ARVAG

UNU.RAN

A library for Universal Non-Uniform RANdom variate generators

Concepts and Implementation

Josef Leydold (WU Wien), Wolfgang Hörmann (BU Istanbul)
Lorenzo Moneta (CERN)

Learning Objectives

- Some remarks on uniform random numbers
- General Principles:
  Inversion, Rejection, and Composition
- Special Algorithms vs. Universal Methods
- Random Vectors
- Implementation: UNU.RAN

Non-Uniform Random Variate Generation

Purpose:

Generate a sequence X_i of IID random variates with given distribution.

Non-Uniform Random Variate Generation

Purpose:

Generate a sequence X_i of IID random variates with given distribution.

Independent and **I**dentically **D**istributed

Independence can be dropped for some applications (MCMC).

ARVAG

Non-Uniform Random Variate Generation

Purpose:

Generate a sequence X_i of IID random variates with given distribution.

Solution:

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

$$U_1, U_2, U_3, U_4, \ldots \quad \longrightarrow \quad X_1, X_2, X_3, \ldots$$

This transformation needs not to be one-to-one.

ARVAG

Assumptions

- Have a source of **perfect** (truely) uniform random numbers.

- Real numbers.

ARVAG

Assumptions

- Have a source of **perfect** (truely) uniform random numbers.

  Not possible **in practice**!

- Real numbers.

  Computer uses **Floating Point Arithmetic**!

ARVAG

Uniform Random Numbers Generators

- Pseudo-random numbers.
  Physical devices are slow, simulations not reproducible.
- Have restricted resolution.
  Typically 10^{-9}.
- Always use generators designed by experts.
  No self-made combined generators!
- Test generator, if possible.
  Use a test similar to application.
  E.g. TestU01 (L'Ecuyer), DIEHARD (Marsaglia),
  NIST test-suite.
- PRNG in libraries usually not state-of-the-art.
  Never use standard libraries for serious computations
  (e.g. Visual Basic).

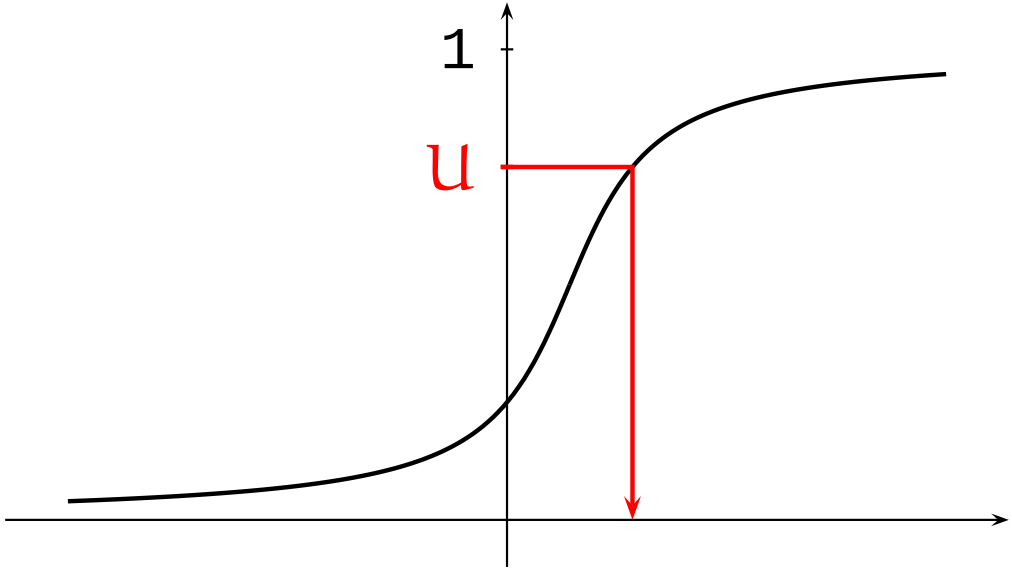
ARVAG

General Principles:

