Gerhard Derflinger and Wolfgang Hörmann and Josef Leydold

Online Supplement to "Random Variate Generation by Numerical Inversion When Only the Density Is Known"

Paper

Original Citation:

This version is available at: http://epub.wu.ac.at/1290/
Available in ePubWU: September 2009

ePubWU, the institutional repository of the WU Vienna University of Economics and Business, is provided by the University Library and the IT-Services. The aim is to enable open access to the scholarly output of the WU.
Online Supplement to
“Random Variate Generation by Numerical Inversion When Only The Density Is Known”

Gerhard Derflinger, Wolfgang Hörmann, Josef Leydold

Department of Statistics and Mathematics
WU Wirtschaftsuniversität Wien

Research Report Series
Report 91
September 2009

http://statmath.wu.ac.at/
Online Supplement to

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

GERHARD DERFLINGER
WU (Vienna University of Economics and Business)
WOLFGANG HÖRMANN
BU Istanbul
and
JOSEF LEYDOLD
WU (Vienna University of Economics and Business)

This Online Supplement summarizes our computational experiences with Algorithm NINIGL presented in our paper “Random Variate Generation by Numerical Inversion when only the Density Is Known”. It is 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 nearly the same for all these 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. Thus our algorithm is 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

Author’s address: Gerhard Derflinger and Josef Leydold: Department of Statistics and Mathematics, WU (Vienna University of Economics and Business), Augasse 2-6, 1090 Vienna, Austria, email: Gerhard.Derflinger@wu.ac.at, Josef.Leydold@wu.ac.at
Wolfgang Hörmann: Department of Industrial Engineering, Boğaziçi University, 34342 Bebek-Istanbul, Turkey, email: hormann@boun.edu.tr
©ACM (2009). This is the author’s version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in ACM Transactions on Modeling and Computer Simulation, in press http://doi.acm.org/10.1145/nmnm nmnm.
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 attractive 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 (and there is a way to compute its inverse).

— 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 as one has to transform the uniformly distributed marginals of the copula into the marginals of the target distribution.

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 often not given in closed form and thus one must use numerical methods. In our paper “Random Variate Generation by Numerical Inversion when only the Density Is Known” we described such a method that works whenever the probability density function (PDF) of the distribution is given. We assumed that this density is bounded and sufficiently smooth. Nevertheless, in our experiments the algorithm also worked when the density or some of its derivatives are unbounded. The method is based on polynomial interpolation of the inverse CDF utilizing Newton’s formula together with adaptive Gauss-Lobatto integration.

In this Online Supplement we restate the complete algorithm for the sake of completeness (see Section 2). Section 3 reports our computational experiences. Again for the sake of completeness we added the general conclusions of the paper as Section 4.
2. THE ALGORITHM

2.1 Approximation Error and u-Resolution

Let $F$ denote the CDF of the distribution and let $F^{-1}_a$ denote an approximation to the inverse CDF. Then we define the $u$-error at a point $u \in (0,1)$ as

$$\varepsilon_u(u) = |u - F(F^{-1}_a(u))|.$$  

We are convinced that the $u$-error is a 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 a numeric inversion algorithm the $u$-resolution of the algorithm, denoted by $\varepsilon_u$. In the sequel we consider it as a control parameter for our numerical inversion algorithm, i.e., the user can provide $\varepsilon_u$ and the setup of our algorithm should guarantee that

$$\sup_{u \in (0,1)} |u - F(F^{-1}_a(u))| \leq \varepsilon_u.$$  

Notice, however, that $\varepsilon_u$ cannot be smaller than machine precision ($2^{-52} \approx 2.2 \times 10^{-16}$ for IEEE-754 compliant floating point arithmetic, see Overton [2001]).

2.2 Design of the Automatic Inversion Algorithm

The inversion algorithm can sample from a variety of different distributions. The user has to provide the following information:

— 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 $\varepsilon_u$.

2.3 The Algorithm

Algorithm NINIGL (Numerical Inversion with Newton Interpolation and Gauss-Lobatto integration) compiles all building blocks in a lean form. The required subroutines are given as Subroutine 1–4. $\hat{I}_{(a,b)}[f]$ denotes an approximation of the integral $\int_a^b f(x) \, dx$ as described in Sect. 3.3 (Gauss-Lobatto Quadrature) of the paper.

