of the interpolating polynomial. For many (in particular for all unimodal) distributions this can occur only in the tails. It is therefore important for the successful application of our algorithm that we have to cut off these parts from the domain of a distribution. Of course the tail regions must have small probabilities as they contribute to the total u-error. We used a probability of $0.05\,\varepsilon_u$ for either tail.

For this task we have to find easy-to-compute approximations for the quantiles in the far tails. Fortunately this can be solved by means of the concept of *local concavity* of a density f at a point x [Hörmann et al. 2004, Sect. 4.3]. Let T_c be the strictly monotone increasing transformations[2] $T_c(x) = \operatorname{sgn}(c)\,x^c$ if $c \neq 0$ and $T_0(x) = \log(x)$ otherwise. The local concavity is the maximal value for c such that the transformed density $\tilde{f}(x) = T_c(f(x))$ is concave near x. For a twice differentiable density f it is given by

$$\mathrm{lc}_f(x) = 1 - \frac{f''(x)\,f(x)}{f'(x)^2}$$

and can be calculated sufficiently accurate by using

$$\mathrm{lc}_f(x) = \lim_{\delta\to 0}\left(\frac{f(x+\delta)-f(x)}{f(x+\delta)} + \frac{f(x-\delta)-f(x)}{f(x-\delta)}\right) - 1$$

with a suitably chosen finite δ. Now consider the tangent $\tilde{g}(x) = \tilde{f}(p) + \tilde{f}'(p)\,(x-p)$ on $\tilde{f}(x)$ in the point p of the left tail region with $c = \mathrm{lc}_f(p)$. Let us denote by $g(x)$ the function which one obtains by applying the inverse transformation T_c^{-1} on $\tilde{g}(x)$,

$$g(x) = T_c^{-1}(\tilde{g}(x)) = \begin{cases} f(p)\,\left(1 + c\,\frac{f'(p)}{f(p)}\,(x-p)\right)^{1/c} & \text{for } c \neq 0\,, \\ f(p)\,\exp\left(\frac{f'(p)}{f(p)}\,(x-p)\right) & \text{for } c = 0\,. \end{cases}$$

Notice that $\tilde{f}(x)$ is concave for $x \le p$ (we then say that f is T_c-concave) whenever $\mathrm{lc}_f(x) \ge c$ for all $x \le p$. Then $\tilde{g}(x) \ge \tilde{f}(x)$ and $g(x) \ge f(x)$ for $x \le p$. Similarly, $\tilde{f}(x)$ is convex and $g(x) \le f(x)$ for $x \le p$ when $\mathrm{lc}_f(x) \le c$ for all $x \le p$. Equality holds if $\mathrm{lc}_f(x)$ is constant. Thus assume that $\mathrm{lc}_f(x)$ is approximately constant in the far tails and let us replace f by g. The function $G(x) = \int_{-\infty}^{x} g(t)\,dt$ or $G(x) = \int_{b_0}^{x} g(t)\,dt$ if the domain of g is bounded at b_0 can be computed and inverted in closed form and thus it is no problem to solve the equation $G(p) = \varepsilon = 0.05\,\varepsilon_u$.

[2] Notice that these are quite similar to the Box-Cox transformations.

Denoting its root by p^* we get the simple formulas

$$p^* = \begin{cases} p + \frac{f(p)}{c\,f'(p)} \left(\left(\frac{\varepsilon\,|f'(p)|\,(1+c)}{f(p)^2} \right)^{c/(1+c)} - 1 \right) & \text{for } c \neq 0, \\ p + \frac{f(p)}{f'(p)} \log \frac{\varepsilon\,|f'(p)|}{f(p)^2} & \text{for } c = 0. \end{cases} \tag{7}$$

Note that by using the absolute value $|f'(p)|$ we obtain for both $c \neq 0$ and $c = 0$ a single formula applicable for both, the right and the left tail. The result p^* can be used as new value for p thus defining a simple recursion that converges very fast close to exact results. It works for all standard distributions we have tried.

Remark 6. There is no need for a cut-off point when the domain is bounded from the left by d_l and $f(d_l)$ or $f'(d_l)$ is greater than zero. Analogously for a right bound d_r.

