An Automatic Code Generator for Nonuniform Random Variate Generation ⋆

Josef Leydold [a], Gerhard Derflinger [a], Günter Tirler [a], and Wolfgang Hörmann [b]

[a] *University of Economics and Business Administration, Department for Applied Statistics and Data Processing, Augasse 2-6, A-1090 Vienna, Austria.*

[b] *IE Department, Boğaziçi University Istanbul, 80815 Bebek-Istanbul, Turkey*

Abstract

There exists a vast literature on nonuniform random variate generators. Most of these generators are especially designed for a particular distribution. However in pratice only a few of these are available to practioners. Moreover for problems as (e.g.) sampling from the truncated normal distribution or sampling from fairly uncommon distributions there are often no algorithms available. In the last decade so called universal methods have been developed for these cases. The resulting algorithms are fast and have properties that make them attractive even for standard distributions.

In this contribution we describe the concept of *Automatic random variate generation* where these methods are used to produce a single piece of code in a high level programming language. Using a web-based front-end to such a program this is an easy-to-use source for researchers and programmers for high quality generators for a large class of distributions. Using our UNURAN library we have implemented such a system, which is accessable at `http://statistik.wu-wien.ac.at/anuran`.

Key words: nonuniform random variate generation, universal algorithm, automatic code generator, transformed density rejection, continuous distribution
PACS: 02.70.-c, 02.70.Tt

⋆ This work was supported by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung*, Proj. 12805-MAT and by the Austrian Academy of Science, APART-scholarship.

Email addresses: `Josef.Leydold@statistik.wu-wien.ac.at` (Josef Leydold), `Wolfgang.Hoermann@statistik.wu-wien.ac.at` (Wolfgang Hörmann).

URLs: `http://statistik.wu-wien.ac.at/staff/leydold` (Josef Leydold), `http://statistik.wu-wien.ac.at/staff/hoermann` (Wolfgang Hörmann).

1 Introduction

In the last decades high quality generators for nonuniform random variates have been developed (see e.g. Devroye (1986), Dagpunar (1988), Fishman (1996), or Gentle (1998) for surveys). However searching through the literature on simulation (e.g. Tyszer (1999)) or web based software repositories (like `http://gams.nist.gov/` or `http://www.netlib.org/`) one finds only simple generators. Even for the normal distribution one finds such infamous methods like the sum of 12 uniform random numbers (sic!). Moreover for problems as (e.g.) sampling from the truncated normal distribution there are often no algorithms provided. When generators for fairly uncommon distributions are required the help of an expert is necessary.

During the last decade so called *universal* algorithms have been developed to avoid the design of special algorithms for simulation situations where the application of standard distributions is not adequate. These universal methods work for large classes of distributions and require the knowledge of some data about the desired distribution. Often no more than (a multiple of) the probability density function is necessary. Depending on the chosen method additional information like the (approximate) mode of the distribution, (approximate) regions of monotonicity or log-concavity, of the density function are required or useful. Obviously these universal methods need some setup step, in opposition to special generators, e.g., to the Box-Muller method (Box and Muller 1958). However we always can select between a fast setup step and slow marginal generation times or (very) fast marginal generation times at the expense of a time consuming setup step.

Although originally motivated to sample from non-standard distributions these methods have advantages that make them attractive even to sample from standard distributions. In Sect. 2 we decribe *transformed density rejection* (TDR), one of the most efficient universal algorithms, and demonstrate on this example the power of such algorithms. (Other universal algorithms are, e.g., `arou` introduced by Leydold (2000) or `table` by Ahrens (1993); see Leydold and Hörmann (2001) for a short survey.)

