Non-Uniform Random Variate Generation

Concepts and Applications

Wolfgang Hörmann, Josef Leydold, Gerhard Derflinger

hormannw@boun.edu.tr

Institut für Statistik, WU Wien and IE-Department, Bogazici University Istanbul
Non-Uniform Random Variate Generation

Purpose:
Generate a sequence $X_i$ of IID random variates with given distribution.
Independence can be dropped for some applications (MCMC).

Solution:
Transform sequence $U_i$ of IID $U(0,1)$ random numbers into sequence $X_i$.

$$u_1, u_2, u_3, u_4, \ldots \longrightarrow x_1, x_2, x_3, \ldots$$

Transformations need not be one-to-one.
Assumptions

- Have a source of **perfect** (real) uniform random numbers.
  
  Not possible in practice!!
  
- Real numbers.
  
  Computer uses **Floating Point Arithmetic**!
Uniform Random Numbers

- Pseudo-random numbers. Physical devices are very slow and have other problems.
- Have restricted resolution. Typically $10^{-9}$.
- Always use generators designed by experts. No selfmade combined generators.
- Test generator, if possible. Use a test similar to application. E.g. Marsaglia’s DIEHARD testsuite, or NIST testsuite, http://www.nist.gov/rng
Uniform Random Numbers

Properties of good generators

- Long period.
- Strong theoretical support.
- Passes many empirical tests.

See http://random.mat.sbg.ac.at/
Inversion Method

**Theorem:**

Let \( F(x) \) be a CDF of the given distribution. If \( U \) is a \( U(0, 1) \) random number, then

\[ X = F^{-1}(U) \]

is a random variate with CDF \( F \).
**Inversion Method**

**Required:** (Inverse of) CDF $F$ of Distribution.
**Inversion Method**

**Required:** (Inverse of) CDF $F$ of Distribution.

$$u \sim U(0, 1)$$
Inversion Method

**Required:** (Inverse of) CDF $F$ of Distribution.

\[
\begin{align*}
u & \sim U(0, 1) \\
\rightarrow \quad X & = F^{-1}(u)
\end{align*}
\]
Inversion Method // Example

Exponential distribution:

\[ F(x) = 1 - \exp(-x) \]

\[ F^{-1}(u) = -\log(1 - u) \]
Inversion Method // Algorithm

**Required:** Inverse of CDF

- Generate $U \sim U(0, 1)$.
- Compute $X = F^{-1}(U)$.
- Return $X$.  

Hörmann – 2005/10/05 – Non-Uniform Random Variate Generation – p.8/70
Inversion Method // Properties

- The most general method for generating non-uniform random variates.
- Works for all distributions provided that the CDF is given.
Inversion Method // Advantages

- Get one random variate $X$ for each uniform $U$.
- Preserves the structural properties of the underlying uniform PRNG.

Consequently . . .

- Can be used for variance reduction techniques.
- Sampling from truncated distributions.
- Quality of generated random numbers depends only on the underlying uniform PRNG, not on distribution.

Inversion Method // Disadvantages

- CDF and its inverse often not given in closed form (or unknown).

- Methods based on numerical inversion are slow and can only be speeded up by the usage of (large) tables.

- Numerical methods are not exact, i.e., they produce random numbers which are only approximately distributed as the given distribution.
Rejection Method // Idea

\begin{align*}
1/2
\end{align*}
Rejection Method // Idea

\[ \frac{1}{2} \]
Rejection Method // Idea

The diagram illustrates the rejection method for generating non-uniform random variates. The shaded area represents the target distribution, and the dots indicate the points that are accepted. The method involves generating random points within a bounding rectangle and accepting those that fall within the shaded area. This process efficiently samples from the desired distribution.
Rejection Method // Idea
Rejection Method // Idea
Rejection Method

Required:
- density $f(x)$
Rejection Method

Required:
- density $f(x)$
- hat $h(x) \geq f(x)$
Rejection Method

Required:
- density $f(x)$
- hat $h(x) \geq f(x)$

Optional:
- squeeze $s(x) \leq f(x)$
Rejection Method // Algorithm

- Generate $X \sim \text{hat}$.  

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

- If $U \cdot h(X) \leq s(X)$,  
  Return $X$.  

- If $U \cdot h(X) \leq f(X)$,  
  Return $X$.  

- Else try again.
Rejection Method // Algorithm

Generate $X \sim \text{hat}$.

Generate $U \sim \mathcal{U}(0, 1)$.