3.5 Construct Subintervals for Piecewise Interpolation

We have to subdivide the domain of the distribution into intervals $[a_{i-1}, a_i]$, $i = 1, \ldots k$, with $b_l = a_0 < a_1 < \ldots < a_k = b_r$. We need sufficiently short intervals to reach our accuracy goal. On the other hand many intervals result in large tables. We use a simple technique to solve this problem: We construct the intervals starting from the left boundary point of the computational point and proceed to its right boundary. We start with some initial interval length, compute the interpolating polynomial and estimate the error using (4). If the error is larger then ε_u we have to shorten the interval and try again. Otherwise we fix that interval and store its interpolation coefficients and proceed to the next interval. If the error for the last interval was smaller than required we try a slightly longer one now.

Remark 7. An alternative approach is interval bisection: Start with a partition of the (computational) domain $[b_l, b_r]$. Whenever the u-error is too large in a interval it is split into two subintervals. This procedure is used in [Hörmann and Leydold 2003] (where the CDF is directly available) but results in a larger number of intervals and thus larger tables. For the setting of this paper, where we start from the PDF and combine numeric integration with interpolation, interval bisection it is less suited.

3.6 Adjust Error Bounds

We have several sources of numerical error: Cutting off tails, integration errors and interpolation errors. As we want to control the maximal error we have to adjust the tolerated u-error in each of these steps. Let $\breve{\varepsilon}_u$ denote the requested u-resolution. Then we use $\varepsilon_u = 0.9\,\breve{\varepsilon}_u$ for the maximal tolerated error for the interpolation error as computed in (4). For the probabilities of the truncated tails we use $0.05\,\varepsilon_u$ and for the integration error we allow at most $0.05\,I_0\,\varepsilon_u$. By this strategy the total u-error was always below $\breve{\varepsilon}_u$ for all our test distributions (when $\breve{\varepsilon}_u \geq 10^{-12}$).

4. THE ALGORITHM

Algorithm `NINIGL` (Numerical Inversion with Newton Interpolation and Gauss-Lobatto integration) compiles all building blocks in a lean form.

Algorithm 1 NINIGL

Input: Density $f(x)$, center x_c of distribution, u-resolution ε_u, order n.
Output: Random variate with approximate density f and maximal u-error ε_u.

▷ Adjust

1: $\varepsilon_u \leftarrow 0.9\,\varepsilon_u$.

▷ Preprocessing

2: Find points $\tilde{b}_l < x_c < \tilde{b}_r$ with $f(\tilde{b}_l) \approx f(\tilde{b}_r) \approx 10^{-13} f(x_c)$.
3: Roughly Estimate $I_0 \leftarrow \hat{I}_{(\tilde{b}_l,\tilde{b}_r)}[f]$.
4: Find cut-off points b_l and b_l for computational domain with $\text{Prob}(X < bl) \approx \text{Prob}(X > br) \approx 0.05\, I_0\, \varepsilon_u$. Use recursion (7).
5: Compute $I \leftarrow \texttt{AGL}(f, [b_l, b_r], tol = 0.05\, I_0\, \varepsilon_u)$.
[Store all calculated subintervals and their CDF values in a table.]

▷ Setup

6: Set $a_0 \leftarrow b_l$, $h \leftarrow (b_r - b_l)/128$, $F_0 = 0$, and $k \leftarrow 0$.
7: **while** $a_k < b_r$ **do**
8: **loop** ▷ interpolating polynomial on $[a_k, a_k + h]$
9: Set $x_0 = 0, x_1, \ldots, x_n = h$ to rescaled Chebyshev points, see (3).
10: Set $u_0 \leftarrow 0$, compute $u_i \leftarrow u_{i-1} + \hat{I}_{(x_{i-1},x_i)}[f]$ for all $i = 1, \ldots n$.
[Reuse table from Step 5 together with simple Gauss-Lobatto.]
11: Compute coefficients $\{c_j\} \leftarrow \texttt{NCoef}(\{u_j\}, \{x_j\})$.
12: Compute test points $\{t_j\} \leftarrow \texttt{NTest}(\{u_j\})$.
13: Compute $\xi_i \leftarrow \texttt{NEval}(\{c_j\}, \{u_j\}, t_i) \quad [= F_a^{-1}(t_i)]$ for all $i = 1, \ldots n$.
14: Compute $\varepsilon_i \leftarrow |\hat{I}_{(0,\xi_i)}[f] - t_i|$ for all $i = 1, \ldots, n$.
15: **if** $\max_{i=1,\ldots,n} \varepsilon_i \le \varepsilon_u$ and $x_{i-1} \le \xi_i \le x_i$ for $i = 1, \ldots, n$ **then**
16: Stop (i.e. continue with line 20). ▷ u-error and monotonicity condition satisfied
17: **else**
18: Set $h \leftarrow 0.8\, h$ and try again (i.e. continue with line 9).
19: **end loop**
20: Set $h \leftarrow 1.3\, h$ if $\max \varepsilon_i \le \varepsilon_u/3$.
21: Store $\{c_j\}$, $\{u_j\}$, $\{x_n\}$, a_k, and F_k in table.
22: Set $h \leftarrow \min(h, b_r - (a_k - h))$ [take care of right boundary].
23: Set $k \leftarrow k + 1$, $a_k \leftarrow a_{k-1} + h$, and $F_k \leftarrow F_{k-1} + u_n$.
24: Create table for indexed search on $\{F_j\}$.