Universal methods like TDR provide high quality generators even for non-standard distributions. However implementing such algorithms in a reliable and robust way result in rather complex programs. Installing and using such a program might seem too tedious for “just a random number generator”. To improve the accessibility of these generators for practioners we suggest the concept of *Automatic code generator for nonuniform random variate generation* in Sect. 3. Instead of a library containing routines for both the setup step and the generation part a program is used to produce a single piece of code in a high level programming language. Using a web-based front-end to such a

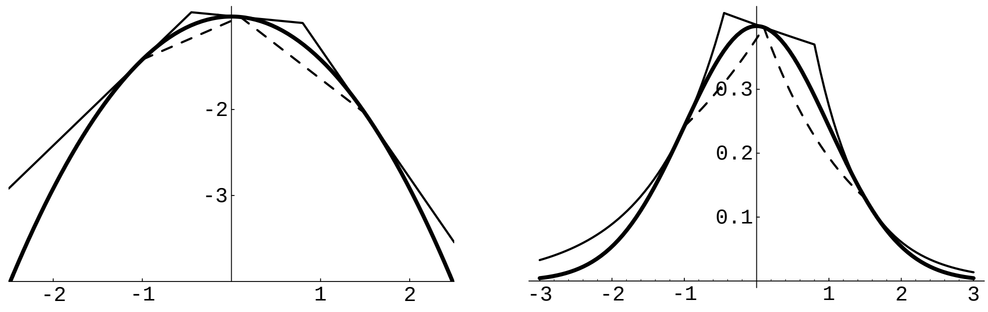

Fig. 1. Construction of a hat function for the normal density utilizing transformed density rejection. The left hand side shows the transformed density with three tangents. The right hand side shows the density function with the resulting hat. Squeezes are drawn as dashed lines.

program interested researchers or programmers have an easy to use source for high quality generators for many standard and non-standard distributions. By this concept it is possible to reduce the gap between theorists who create new algorithms and programmers or researches in different scientific areas who need state-of-the-art generators (or simply any generator for an uncommon simulation problem).

We have implemented such a system using our UNURAN library which is accessable at `http://statistik.wu-wien.ac.at/anuran`. As first version this system delivers source code in C, C++, Java, or FORTRAN for generators for continuous T-concave distributions.

2 Transformed Density Rejection

TDR ist the most flexible method. It is an acceptance/rejection method and has been introduced under a different name in Gilks and Wild (1992), and generalized in Hörmann (1995). It is based on the idea that the given density f is transformed by a strictly montonically increasing differentiable transformation $T\colon (0,\infty) \to \mathbb{R}$ such that $T(f(x))$ is concave. We then say that f is *T-concave*; log-concave densities are examples with $T(x) = \log(x)$.

By the concavity of $T(f(x))$ it is easy to construct a majorizing function for the transformed density as the minimum of several tangents. Transforming this function back into the original scale we get a hat function $h(x)$ for the density f. By using secants between the touching points of the tangents of the transformed density we analogously can construct squeezes; details can be found in Hörmann (1995). Figure 1 illustrates the situation for the standard normal distribution and $T(x) = \log(x)$. Evans and Swartz (1998) have shown that this technique is even suitable for arbitrary densities provided that the inflection points of the transformed density are known. It should be noted here that the tangent on the transformed density can be replaced by secants through two points that are close together, shifted away from the mode by the distance of these two points. Thus no derivatives are required.

Algorithm `tdr` applies this idea for a universal algorithm. The I_j are the intervals where the hat h is given by the tangent with touching point c_j.

Algorithm **1** `tdr`

```
Require: density f(x); transformation T(x), construction points c_1, ..., c_n.
   /* Setup */
1: Construct hat h(x) and squeeze s(x).
2: Compute intervals I_1, ..., I_n.
3: Compute areas H_j below the hat for each I_j.
   /* Generator */
4: loop
5:    Generate I with probability vector proportional to (H_1, ..., H_n).
6:    Generate X with density proportional to h|_I (by inversion).
7:    Generate U ~ U(0,1).
8:    if U h(X) <= s(X) return X. /* evaluate squeeze */
9:    if U h(X) <= f(X) return X. /* evaluate density */
```

Step 5 is executed in constant time by means of *indexed search* (Chen and Asau 1974). Notice that then the random variate X in step 6 is generated by inversion, when random numbers are *recycled* (Devroye 1986, §II.3, p.58) and the algorithm is implemented properly.