3. IMPLEMENTATION AND COMPUTATIONAL EXPERIENCES

We coded Algorithm NINIGL and added it as new method PINV to our C library UNU.RAN [Leydold and Hörmann 2009b] for non-uniform 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 not exceed the maximum tolerated error $\varepsilon_u$ given by the user. (Of course we cannot expect that it works for every distribution due to limitations of floating point arithmetic.) 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 2009a] to make our UNU.RAN library accessible within R. This allows us to test our algorithms with
Algorithm 1 NINIGL

**Input:** Density $f(x)$, center $x_c$ of distribution, $u$-resolution $\varepsilon_u$, order $n$.

**Output:** Random variate with approximate density $f$ and maximal $u$-error $\varepsilon_u$.

1. $\varepsilon_u \leftarrow 0.9 \varepsilon_u$. ▷ Adjust

2. Find points $\tilde{b}_l < x_c < \tilde{b}_r$ with $f(\tilde{b}_l) \approx f(\tilde{b}_r) \approx 10^{-13} f(x_c)$. ▷ Preprocessing

3. Roughly Estimate $I_0 \leftarrow \tilde{I}_{(\tilde{b}_l, \tilde{b}_r)}[f]$. ▷ Setup

4. Find cut-off points $b_l$ and $b_r$ for computational domain with

   $\text{Prob}(X < b_l) \approx \text{Prob}(X > b_r) \approx 0.05 I_0 \varepsilon_u$. Use recursion (10) of the paper.

5. Compute $I \leftarrow \text{AGL}(f, [b_l, b_r], \text{tol} = 0.05 I_0 \varepsilon_u)$.

   [Store all calculated subintervals and their CDF values in a table.] ▷ Setup

6. Set $a_0 = b_l$, $h = (b_r - b_l)/128$, $F_0 \leftarrow 0$, and $k \leftarrow 0$.

7. **while** $a_k < b_r$ **do** ▷ interpolating polynomial on $[a_k, a_k + h]$

   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 Equation (4) of the paper.

   10. Set $u_0 \leftarrow 0$, compute $u_i \leftarrow u_{i-1} + \tilde{I}_{(x_{i-1}, x_i)}[f]$ for all $i = 1, \ldots, n$.

      [Reuse table from Step 5 together with simple Gauss-Lobatto.]

   11. Compute coefficients $\{c_j\} \leftarrow \text{NCoef}(\{u_j\}, \{x_j\})$.

   12. Compute test points $\{t_i\} \leftarrow \text{NTest}(\{u_j\})$.

   13. Compute $\xi_i \leftarrow \text{NEval}(\{c_j\}, \{u_j\}, t_i) \leftarrow F_n^{-1}(t_i)$ for all $i = 1, \ldots, n$.

   14. Compute $\varepsilon_i \leftarrow |\tilde{I}_{(0, \xi_i)}[f] - t_i|$ for all $i = 1, \ldots, n$.

   15. **if** $\max_{i=1, \ldots, n} \varepsilon_i \leq \varepsilon_u$ and $x_{i-1} \leq \xi_i \leq x_i$ for $i = 1, \ldots, n$ **then**

      16. Exit loop (goto line 20). ▷ $u$-error and monotonicity condition satisfied

   **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 $\max \varepsilon_i \leq \varepsilon_u / 3$.

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 $U \sim U(0, 1)$.

26. Find interval $J$ with $F_J \leq U < F_{J+1}$ using indexed search.

27. Compute $X \leftarrow a_J + \text{NEval}(\{c_j\}, \{u_j\}, U - F_J)$.

28. **return** $X$.

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 tests were performed on distributions of different shapes including Gaussian, Cauchy, beta, gamma, and $t$-distributions with various parameter settings.
Routine 1 \texttt{NCoef} (Newton-Coefficients)
\textbf{Input:} Nodes \(x_0 < \ldots < x_n\), values \(g(x_0), \ldots, g(x_n)\).
\textbf{Output:} Coefficients \(c_0, \ldots, c_n\) for interpolating polynomial \(P_n\).
\begin{enumerate}
\item for \(i = 0, \ldots, n\) do
\item \(c_i \leftarrow g(x_i)\).
\item for \(k = 1, \ldots, n\) do
\item \(c_i \leftarrow (c_i - c_{i-1})/(x_i - x_{i-k})\).
\item return \(c_0, \ldots, c_n\).
\end{enumerate}