If $U \cdot h(X) \leq s(X)$, Return $X$.

If $U \cdot h(X) \leq f(X)$, Return $X$.

Else try again.
Rejection Method // Algorithm

Generate $X \sim \text{hat}$.

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

If $U \cdot h(X) \leq s(X)$,
Return $X$.

Else if $U \cdot h(X) \leq f(X)$,
Return $X$.

Else try again.
Rejection Method // Algorithm

Generate $X \sim \text{hat}$.  
Generate $U \sim U(0,1)$.  
If $U \cdot h(X) \leq s(X)$,  
Return $X$.  
Else try again.

If $U \cdot h(X) \leq f(X)$,  
Return $X$.  
Else try again.
Rejection Method // Algorithm

Generate $X \sim \text{hat}$.  
Generate $U \sim U(0, 1)$.  
If $U \cdot h(X) \leq s(X)$,  
Return $X$.  
If $U \cdot h(X) \leq f(X)$,  
Return $X$.  
Else try again.
Rejection Constant

$$\alpha = \frac{\int h(x) \, dx}{\int f(x) \, dx} = \frac{\text{area below hat}}{\text{area below density}}$$

is called the rejection constant and gives the expected number of iterations to get one random variate.

In practice more useful:

$$\rho = \frac{\int h(x) \, dx}{\int s(x) \, dx} = \frac{\text{area below hat}}{\text{area below squeeze}}$$

Notice that $f$ need not integrate to 1 and $\int f$ need not be known.
Rejection Method // Design Objectives

- Generation from hat distribution must be fast and easy (via inversion).
- Computation of squeeze must be cheap (compared to evaluation of density).
- Rejection constant $\alpha$ (or ratio $\rho$) should be small.
Composition Method

Write density $f(x)$ as discrete mixture

$$f(x) = \sum p_i f_i(x)$$

$p_i$ ... probability vector

$f_i(x)$ ... densities

**Applications:**

- Split region below density. *(Patchwork methods)*
- Partition domain of density and make hat function for each subinterval.
Algorithm

- Generate random variate $J \in \mathbb{Z}$ with prob. vector $(p_i)$.
- Generate random variate $X$ with density $f_J$.
- Return $X$.

\[
(f(x) = \sum p_i f_i(x))
\]
Goal for design:

- fast generator and/or
- simple code and/or
- little storage requirements
- sometimes: special applications

Vast literature on generation methods for standard distributions; eg. Book: *Devroye (1986).*
Special Algorithms // Design

Algorithms are especially tailored for one distribution.

Thus one of the following has to be done:

- Find a good approximation for inverse CDF.
- Find a (tight) upper bound for density function, that is integrable and the inverse of the anti-derivative is easy and fast to compute. (Hat)
- Find a (tight) lower bound for the density that is cheaper to evaluate than the density. (Squeeze)
- Find a clever decomposition of the region below the density. (Patchwork)
Special Algorithm  //  Some Drawbacks

- Expert required for making generator for new distribution. (Testing correctness!!!)
- Many methods are available, only a few are used. (Not always the best!)
  eg. Normal distribution:
    - Box and Muller (1958)
    - Kinderman and Ramage (1976)
    - sum of 12 uniforms (*sic!*)
- Only speed and storage requirements of such generators are described.
- Structural properties are hardly investigated. (In opposition to uniform random number generators.)
Universal Methods

Universal algorithms have been developed for non-standard distributions.

Also called black-box or automatic methods.

Idea:
One algorithm works for a large class of distributions.

Today these algorithms have properties that make them also attractive for generating from standard distributions. Even for sampling from Gaussian distributions.
Universal Methods // Why?

Reasons for development:

- In many simulation situations the application of standard distributions is not adequate.
- Development of generator for special distribution too “expensive”.

Today:

- Generators with known structural properties for a large class of distributions.
- Can be used by “non-experts” for special problems.
Universal Methods // Design Goals

Need a procedure that adjusts parameters of universal algorithm to the given distribution.

Obvious costs:
- Higher storage requirements, /or
- Expensive setup, /or
- Slower marginal generation times.

Modern algorithms give the freedom of choice.

Control over structural properties of generated random variates.
Universal Methods

The most efficient black-box algorithms are based on composition combined with the rejection method. Hat function and squeezes are computed automatically.

(TDR)

*Transformed density rejection* by Gilks and Wild (1992), and Hörmann (1995).