Inversion, Rejection, and Composition

Inversion Method

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

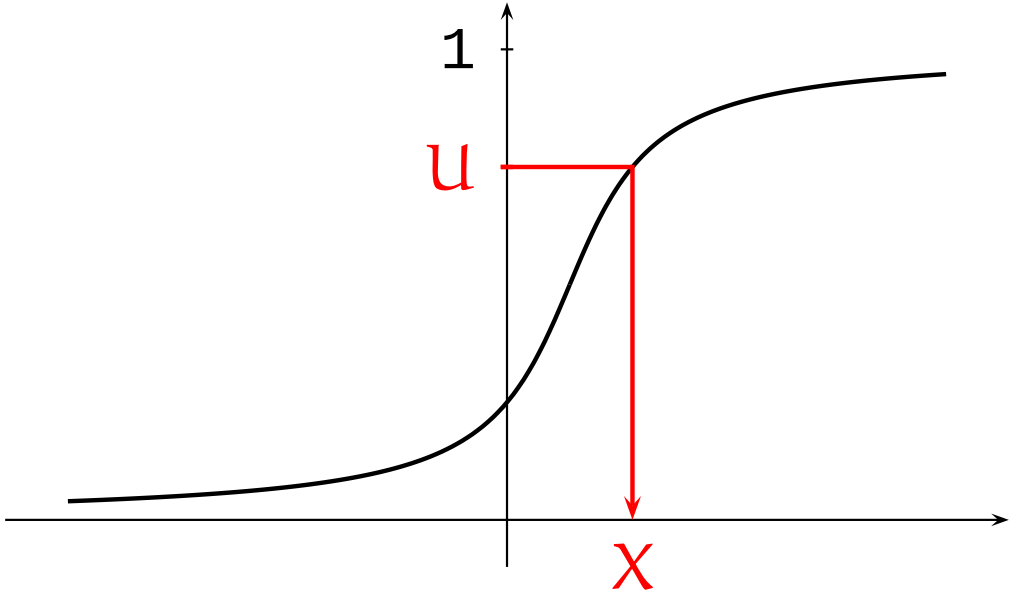


$$U \sim U(0,1) \quad \longrightarrow \quad X = F^{-1}(U)$$

Exponential distribution: $F^{-1}(u) = -\log(1-u)$

Inversion Method

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



$$U \sim U(0,1) \quad \longrightarrow \quad X = F^{-1}(U) \quad \Leftarrow \textbf{Problem (?)}$$

Inversion Method // Properties

- The most **general** method for generating non-uniform random variates.
  Works for all distributions provided that the inverse CDF is given.
- Get **one** random variate X for each uniform U.
- **Preserves** the structural properties of the underlying uniform PRNG.

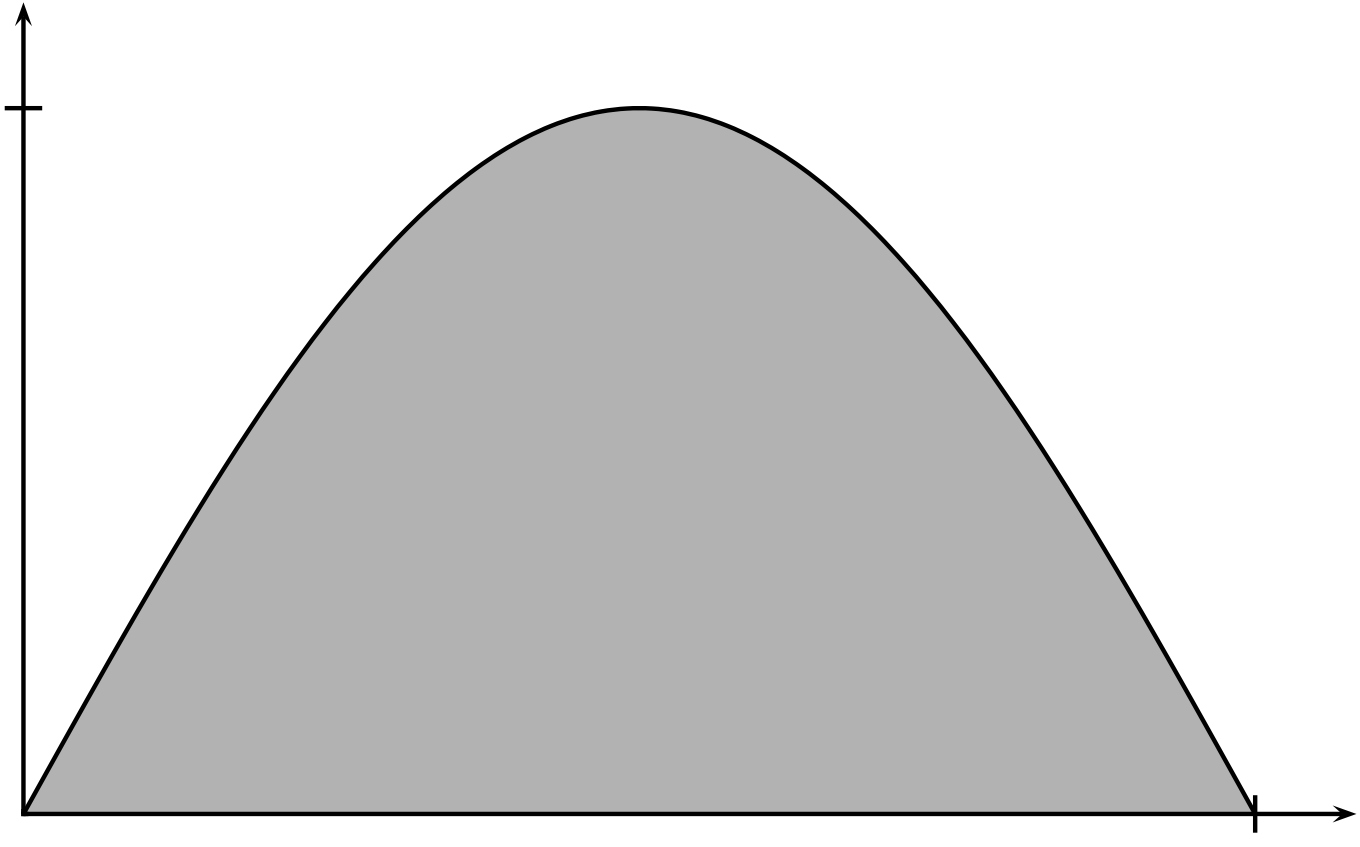
However:

- CDF and its inverse often not given in closed form.
- Need slow and/or **approximate** numerical methods.

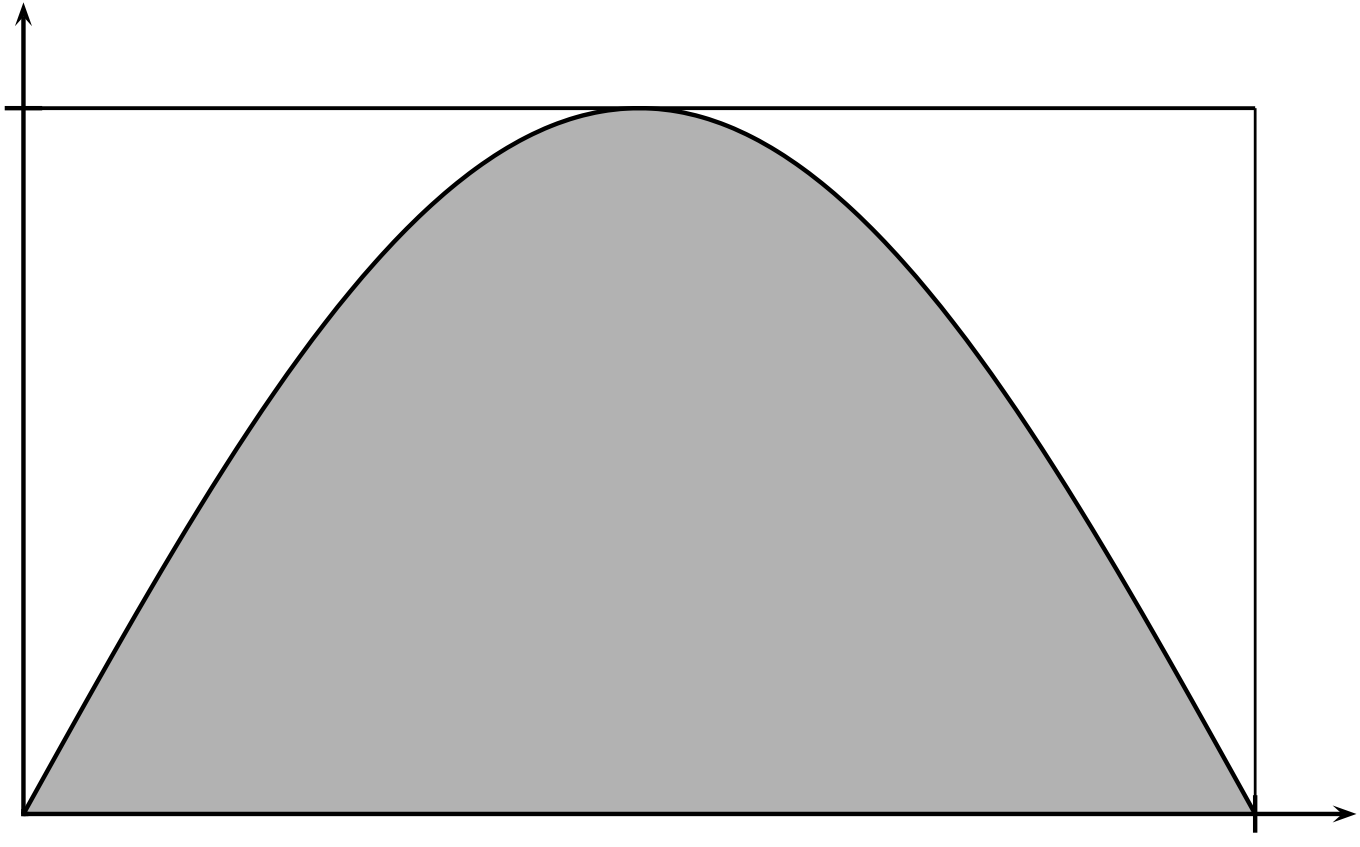
Inversion Method // Application

- Sampling from **truncated** distributions.
- Can be used for **variance reduction** techniques. (common / antithetic variates, stratified sampling, . . .)
- **Quality** of generated random numbers depends only on the underlying uniform PRNG, not on distribution.
- Applicable for **copula** methods and in the framework of **Quasi-Monte Carlo** methods (HUPS).