Routine 2 \texttt{NEval} (Newton-Evaluate)
\textbf{Input:} Coefficients \(c_k\) of \(P_n\), nodes \(x_0, \ldots, x_n\), point \(x \in [x_0, x_n]\).
\textbf{Output:} Value of \(P_n(x)\).
\begin{enumerate}
\item \(p \leftarrow c_n\).
\item for \(k = n-1, n-2, \ldots, 0\) do
\item \(p \leftarrow c_k + (x - x_k)p\).
\item return \(p\).
\end{enumerate}

Routine 3 \texttt{NTest} (Newton-Testpoints)
\textbf{Input:} Nodes \(u_0 < \ldots < u_n\).
\textbf{Output:} Test points \(t_1 < \ldots < t_n\).
\begin{enumerate}
\item for \(i = 1, \ldots, n\) do
\item \(t_i \leftarrow (u_{i-1} + u_i)/2\).
\item for \(j = 1, 2\) do \hspace{1cm} \(\triangleright 2\) Newton steps
\item \(s \leftarrow 0\), \(sq \leftarrow 0\).
\item for \(k = 0, \ldots, n\) do
\item \(s \leftarrow s + 1/(t_i - u_k)\), \(sq \leftarrow sq + 1/(t_i - u_k)^2\).
\item \(t_i \leftarrow t_i + s/sq\).
\item return \(t_1, \ldots, t_n\).
\end{enumerate}

Routine 4 \texttt{AGL} (Adaptive-Gauss-Lobatto)
\textbf{Input:} Density \(f(x)\), domain \([a, a + h]\), tolerance \(tol\).
\textbf{Output:} \(\int_a^{a+h} f(x) dx\) with estimated maximal error less than \(tol\).
\begin{enumerate}
\item \(I_0 \leftarrow \hat{I}_{(a,a+h)}[f]\).
\item \(I_1 \leftarrow \hat{I}_{(a,a+h/2)}[f] + \hat{I}_{(a+h/2,a+h)}[f]\).
\item if \(|I_0 - I_1| < tol\) then
\item return \(I_1\).
\item else
\item return \((\text{AGL}(f, (a, a+h/2), tol) + \text{AGL}(f, (a + h/2, a + h), tol))\).
\end{enumerate}
3.1 Stability and Accuracy

In our extensive tests we observed that the (adaptive) quadrature rule became inaccurate and required many intervals only if the derivative of the density is very large (unbounded).

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 our procedure for computing the cut-off points for the computational domain as described in Sect. 3.4 (Cut-off Points for the Computational Domain) of the paper 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.

Our final stability and accuracy tests were performed for a total of 778 parameter sets for the gamma, beta and $t$-distribution and for $\varepsilon_u = 10^{-8}, 10^{-9}, \ldots, 10^{-13}$. The $u$-error was evaluated for $10^9 u$-values; one third of them was selected close to 0 and 1, respectively, for testing the accuracy of our interpolation especially in the tails. 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 $\varepsilon_u$ 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.

**Remark 1.** 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.

3.2 Speed

Measuring the speed of random number generators is always disputable as it is influenced by many properties of the used computing environment. Thus we used R's quantile functions for comparison. We investigated two situations: (1) the total execution times (including setup) to generate $10^6$ random variates, and (2) the sample size at which our algorithm has the same speed. All experiments were performed 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) and the same (implementation of the) uniform random number generator (i.e., the Mersenne twister). 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 were almost independent from the target distribution and increased slightly when $\varepsilon_u$ was decreased. It turned out that these times were practically the same as calling the uniform random number generator `runif()` and three times faster than `log(runif())`. They were much faster than the built-in quantile functions (using `q(dist)(runif())`): about 3 times for
Table I. Required number of intervals for different $u$-resolutions $\varepsilon_u$ using polynomials of order 1, 3 and 5, respectively.