(TABL)


(HINV)

A numerical *inversio method* using Hermite interpolation. Not exact (error can be made small), set-up very slow, large tables, CDF required.
How can we make the choice and the construction of the hat-function automatic?
Transformed Density Rejection (TDR)

- Find a monotone differentiable transformation $T$, such that the transformed density $T(f(x))$ is concave.

  Such densities are called $T$-concave densities.
  (Eg. $T(x) = \log(x)$ ... log-concave densities.)

- Use tangents and secants to construct hat and squeeze for the transformed density, resp. The hat is then the minimum of all these tangents.

- By transforming back into the original scale using $T^{-1}$ we get hat $h(x)$ and squeeze $s(x)$ for the density.

(Devroye 1986; Gilks and Wild 1992; Hörmann 1995; Evans and Swartz 1998)
Transformed Density Rejection

Gaussian distribution, \( T(x) = \log(x) \)

Transformed density  Density with hat and squeeze
Required: T-concave density

[Setup]

- Choose $N$ appropriate construction points for tangents.
- Construct hat $h(x)$ and squeeze $s(x)$.
- Compute intervals $I_1, \ldots, I_N$. (Boundaries are the intersection points of the tangents.)
- Compute areas $A_j$ below hat for each $I_j$. 
Generators

Generate $J$ with probability vector $(\lambda_1, \ldots, \lambda_N)$. (Can be done in constant time (independent of $N$) by means of e.g. Indexed Search.)

Generate $X$ with density prop. to $h|_J$ (by inversion).

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

If $U \cdot h(X) \leq s(X)$, Return $X$.

Else if $U \cdot h(X) \leq f(X)$, Return $X$.

Else try again.
Family $T_c$ of transformations (H"ormann 1995):

$$T_0(x) = \log(x) \quad \text{for } c = 0$$
$$T_c(x) = -x^c \quad \text{for } c \in (-1, 0)$$

$c \leq -1$ or $c > 0$ possible for density with bounded domain.

If $f$ is $T_c$-concave, then $f$ is $T_{c'}$-concave for every $c' \leq c$.

For computational reasons the choice of $c = -1/2$ (if possible) is suggested.
### $T_{-1/2}$-concave Densities

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Density Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>$e^{-x^2/2}$</td>
<td></td>
</tr>
<tr>
<td>Exponential</td>
<td>$\lambda e^{-\lambda x}$</td>
<td>$\lambda &gt; 0$</td>
</tr>
<tr>
<td>Gamma</td>
<td>$x^{a-1} e^{-b x}$</td>
<td>$a \geq 1$, $b &gt; 0$</td>
</tr>
<tr>
<td>Beta</td>
<td>$x^{a-1} (1 - x)^{b-1}$</td>
<td>$a, b \geq 1$</td>
</tr>
<tr>
<td>Weibull</td>
<td>$x^{a-1} \exp(-x^a)$</td>
<td>$a \geq 1$</td>
</tr>
<tr>
<td>Perks</td>
<td>$1/(e^x + e^{-x} + a)$</td>
<td>$a \geq -2$</td>
</tr>
<tr>
<td>G. I. G.</td>
<td>$x^{a-1} \exp(-bx - b^*/x)$</td>
<td>$a \geq 1$, $b, b^* &gt; 0$</td>
</tr>
<tr>
<td>Student's t</td>
<td>$(1 + (x^2/a))^{-(a+1)/2}$</td>
<td>$a \geq 1$</td>
</tr>
<tr>
<td>Pearson VI</td>
<td>$x^{a-1}/(1 + x)^{a+b}$</td>
<td>$a, b \geq 1$</td>
</tr>
<tr>
<td>Cauchy</td>
<td>$1/(1 + x^2)$</td>
<td></td>
</tr>
<tr>
<td>Planck</td>
<td>$x^a/(e^x - 1)$</td>
<td>$a \geq 1$</td>
</tr>
<tr>
<td>Burr</td>
<td>$x^{a-1}/(1 + x^a)^b$</td>
<td>$a \geq 1$, $b \geq 2$</td>
</tr>
<tr>
<td>Snedecor’s F</td>
<td>$x^{m/2-1}/(1 + \frac{m}{n}x)^{(m+n)/2}$</td>
<td>$m, n \geq 2$</td>
</tr>
</tbody>
</table>
Performance

The performance is controlled by a single parameter, the ratio

\[ \rho = \frac{\int h(x) \, dx}{\int s(x) \, dx} = \frac{\text{area below hat}}{\text{area below squeeze}} \]

- \( \rho - 1 \approx \) expected number of evaluation of the density.
- \( \rho \) is an upper bound for the rejection constant.
- \( \rho - 1 \) is a “measure” of the deviation from inversion.