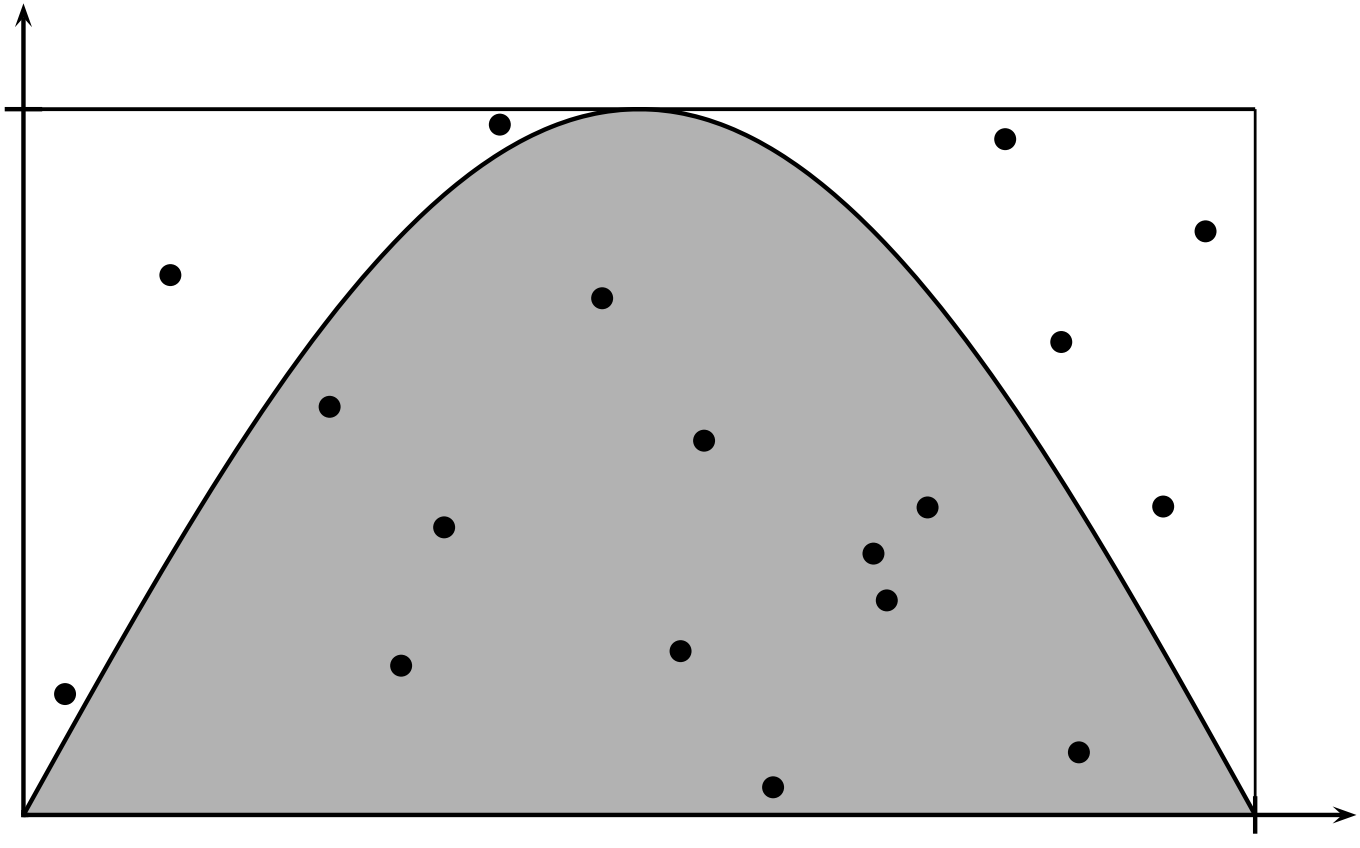
Rejection Method // Idea



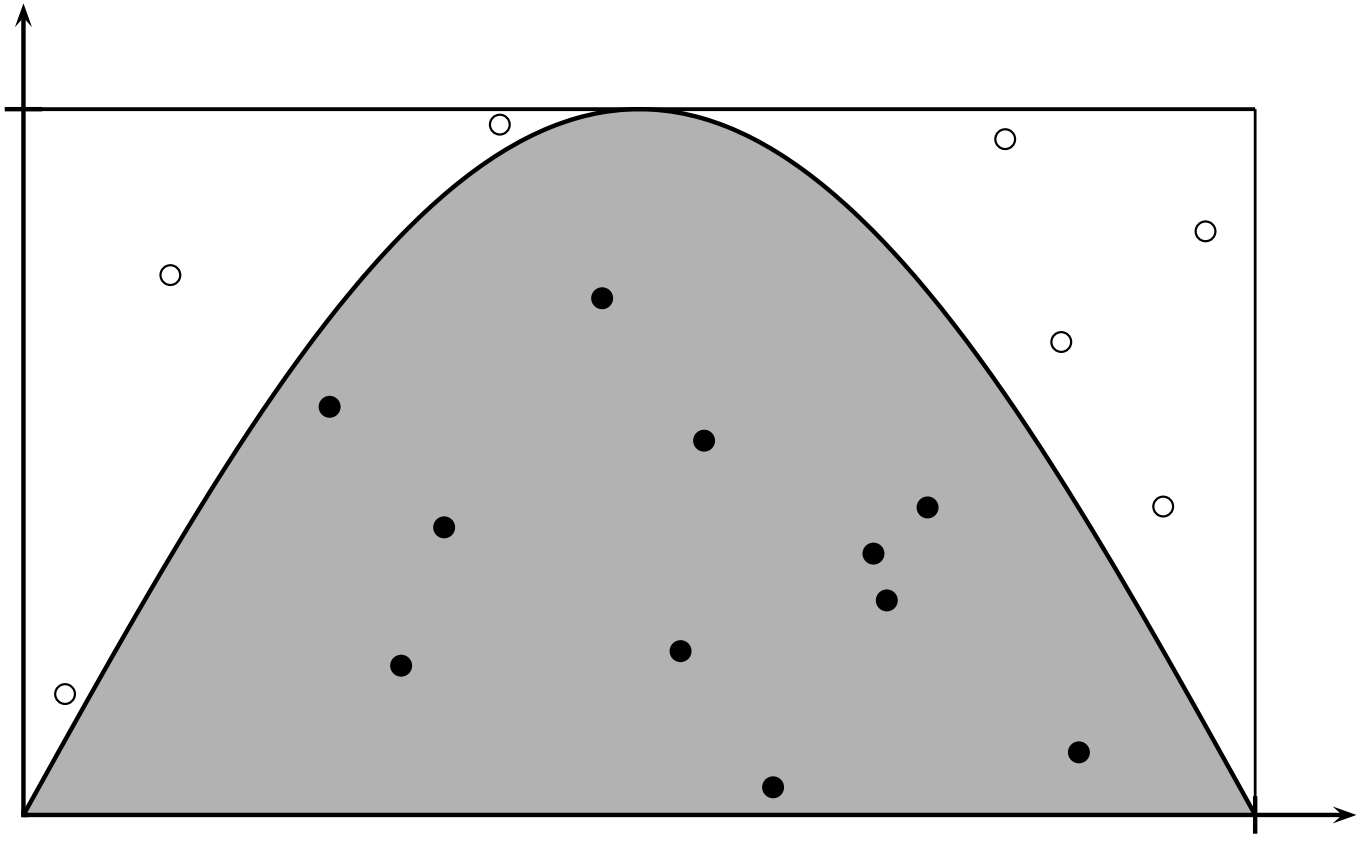