It is obvious that the transformation T must have the property that the area below the hat is finite, and that generating a random variable with density proportional to the hat function by inversion must be easy (and fast). Thus we have to choose the transformations T carefully. Hörmann (1995) suggests the family T_c of transformations, where

$$T_0(x) = \log(x) \quad \text{and} \quad T_c(x) = \operatorname{sign}(c)\, x^c. \tag{1}$$

($\operatorname{sign}(c)$ makes T_c increasing for all c.) For densities with unbounded domain we must have $c \in (-1, 0]$. For the choice of c it is important to note that the area below the hat increases when c decreases. Moreover we find that if a density f is T_c-concave, then f is $T_{c'}$-concave for every $c' \leq c$ (Hörmann 1995).

Because of computational reasons, the choice of $c = -1/2$ (if possible) is suggested (this choice includes all log-concave distributions). Table 1 gives examples of $T_{-1/2}$-concave distributions.

Distribution	Density	Support	$T_{-1/2}$-concave for
Normal	$e^{-x^2/2}$	$\mathbb{R}$	
Log-normal	$1/x \exp(-\ln(x-\mu)^2/(2\sigma^2))$	$[0,\infty)$	$\sigma \leq \sqrt{2}$
Exponential	$\lambda\, e^{-\lambda\, x}$	$[0,\infty)$	$\lambda > 0$
Gamma	$x^{a-1}\, e^{-b\, x}$	$[0,\infty)$	$a \geq 1,\, b > 0$
Beta	$x^{a-1}\,(1-x)^{b-1}$	$[0,1]$	$a, b \geq 1$
Weibull	$x^{a-1} \exp(-x^a)$	$[0,\infty)$	$a \geq 1$
Perks	$1/(e^x + e^{-x} + a)$	$\mathbb{R}$	$a \geq -2$
Gen. inv. Gaussian	$x^{a-1} \exp(-bx - b^*/x)$	$[0,\infty)$	$a \geq 1,\, b, b^* > 0$
Student's t	$(1 + (x^2/a))^{-(a+1)/2}$	$\mathbb{R}$	$a \geq 1$
Pearson VI	$x^{a-1}/(1+x)^{a+b}$	$\mathbb{R}$	$a, b \geq 1$
Cauchy	$1/(1+x^2)$	$\mathbb{R}$	
Planck	$x^a/(e^x - 1)$	$[0,\infty)$	$a \geq 1$
Burr	$x^{a-1}/(1+x^a)^b$	$[0,\infty)$	$a \geq 1,\, b \geq 2$
Snedecor's F	$x^{m/2-1}/(1+m/n\, x)^{(m+n)/2}$	$[0,\infty)$	$m, n \geq 2$

Table 1
Examples of $T_{-1/2}$-concave densities (normalization constants omitted).

Construction Points

Algorithm `tdr` works well when the area between hat $h(x)$ and squeeze $s(x)$ is small compared to the area below the density function, i.e., when the ratio

$$\varrho = \frac{\int h(x)\, dx}{\int s(x)\, dx} = \text{area below hat/area below squeeze} \tag{2}$$

is close to one. In fact the performance of the algorithm can be controlled by this single parameter. If $\varrho \approx 1$ then the algorithms is close to inversion and shares most of its good properties but is much faster. In fact, then the marginal generation time hardly depends on the density function and is faster than many of the specialized algorithms (Leydold and Hörmann 2001). Moreover the quality of the generated random numbers only depends on the underlying uniform pseudo random number generator. The problem of finding good construction points c_j for the hat function is crucial for the performance of the algorithm. For n construction points in a compact interval with equal distances we have $\varrho = 1 + O(1/n^2)$ (Leydold and Hörmann 1998), which must also hold for n optimal construction points. For practical use several methods have been suggested:

Adaptive rejection sampling (ARS). Gilks and Wild (1992) introduce the ingenious concept of *adaptive rejection sampling.* For TDR it works in the following way: Start with (at least) two points on either side of the mode and sample points x from the hat distribution. Add a new construction point at x whenever the density $f(x)$ has to be evaluated, i.e., when $s(X) < U\, h(X)$, until the target ratio ϱ or the maximal number of construction points is reached. Obviously the ratio ϱ is a random variable that converges to 1 almost surely when the number n of construction points tends to infinity.