By construction, \( \rho \) is easy to compute.

Notice that \( f \) need not integrate to 1 and \( \int f \) need not be known.
Choice of $\rho$ provides some flexibility.

$\rho$ large (small $N$ is enough):
- fast setup, but
- slow marginal generation time.

$\rho \approx 1$ (large $N$ is necessary):
- rather expensive setup, but
- very fast marginal generation time.
Properties

If $\rho \approx 1$ then the algorithm is close to inversion and shares its desired properties.

- Can be used for variance reduction techniques.
- Sampling from truncated distributions.
- Quality of generated random numbers depends only on the underlying uniform random number generator, not on distribution.

Additionally

- Exact method.
- The marginal generation time is fast and does not depend on the distribution.
Scatterplots

Scatterplots of tuples \((u_{n+1}, u_n)\) for “Baby”-generator

\[u_{n+1} = 869 \cdot u_n + 1 \mod 1024\]
Scatterplots of tuples $(F(X_{n+1}), F(X_n))$ for “Baby”-generator

$$u_{n+1} = 869 \, u_n + 1 \mod 1024$$

Gaussian

$\rho = 1.86$
Scatterplots

Scatterplots of tuples $(F(X_{n+1}), F(X_n))$ for “Baby”-generator

$$u_{n+1} = 869 \, u_n + 1 \mod 1024$$

Gaussian

$$\rho = 1.431$$
Scatterplots

Scatterplots of tuples \((F(X_{n+1}), F(X_n))\) for “Baby”-generator

\[ u_{n+1} = 869 \, u_n + 1 \mod 1024 \]

Gaussian
\[ \rho = 1.230 \]
Scatterplots

Scatterplots of tuples \((F(X_{n+1}), F(X_n))\) for “Baby”-generator

\[ u_{n+1} = 869 \, u_n + 1 \mod 1024 \]

Gaussian

\[ \rho = 1.022 \]
Scatterplots

Scatterplots of tuples \((F(X_{n+1}), F(X_n))\) for “Baby”-generator

\[ u_{n+1} = 869 \, u_n + 1 \mod 1024 \]

Gaussian

\[ \rho = 1.003 \]
Scatterplots of tuples \((F(X_{n+1}), F(X_n))\) for “Baby”-generator

\[ u_{n+1} = 869 \, u_n + 1 \mod 1024 \]
Scatterplots of tuples \((F(X_{n+1}), F(X_n))\) for “Baby”-generator

\[ u_{n+1} = 869 \, u_n + 1 \mod 1024 \]
TDR // Constructions Points

TDR works well when the ratio $\rho$ is close to one.

**Problem:**
We have to find construction points, such that $\rho$ is small.

**Constraint:**
Do not use much more than necessary.
Adaptive rejection sampling *(Gilks and Wild 1992)*

1. Start with two arbitrary construction points.
2. Sample random points $x$.
3. Whenever the density has to be evaluated at $x$, add $x$ as new construction points until the aimed ratio $\rho$ has been reached.
Adaptive rejection sampling (Gilks and Wild 1992)

1. Start with two arbitrary construction points.
2. Sample random points \( \chi \).
3. Whenever the density has to be evaluated at \( \chi \), add \( \chi \) as new construction points until the aimed ratio \( \rho \) has been reached.
Adaptive rejection sampling (Gilks and Wild 1992)

1. Start with two arbitrary construction points.
2. Sample random points $\chi$.
3. Whenever the density has to be evaluated at $\chi$, add $\chi$ as new construction points until the aimed ratio $\rho$ has been reached.
Adaptive rejection sampling (Gilks and Wild 1992)

1. Start with two arbitrary construction points.
2. Sample random points $x$.
3. Whenever the density has to be evaluated at $x$, add $x$ as new construction points until the aimed ratio $\rho$ has been reached.
Adaptive rejection sampling (Gilks and Wild 1992)

1. Start with two arbitrary construction points.
2. Sample random points \( x \).
3. Whenever the density has to be evaluated at \( x \), add \( x \) as new construction points until the aimed ratio \( \rho \) has been reached.
Adaptive rejection sampling

It was developed for Gibbs sampling applications.

There just one random variate is needed from one distribution (the varying parameter case).

For the speed of the algorithm it is therefore important to have good starting values for the points of contact.