Rejection Method // Idea



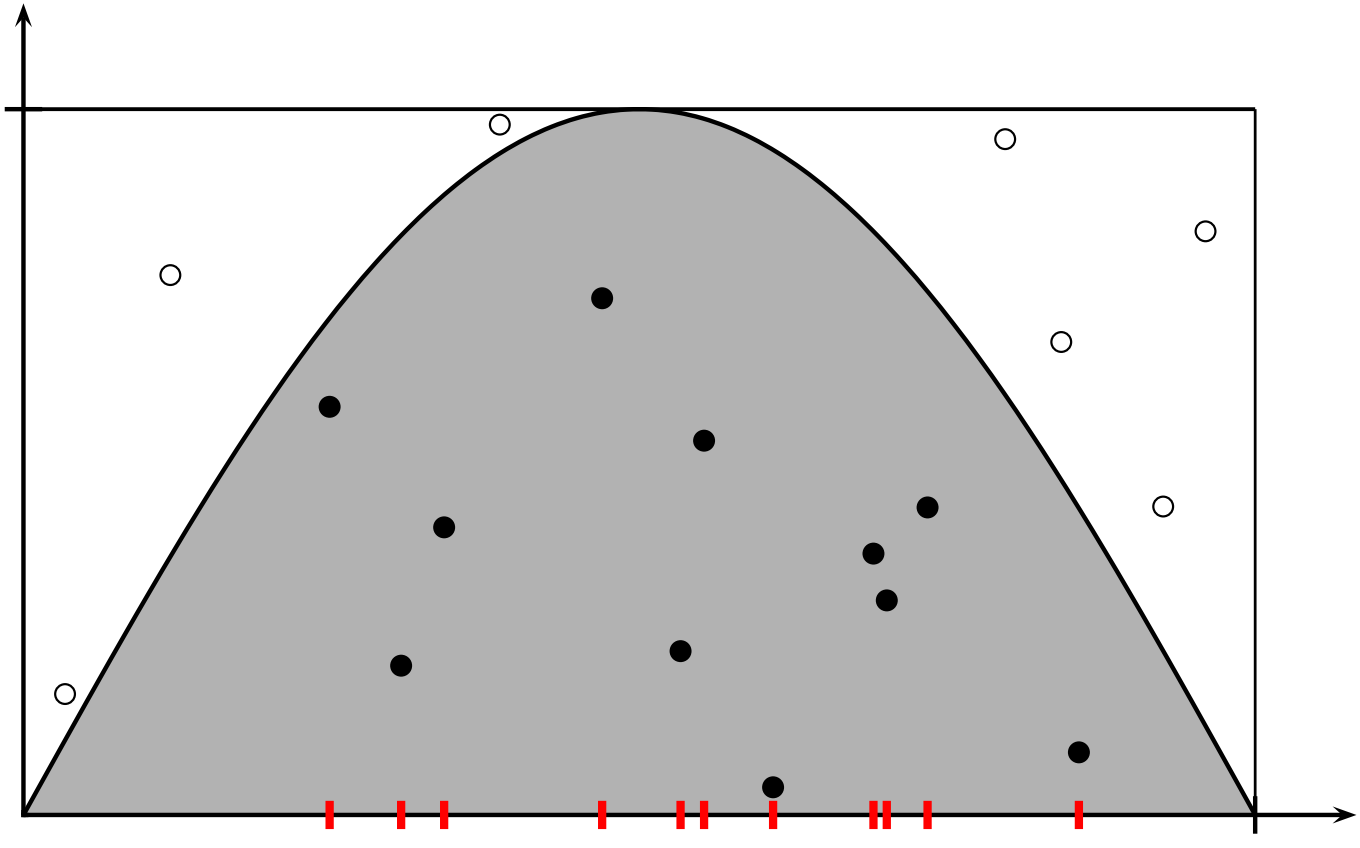