▷ Sampling

25: Generate $U \sim U(0, I)$.
26: Find interval J with $F_J \le U < F_{J+1}$ using indexed search.
27: Compute $X \leftarrow a_J + \texttt{NEval}(\{c_J\}, \{u_J\}, U - F_J)$.
28: **return** X.

5. IMPLEMENTATION AND COMPUTATIONAL EXPERIENCE

5.1 Implementation, Stability and Precision

We coded Algorithm NINIGL and added it as new method PINV to our C library UNU.RAN [Leydold and Hörmann 2008b] for random variate generation. Our major concerns were stability and reliability, that is, the algorithm should be able

to handle numerically difficult distributions and the maximal u-error should[3] not exceed the maximum tolerated error ε_u given by the user. So during the coding phase we tested the implementation for distributions of different shapes including Gaussian, Cauchy, beta, gamma, and t-distributions with various parameter settings. Then we computed the u-error at many points (up to 10^9) where we put one third of all test points close to 0 and 1, respectively. We observed that the quadrature rules became inaccurate and required many intervals when derivatives are large whereas the interpolation of the inverse CDF became numerically unstable when the density is close to zero and thus the CDF is flat. This is in particular a problem in the (far) tails of heavy-tailed distributions. Thus computing the cut-off points for the computational domain is a crucial part of the algorithm. Even for the gamma distribution with shape parameter 3 one cannot just use 0 for the left boundary. We experimented with many different versions of the cut-off procedure; finally we arrived at the simple, fast and stable method described above.

We also tested a version of the algorithm that uses the CDF (instead of the PDF) and which avoids the integration error (given that an accurate implementation of the CDF is available). However, this version was less robust and did not work for (very) small u-resolutions due to severe round-off errors when computing differences of CDF values.

We used the R Project for Statistical Computing [R Development Core Team 2008] as a convenient environment for doing stochastic simulations. Hence we have prepared package Runuran [Leydold and Hörmann 2008a] to make our UNU.RAN library accessible within R. This allows us to test our algorithms with CDF implementations that are independent from our C code. For moderate (or large) sample sizes the generation times of this R version is almost the same as for the C version.

Our final stability and precision tests were performed in R. We then calculated the maximal u-error for a total of 778 parameter sets for the gamma, beta and t-distribution and for $\varepsilon_u = 10^{-8}, 10^{-9}, \ldots, 10^{-13}$. To our own astonishment there was only one case where the maximal u-error was larger than requested: for gamma distribution with shape parameter $\alpha = 1.01$ and $\varepsilon_u = 10^{-13}$ the maximal error was $1.0033\,\varepsilon_u$; however, this density has unbounded derivatives at zero and thus we would expect possible problems. When we further decreased the u-resolution to 10^{-15} the observed maximal u-errors were not bounded by ε_u any more but close to machine precision. However, in that situation it happened for some distributions that the computation of the approximating polynomial failed entirely due to round-off errors.

5.2 Speed and Memory Consumption

As a second aim we wanted to design an algorithm that has a really fast marginal execution time but we were ready to accept a slow setup. Measuring and comparing the speed of random number generators is always disputable as it is influenced by many properties of the used computing environment. Thus we investigated two situations: We ran our experiments both in R and in a pure C version where we used the Rmath library from the R project (which provides the same quantile functions)

[3]Of course we cannot expect that it works for *every* distribution due to limitations of floating point arithmetic.

and the same (implementation of the) uniform random number generator (i.e., the Mersenne twister). We measured the total execution times (including setup) to generate 10^6 random variates by different methods in both environments on a laptop with Intel Core Duo 1.6Ghz chip, running Linux and using gcc-4.2.1. We tested normal, t, gamma, and beta distributions with various parameters (with bounded densities). We used order 5 (the default for method PINV in UNU.RAN) and u-resolutions between 10^{-8} and 10^{-13}.