Equidistributed points. Leydold (2000) suggests the following rule of thumb for construction points:

$$c_i = \tan(-\pi/2 + i\,\pi/(n+1)), \quad i = 1, \dots, n. \tag{3}$$

If the mode m of the distribution is known, the points $c_i + m$ are prefered. Numerical simulations with several density functions have shown that this heuristic rule gives an acceptable good choice for construction points for "usually shaped" densities. These points are at least good starting points for adaptive rejection sampling.

Optimal construction points. There exist methods for finding construction points for TDR such that ϱ is minimized for given number of construction points, transformation and distribution. Hörmann (1995) describes a method how three optimal construction points can be found. If more points are required, Derflinger, Hörmann, and Tirler (2001) describe a very efficient method.

Derandomized adaptive rejection sampling (DARS). Adaptive rejection sampling has a stochastic component that makes it difficult to predict the actual number of constructions points for a given target ratio ϱ, which might be large with low probability. Moreover it is necessary to recompute the guide table for index search each time a construction point is added. Using optimal construction points, on the other hand, is the most expensive setup technique. Moreover it is not easy to estimate the necessary number of optimal construction points for the aimed ratio ϱ. The equidistribution rule, at last, performs terrible for, e.g., the normal distribution with standard deviation 10^{-5}.

We have made several experiments to solve this problem of computing good construction points for a hat function with a given ration ϱ, such that the number of points is not too large. As a fast *new* method we suggest a derandomized version of adaptive rejection sampling:

1. Start with a proper set of construction points as for adaptive rejection sampling (two arbitrary points on either side of the mode, optimal construction points or the equidistribution rule).
2. Each interval I, where the area between hat and squeeze is too large, is splitted by inserting a new construction point. As threshold values the average area over all intervals or a constant fraction of it is used. For finding splitting points the following rules can be used:
   (a) the expected new construction point in adaptive rejection sampling, i.e.,

$$c_{\exp} = \frac{\int_I x\,(h(x) - s(x))\,dx}{\int_I h(x) - s(x)\,dx} \tag{4}$$

(b) the "arc-mean" of the boundary of the interval (x_l, x_r), i.e.,

$$c_{\text{arc}} = \tan((\arctan(x_l) + \arctan(x_r))/2), \tag{5}$$

where $\arctan(\pm\infty)$ is set to $\pm\pi/2$.

(c) the mean of the boundary of the interval (x_l, x_r), i.e.,

$$c_{\text{mean}} = (x_l + x_r)/2. \tag{6}$$

Notice that rules (a) (for $c \leq -1/2$) and (c) can only be applied for bounded intervals. It is recommended to use different rules if one of these splitting points cannot be used due to numerical problems due to round-off error when computing the parameters of the new intervals.

3. If ϱ is larger than the desired bound repeat step 2 with updated threshold value. Otherwise we are done.

3 Automatic Code Generator

We have implemented TDR in a library called UNURAN (Universal NonUniform RANdom variate generators), see Leydold et al. (2001). This library provides routines both for the setup step as well as for the sampling algorithm. However implementing universal algorithms in a flexible, reliable, and robust way result in rather large computer code. Installing and using such a library might seem too tedious for "just a random number generator" at a first glance, especially when only a generator for a particular distribution is required. The complexity of such a library arises from the setup step, from parts performing adaptive steps, and from checking the data given by the user, since not every method can be used for every distribution. The sampling routine itself, however, is very simple and consists only of a few lines of code.