Rejection Method // Idea



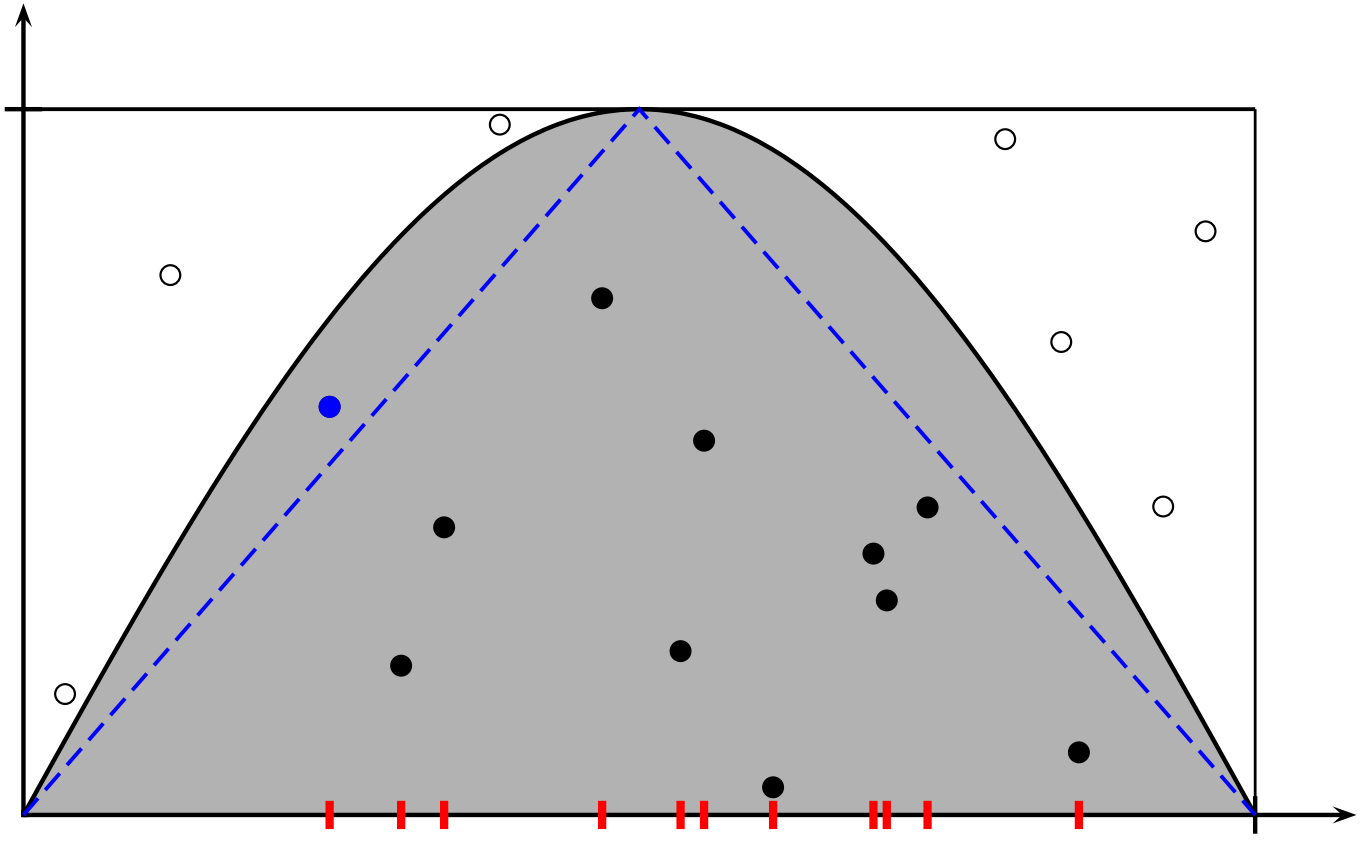
Rejection Method // Idea