In R the generation times for our algorithm was almost independent from the target distribution and increased slightly when ε_u was decreased. It turned out that these times were practically the same as for calling `runif()` and 25-30% faster compared to `rexp()` which generates exponential distributed random variates! (Compared to `log(runif())` it was three times a fast). Our proposed algorithm was much faster than the built-in quantile functions (using `q`⟨*dist*⟩`(runif())`): about 3 times for normal, Cauchy, exponential distributions, 50–100 times for gamma distributions and 80–120 times for beta distributions (with shape parameters greater than 1), 50–130 times for t-distributions for at least one degrees of freedom and greater than 400 if df is less[4] than 1. Our algorithm was often even several times faster than the R built-in random number generators (which are mostly not based on inversion). For the C implementation these results at quite similar (except for the fact that calling the uniform random number generator is in C about three times fast than in R; all speed up factors mentioned above remain more or less the same).

To get some more insight into the speed of the setup we also tried to find the sample size at which our algorithm has the same speed as R's quantile functions. It turned out that the break-even point was about 15 000 for the normal distribution and depending on the parameters between 300 and 700 for the gamma, beta and t-distributions. These results indicate that our algorithm is of course not competitive in the varying parameter case but even for moderate sized samples it is clearly faster than specialized algorithms.

The required number of intervals is also an important characteristic of the algorithm as it influences both the setup time and the size of the required table. Using the error-bound for interpolation which is $O(h^{n+1})$ for interval length h and order n it is obvious that the required number of intervals $O(1/\varepsilon_u^{n+1})$. This implies that for linear interpolation an error-reduction by a factor of $1/100$ requires about ten times the number of intervals. Therefore, linear interpolation is not useful if small error values are required as the table sizes explode. For order $n = 3$ an error-reduction by a factor of $1/100$ requires $\sqrt{10} = 3.16$ times the number of intervals, for $n = 5$ this factor is reduced to $\sqrt[3]{10} = 2.16$. In Table I we report the required number of intervals for some standard distributions and practically important values of ε_u. The results clearly illustrate the asymptotic considerations for the required number of intervals. They also indicate that the order $n = 5$ is enough to reach close to machine precision with a moderate number of intervals, never more than 400 for our examples. The necessary number of floating point constants for our algorithm without considering the guide-table is simple $2\,n$ times the required number of intervals. So we may conclude that for $n = 5$ and $\varepsilon_u = 10^{-12}$ we require a table

[4] `qt()` did not work for `df` less than 1 prior to R version 2.8.

Table I. Required number of intervals for different u-resolutions ε_u using polynomials of order 3 and 5, respectively.

ε_u	10^{-8}	10^{-10}	10^{-12}	10^{-8}	10^{-10}	10^{-12}
distribution	Order $n = 3$			Order $n = 5$		
Normal	173	517	1603	63	123	252
Cauchy	288	826	2504	112	203	393
Exponential	128	382	1192	44	87	176
Gamma(5)	177	526	1647	62	124	255
Beta(5,5)	155	477	1491	58	114	236
Beta(5,500)	178	527	1648	62	124	256

with not more than 4000 double constants, i.e. with a size of not more than 32 kilobyte. If higher precision is required (we consider it only necessary for computing environments with a smaller machine epsilon) it is necessary to increase the order n of the polynomials to keep the size of the tables moderate.

The results of Table I also indicate that the differences between the different distributions are not too large. The worst case of our examples is the Cauchy distribution whose heavy tails imply a large computationally relevant domain and thus many intervals. Otherwise the differences are small, monotone densities (like the exponential density) and densities without tail (like the Beta(5,5) density) require slightly less intervals than bell-shaped densities with two tails.

5.3 Non-standard Distributions

Of course the main advantage of an automatic algorithm is that it can be used for non-standard distributions. As the setup is fastest for distributions which have a density given by a simple expression we start with trying the hyperbolic distribution which is used in finance, see [Eberlein and Keller 1995]. Here only a quite slow specialized inversion method that reaches maximal u-errors around 10^{-7} is available in the literature, see [Leobacher and Pillichshammer 2002]. Our automatic algorithm has no problems with that density. Using UNU.RAN the marginal execution time is very fast, almost as fast as the fastest normal generator of UNU.RAN and about 3 times faster than the Box-Muller method or inversion for the normal distribution. It lies in the nature of table-based inversion that the setup is slow. On our computer the setup takes about as long as the marginal generation of 50,000 variates. Perhaps that sounds slow but on our Linux PC that means that we can simulate samples of size 10^5 of the hyperbolic distribution for 100 different parameter sets within one second and that is shorter than simulating 10^7 normal variates using the Box-Muller or the inversion method.