By the concept of *Automatic code generator for random variate generation* we can avoid to install and use such library. Instead we use the setup routines of such universal generators and produce a single piece of code in a high level language, e.g., in C, C++, Fortran, or Java. A program then can be used to create the source code of a generator for a particular distribution. Using a simple interface this can be done even by a researcher or programmer with hardly any knowledge about random variate generation. Fig. 2 gives an example of the resulting code. (This is only a simplified output. For fast generators the data block is larger.)

Using a web-based front-end to such a program we have an easy-to-use source for high quality generators for a large class of distributions, including standard and non-standard distributions. Then it is not even necessary to install this program. Now a distribution has to be selected from a menu or, alternatively,

```c
/* --------------------------------------------------------------- */
/* Generator for truncated gamma distribution.                     */
/* --------------------------------------------------------------- */
/*   PDF(x) = ((x-gamma)/beta)^(alpha-1) * exp( -(x-gamma)/beta )  */
/*       alpha = 5                                                 */
/*       beta = 3                                                  */
/*       gamma = 0                                                 */
/*       domain = [ 5, inf ]                                       */
/* (normalization constant omitted. PDF formula might be truncated! */
/* --------------------------------------------------------------- */
/* Generated by ANURAN (v.0.1.3).   November 20 2001 15:26:40      */
/*                                                                 */
/* Copyright (c) 2001   Wolfgang Hoermann and Josef Leydold        */
/* Dept. for Statistics, University of Economics, Vienna, Austria  */
/* All rights reserved.                                            */
/* --------------------------------------------------------------- */

/* --------------------------------------------------------------- */
/* Uniform (pseudo-)random number generator.                       */
/* Define the uniform random number generator of your choice here. */

#define uniform()   your_uniform_rng()

/* --------------------------------------------------------------- */
/* PDF for gamma distribution.                                     */

static double pdf_gamma (double x)
{
   /* parameters for PDF */
   const double alpha = 5.0;
   const double beta = 3.0;
   const double LOGNORMCONSTANT = 4.27666611901605531187e+00;

   /* compute PDF */
   x = (x-gamma) / beta;
   return ( (x<0.) ? 0. : exp( (alpha-1.)*log(x) - x - LOGNORMCONSTANT) );
}

/* --------------------------------------------------------------- */
/* Sampling from gamma distribution.                               */

double rand_gamma (void)
{
   /* data */
   const int guide_size = 8;
   const int guide[8] = { 0, 2, 2, 2, 3, 3, 3, 3 };
   const double Atotal = 1.35780537416445290511e+00;
   const struct { double x;
                  double Tfx;
                  double dTfx;
                  double sq;
                  double Acum;
                  double Ahatr;
   } iv[4] = {{ 5.00000000000000000000e+00, -7.02879403304335870217e+00,
                1.64005194104344909078e+00, 9.36117048555679565069e-01,
                1.69556217925627000787e-02, 1.69556217925627000787e-02 },
              { 6.70520562368709605039e+00, -5.19306218985333867266e+00,
                6.83453971482107180968e-01, 9.02681713211720415657e-01,
                1.08931144861056691808e-01, 5.90770476864658569682e-02 },
              { 1.00990195135927720571e+01, -4.03034053839437511613e+00,
                1.26441296384459056501e-01, 6.98221531718896337715e-01,
                5.69585332001876776253e-01, 3.40622691891395212860e-01 },
              { 2.02474280162066868627e+01, -5.44160157657758869476e+00,
                -3.69423195049578101390e-01, 0.00000000000000000000e+00,
                1.35780537416445290511e+00, 4.97449615466715733270e-01 } };

   /* code */
   int I;
   double U, V, X, Thx;

   while (1) {
      U = uniform();
      I =  guide[(int) (U * guide_size)];
      U *= Atotal;
      while (iv[I].Acum < U) I++;
         U -= iv[I].Acum - iv[I].Ahatr;
         X = iv[I].x + (U * iv[I].Tfx * iv[I].Tfx) / (1.-iv[I].Tfx*iv[I].dTfx*U);
         V = uniform();
         if (V <= iv[I].sq) return X;
         Thx = iv[I].Tfx + iv[I].dTfx * (X - iv[I].x);
         V /= Thx*Thx;
         if (V <= pdf_gamma(X)) return X;
      }
   }
}
```