Rejection Method // Idea



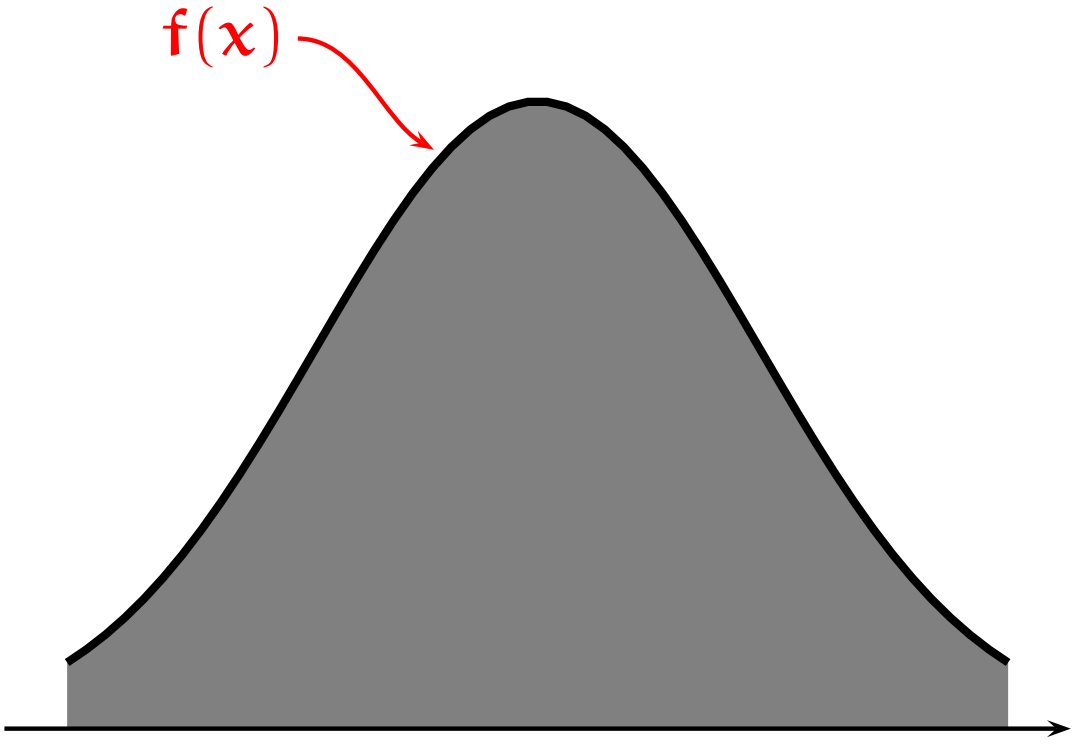
Rejection Method // Idea