We tested our algorithm also for distributions with difficult densities, in particular for the generalized hyperbolic distribution [Barndorff-Nielsen and Blæsild 1983], for the noncentral χ^2-distribution [Johnson et al. 1995] and for the α-stable distribution [Nolan 2009]. For all of them we were able to apply our algorithm successfully and reached the required precision and very fast marginal execution times.

The setup times of course depend strongly on the implementation of the PDF. Using R and Runuran we obtained setup times of about 0.2 seconds for the generalized hyperbolic distribution which is certainly acceptable. Compared to the speed of the quantile functions implemented in two R-packages for generalized hyperbolic

distributions we observed speed-up factors clearly above 1000 when generating one million variates. For the noncentral χ^2-distribution the setup took about 0.05 seconds. Compared to using the built-in quantile function of R, our algorithm is about 10 000 times faster when generating one million variates. For the α-stable distribution the PDF implementation we found in an R package was slow. Also the fact that stable distributions have heavy tails increases the time of the setup. So we observed setup times of more than a minute for $\alpha > 1$. Still our method is much faster than using the quantile function of that R package when many random variates of the stable distribution should be generated by inversion. For $\alpha < 1$ the tails are so heavy that numerical inversion needs a lot of intervals and the setup gets really slow. Here a faster implementation of the PDF is required.

5.4 Comparison to Hermite inversion

We have pointed out in the introduction that the fast automatic inversion algorithms described in [Ahrens and Kohrt 1981; Ulrich and Watson 1987] are based on similar ideas as our algorithm. As they were not designed to reach a user-specified precision and as they are designed for assembler implementation they seem to be of little practical value for modern computing environments and we did not include them in our comparison. As linear interpolation despite its unbeatable simplicity is not capable to reach high precision with moderate tables the only real competitor for our new algorithm is our first numeric inversion algorithm HINV based on Hermite interpolation (see [Hörmann and Leydold 2003]; it is also implemented in our UNU.RAN library). A main difference to the new algorithm is that HINV requires the CDF and PDF for order $n = 3$ polynomials and also the derivative of the PDF for order $n = 5$; orders higher than 5 are not possible. A main reason for developing the new algorithm was that obtaining a precise implementation of the CDF is not easy for most important distributions. Using the CDF allows a simpler cut-off procedure and avoids possible integration errors, but interestingly it does not improve the stability of the algorithm. Especially in the right tail the numerical instabilities of HINV are larger than those of our new algorithm. This is underlined by the fact that we observed several cases with u-errors larger than ε_u (about five percent of all cases we computed) when we tested the u-error of HINV in the way described in Section 5.1 above. For some parameter values of the t-distribution and $\varepsilon_u = 10^{-13}$ HINV is not able to reach the required accuracy and decomposes the domain into a huge number of intervals which never happened for our new algorithm. The numeric instabilities come from the fact that in the far right tail the CDF is only calculated with a precision of 10^{-16} and the probabilities of the intervals are small. Numeric integration in our new algorithm improves that problem as we are calculating the CDF starting only with the left border of the current interval.

The marginal generation times of HINV and of the new algorithm are almost identical as the sampling algorithm is the same. The difference in the setup times is mainly caused by the relative speeds of the evaluations of the CDF and the PDF, respectively. In our experiments with the above distributions the setup of HINV was a bit faster than that of our new algorithm. For non-standard distributions with expensive PDFs the setup of the new algorithms is sometimes considerably faster than that of HINV. For the generalized hyperbolic distribution we observed

that the setup of our new algorithm was about 100 times faster than that of HINV.

Another advantage of the new algorithm is that it requires only about half of the number of intervals to reach the same precision.

6. CONCLUSIONS

We have explained all principles and the most important details of a fast numeric inversion algorithm for which the user provides only a function that evaluates the density and a typical point in its domain. It is the first algorithm of this kind in the literature that is based on a rigorous error control that works for all smooth bounded densities. Extensive numerical experiments showed that the new algorithm always reached the required precision for the Gamma, Beta and t-distribution and also for less well known distributions with computational difficult densities. For the fixed parameter situation our algorithm is by far the fastest inversion method known. Compared to the special inversion algorithms for the respective distributions we reached speed up factors between 50 and 100 for the standard distributions and above 1000 for important special distributions. This makes our algorithm especially attractive for the simulation of copulas and for quasi-Monte Carlo applications.