In Gibbs sampling this is the case, if the random vector has approximately uncorrelated components.

Our algorithms were mainly developed for the fixed parameter case: to sample many variates from a fixed distribution.
Derandomized adaptive rejection sampling

1. Start with two arbitrary construction points.
2. Add expected points in ARS:
   - Look for all intervals where $\int (h(x) - s(x)) \, dx$ is too large.
   - Insert expected point $\int x (h(x) - s(x)) \, dx$.
   - Loop until the aimed ratio $\rho$ has been reached.

(more efficient, but less robust than ARS)
Optimal construction points

3 construction points: Hörmann (1995)

many: Derflinger (to appear)

For all of the introduced selection methods:

$$\rho = 1 + O(N^{-2}),$$

($N$ number of design points).
The basic method has been modified. (see book: WH, Leydold, Derflinger 2004)

- Squeezes proportional to hat ($\rightarrow$ faster, simpler).
- Immediate acceptance ($\rightarrow$ very fast, fewer URN).
- Non-T-concave distributions (Evans and Swartz 1998) ($\rightarrow$ almost all continuous distributions, but inflection points of transformed density must be given).
Universal Algorithms

Universal algorithms have advantages which make their usage attractive even for standard distributions.

- Only one code for many different distributions.
- Performance controlled by a simple parameter $\rho$.
- Sampling from truncated distributions.
- The marginal generation time does not depend on the density.
- They can be used for variance reduction techniques.
Implementation

These methods can be used

- either to create a generator for a particular distribution,
- or to make a library with a simple API where the user only has to give the data of the desired distribution.
We have implemented these (and other) methods in a library, called UNU.RAN (Universal NonUniform Random variate generators).

Features:

- ANSI C (portable).
- Object oriented programming paradigm.
- Open source.

Can be downloaded from [http://statistik.wu-wien.ac.at/unuran/](http://statistik.wu-wien.ac.at/unuran/)
#include <unuran.h>

main() {
    /* Declare UNURAN generator object. */
    UNUR_GEN *gen;

    /* Create the generator object. */
    /* Standard Gaussian distribution. */
    /* Choose a method: AUTOMATIC. */
    gen = unur_str2gen("normal()");

    /* sample */
    x = unur_sample_cont(gen);

    /* destroy generator object */
    unur_free(gen);

    exit(EXIT_SUCCESS);
}
```c
#include <unuran.h>

main() {
    /* Declare UNURAN generator object. */
    UNUR_GEN *gen;

    /* Create the generator object. */
    /* Standard Gaussian distribution. */
    /* Choose a method: AUTOMATIC. */
    gen = unur_str2gen("normal(); domain=(0,inf)");

    /* sample */
    x = unur_sample_cont(gen);

    /* destroy generator object */
    unur_free(gen);

    exit (EXIT_SUCCESS);
}
```
```c
#include <unuran.h>

main() {
    /* Declare UNURAN generator object. */
    UNUR_GEN *gen;

    /* Create the generator object. */
    /* Standard Gaussian distribution. */
    /* Choose a method: AROU. */
gen = unur_str2gen("normal(); domain=(0,inf) & method=arou");

    /* sample */
x = unur_sample_cont(gen);

    /* destroy generator object */
    unur_free(gen);

    exit(EXIT_SUCCESS);
}
```
#include <unuran.h>

main() {
    /* Declare UNURAN generator object. */
    UNUR_GEN *gen;

    /* Create the generator object. */
    /* Standard Gaussian distribution. */
    /* Choose a method: AROU. */
    gen = unur_str2gen("normal(); domain=(0,inf) & \ 
                       method=arou; max_sqhratio=0.99; usedars=on");

    /* sample */
    x = unur_sample_cont(gen);

    /* destroy generator object */
    unur_free(gen);

    exit(EXIT_SUCCESS);
}
Implementing the above methods results in a rather long and complex computer program.

**Reasons:**

- Hat functions and squeezes have to be constructed in a setup step and improved in possible adaptive steps.
- Round-off errors when computing intersection points.
- Have to test whether the given distribution fulfills the assumption for the chosen method.
- Round-off errors when testing the T-concavity.
- Prevent possible overflow.
However, the actual sampling routines consist only of a few lines of code.

The same methods can be used to produce a single piece of code for a fast generator of a particular distribution given by a user who needs no experience in random number generation.

This generator then produces random variates at a known speed and of predictable quality.

A first experimental version of this concept can be found at http://statistik.wu-wien.ac.at/anuran/.