<table>
<thead>
<tr>
<th>distribution</th>
<th>$\varepsilon_u$</th>
<th>$10^{-8}$</th>
<th>$10^{-10}$</th>
<th>$10^{-8}$</th>
<th>$10^{-10}$</th>
<th>$10^{-12}$</th>
<th>$10^{-8}$</th>
<th>$10^{-10}$</th>
<th>$10^{-12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n = 1$</td>
<td></td>
<td></td>
<td>$n = 3$</td>
<td></td>
<td></td>
<td>$n = 5$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normal</td>
<td>12620</td>
<td>118294</td>
<td>173</td>
<td>517</td>
<td>1603</td>
<td>63</td>
<td>123</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>Cauchy</td>
<td>19512</td>
<td>193558</td>
<td>288</td>
<td>826</td>
<td>2504</td>
<td>112</td>
<td>203</td>
<td>393</td>
<td></td>
</tr>
<tr>
<td>Exponential</td>
<td>10914</td>
<td>108882</td>
<td>128</td>
<td>382</td>
<td>1192</td>
<td>44</td>
<td>87</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>Gamma(5)</td>
<td>11890</td>
<td>121602</td>
<td>177</td>
<td>526</td>
<td>1647</td>
<td>62</td>
<td>124</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>Beta(5,5)</td>
<td>11272</td>
<td>99702</td>
<td>155</td>
<td>477</td>
<td>1491</td>
<td>58</td>
<td>114</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>Beta(5,500)</td>
<td>11874</td>
<td>11130</td>
<td>178</td>
<td>527</td>
<td>1648</td>
<td>62</td>
<td>124</td>
<td>256</td>
<td></td>
</tr>
</tbody>
</table>

normal, Cauchy, and 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 degree of freedom and greater than 400 if df is less than 1. Our algorithm was often even several times faster than the R built-in random variate generators (which are mostly not based on inversion). We got quite similar speed-up factors and timing results for our C implementation. (The only remarkable exception is the fact that calling the uniform random number generator in C is about three times faster than in R.)

For situation (2) the break-even points for the sample sizes are about 15,000 for the normal distribution, and between 300 and 700 for 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.

3.3 Memory Consumption

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 is $O(1/\varepsilon_u^{n+1})$. 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 $\sqrt[5]{10} = 2.16$. In Table I we report the required number of intervals for some standard distributions and practically important values of $\varepsilon_u$. These results clearly illustrate the asymptotic considerations for the required number of intervals. They also indicate that order $n = 5$ is enough to reach close to machine precision with a moderate number of intervals. The differences between 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.
3.4 Non-standard Distributions

The main advantage of an automatic algorithm is that it can be used for non-standard distributions. An example with a quite simple density is 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]. The proposed algorithm works well with this density. The marginal execution times hardly depend on the distribution and are about three times faster than generating normal variates with the method of Box and Muller [1958]. On our computer the setup takes about as long as the marginal generation of 50,000 variates. This means that we can simulate samples of size $10^5$ of the hyperbolic distribution for 100 different parameter sets within one second which is faster than simulating $10^7$ normal variates using the Box Muller method.

Similarly our algorithm also worked for distributions with difficult densities, in particular for the generalized hyperbolic distribution [Barndorff-Nielsen and Blæsild 1983], for the noncentral $\chi^2$-distribution ([Fisher 1928], see [Johnson et al. 1995]) and for the $\alpha$-stable distribution ([Lévy 1925], see [Nolan 2010] for a recent survey). The setup times of course depend strongly on the implementation of the respective densities. 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 $\chi^2$-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 $\alpha$-stable distribution implementations of the density are quite expensive. Moreover stable distributions have heavy tails. Thus 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 requires a lot of intervals and thus the setup gets really slow. Here a faster implementation of the PDF is required.

Remark 2. For the sake of completeness we note that simple exact algorithms for sampling from these distributions exist. However, they are not based on inversion and their marginal generation times are slower than that of our algorithm. Chambers et al. [1976] propose a generator for $\alpha$-stable distributions. An algorithm by Dagpunar [1989] can be used for generalized hyperbolic distributions. Noncentral $\chi^2$ distributed random variates can be generated by decomposing it as a Poisson mixture of central $\chi^2$ distributed random variates, see (29.5b-c) in [Johnson et al. 1995].

4. 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 an 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 in particular attractive for the simulation of marginal distributions, when using copula models, and for quasi-Monte Carlo applications.

ACKNOWLEDGMENTS
The authors gratefully acknowledge the useful suggestions of the area editor and two anonymous referees that helped to improve the presentation of the paper. The second author was supported by Boğaziçi-Research-Fund, Project 07HA301.

REFERENCES