Fig. 2. Example for an automatically created C code for a random variate generator.

has to be described by its probability density function given as ASCII-string, e.g., `exp(-2*sqrt(3+x^2)+x)`. Optional data about the distribution (e.g., its domain) or the application of the generator can be added to the input form. The result is then a (relatively short) subroutine for a random variate generator coded in some high level programming language that can be used in ones program by simple cut & paste.

The advantages of such an approach are clear:

- There is no necessity to install and handle a library.
- The complexity of the library is hidden (at the cost that complex tasks cannot be realized via this interface, of course).
- It can be used by a practitioner who has little or no experiences with nonuniform random number generation.
- It is easy to produce source code for generators in several programming languages.
- The resulting code is much smaller than the library.
- A data file with the output of this generator can be appended to verify the code on the target platform. Because every code is tested one will not be confused by obscure error messages and program crashes that makes the usage of external code sometimes so tedious.

By this concept it is possible to reduce the gap between theorists who create new algorithms and researchers in different scientific areas who need state-of-the-art generators (or simply any generator for an uncommon simulation problem).

We have implemented such a system using the UNURAN library. It is accessable at `http://statistik.wu-wien.ac.at/anuran`.

References

Ahrens, J. H. (1993). Sampling from general distributions by suboptimal division of domains. *Grazer Math. Berichte 319*, 20 pp.

Box, G. E. P. and M. E. Muller (1958). A note on the generation of random normal deviates. *Annals of Math. Stat. 29*(2), 610–611.

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

Dagpunar, J. (1988). *Principles of Random Variate Generation.* Oxford, U.K.: Clarendon Oxford Science Publications.

Derflinger, G., W. Hörmann, and G. Tirler (2001). The optimal selection of hat functions for rejection algorithms. in preparation.

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

Evans, M. and T. Swartz (1998). Random variable generation using concavity properties of transformed densities. *Journal of Computational and Graphical Statistics 7*(4), 514–528.

Fishman, G. S. (1996). *Monte Carlo. Concepts, algorithms, and applications.* Springer Series in Operations Research. New York: Springer.

Gentle, J. E. (1998). *Random Number Generation and Monte Carlo Methods.* Statistics and Computing. New York: Springer.

Gilks, W. R. and P. Wild (1992). Adaptive rejection sampling for Gibbs sampling. *Applied Statistics 41*(2), 337–348.

Hörmann, W. (1995). A rejection technique for sampling from T-concave distributions. *ACM Trans. Math. Software 21*(2), 182–193.

Leydold, J. (2000). Automatic sampling with the ratio-of-uniforms method. *ACM Trans. Math. Software 26*(1), 78–98.

Leydold, J. and W. Hörmann (1998). A sweep-plane algorithm for generating random tuples in simple polytopes. *Mathematics of Computation 67*(224), 1617–1635.

Leydold, J. and W. Hörmann (2001). Universal algorithms as an alternative for generating non-uniform continuous random variates. In G. I. Schuëller and P. D. Spanos (Eds.), *Monte Carlo Simulation*, pp. 177–183. A. A. Balkema. Proceedings of the International Conference on Monte Carlo Simulation 2000.

Leydold, J., W. Hörmann, E. Janka, and G. Tirler (2001). *UNU.RAN User Manual.* Augasse 2–6, A-1090 Wien, Austria: Institut für Statistik, WU Wien. available from `http://statistik.wu-wien.ac.at/unuran/`.

Tyszer, J. (1999). *Object-Oriented Computer Simulation of Discrete-Event Systems.* Boston, Dordrecht, London: Kluwer Academic Publishers.