ACKNOWLEDGMENTS

The second author was supported by Boğaziçi-Research-Fund, Project 07HA301.

REFERENCES

Abramowitz, M. and Stegun, I. A., Eds. 1972. *Handbook of mathematical functions*, 9th ed. Dover, New York.

Ahrens, J. H. and Kohrt, K. D. 1981. Computer methods for efficient sampling from largely arbitrary statistical distributions. *Computing 26*, 19–31.

Barndorff-Nielsen, O. and Blæsild, P. 1983. Hyperbolic distributions. In *Encyclopedia of Statistical Sciences*, N. L. Johnson, S. Kotz, and C. B. Read, Eds. Vol. 3. Wiley, New York, 700–707.

Billingsley, P. 1986. *Probability and Measure*. Wiley & Sons, New York.

Bratley, P., Fox, B. L., and Schrage, E. L. 1983. *A Guide to Simulation*. Springer-Verlag, New York.

Butler, E. L. 1970. Algorithm 370: General random number generator [G5]. *Commun. ACM 13,* 1, 49–52.

Chen, H. C. and Asau, Y. 1974. On generating random variates from an empirical distribution. *AIIE Trans. 6*, 163–166.

Costantini, P. 1986. On monotone and convex spline interpolation. *Mathematics of Computation 46,* 173, 203–214.

Dahlquist, G. and Björck, Å. 2008. *Numerical methods in scientific computing*. Vol. 1. SIAM, Philadelphia, PA.

Devroye, L. 1986. *Non-Uniform Random Variate Generation*. Springer-Verlag, New-York.

Eberlein, E. and Keller, U. 1995. Hyperbolic distributions in finance. *Bernoulli 1*, 281–299.

Gander, W. and Gautschi, W. 2000. Adaptive quadrature–revisited. *BIT Numerical Mathematics 40,* 1, 84–101.

Hörmann, W. and Leydold, J. 2003. Continuous random variate generation by fast numerical inversion. *ACM Trans. Model. Comput. Simul. 13,* 4, 347–362.

Hörmann, W., Leydold, J., and Derflinger, G. 2004. *Automatic Nonuniform Random Variate Generation*. Springer-Verlag, Berlin Heidelberg.

Johnson, N. L., Kotz, S., and Balakrishnan, N. 1995. *Continuous Univariate Distributions*, 2nd ed. Vol. 2. Wiley, New York.

L'Ecuyer, P. 2008. *SSJ: Stochastic Simulation in Java.* Département d'Informatique et de Recherche Opérationnelle (DIRO), Université de Montréal. Version 2.1.1 (Oct. 7, 2008).

Leobacher, G. and Pillichshammer, F. 2002. A method for approximate inversion of the hyperbolic cdf. *Computing 69,* 4, 291–303.

Leydold, J. and Hörmann, W. 2008a. *Runuran – R interface to the UNU.RAN random variate generators, Version 0.8.* Department of Statistics and Mathematics, WU Wien, A-1090 Wien, Austria. `http://cran.r-project.org/`.

Leydold, J. and Hörmann, W. 2008b. *UNU.RAN – A Library for Non-Uniform Universal Random Variate Generation, Version 1.3.* Department of Statistics and Mathematics, WU Wien, A-1090 Wien, Austria. `http://statistik.wu-wien.ac.at/unuran/`.

Nolan, J. P. 2009. *Stable Distributions - Models for Heavy Tailed Data.* Birkhäuser, Boston. In progress, Chapter 1 online at academic2.american.edu/∼jpnolan.

Overton, M. L. 2001. *Numerical Computing with IEEE Floating Point Arithmetic.* SIAM, Philadelphia.

R Development Core Team. 2008. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Schwarz, H. R. 1997. *Numerische Mathematik*, 4th edition ed. Teuber, Stuttgart.

Tuffin, B. 1997. Simulation accélérée par les méthodes de Monte Carlo et quasi-Monte Carlo: théorie et applications. Ph.D. thesis, Université de Rennes 1.

Ulrich, G. and Watson, L. 1987. A method for computer generation of variates from arbitrary continuous distributions. *SIAM J. Sci. Statist. Comput. 8*, 185–197.