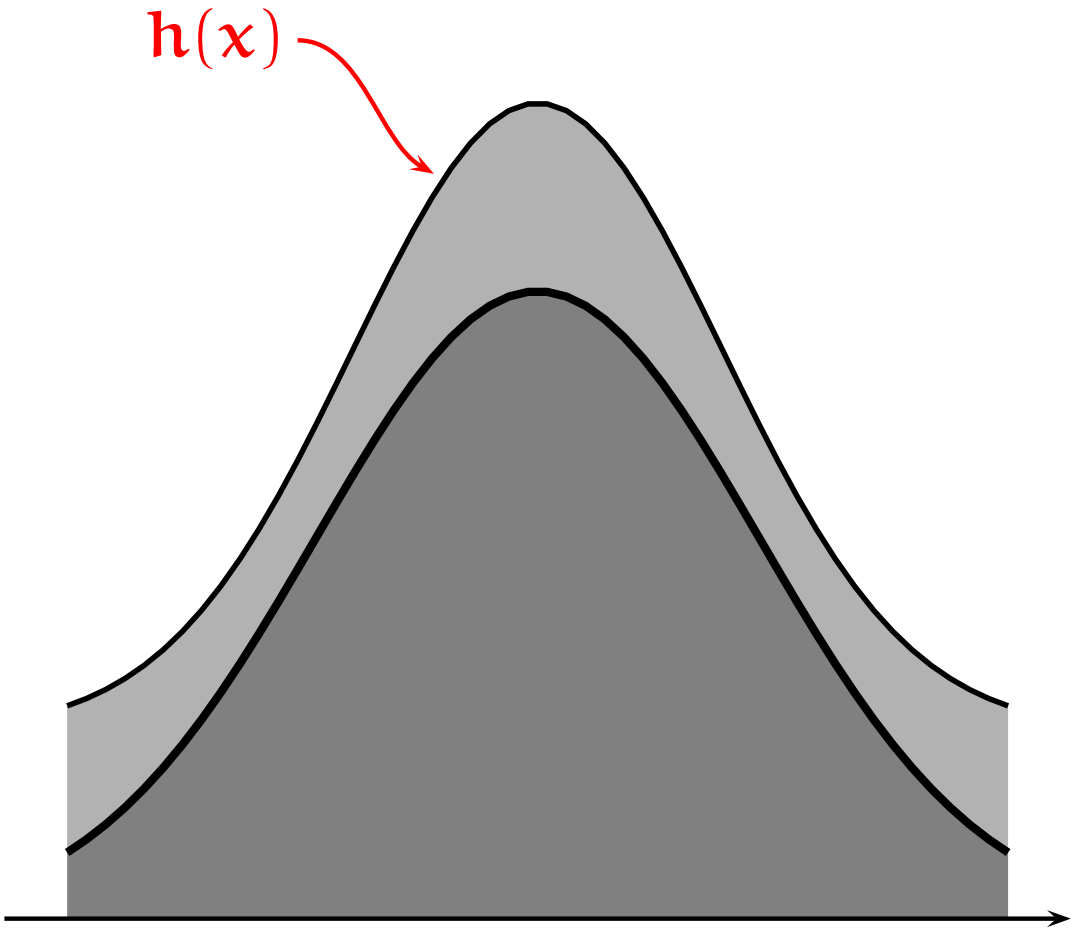
Rejection Method

Required:

- density $f(x)$



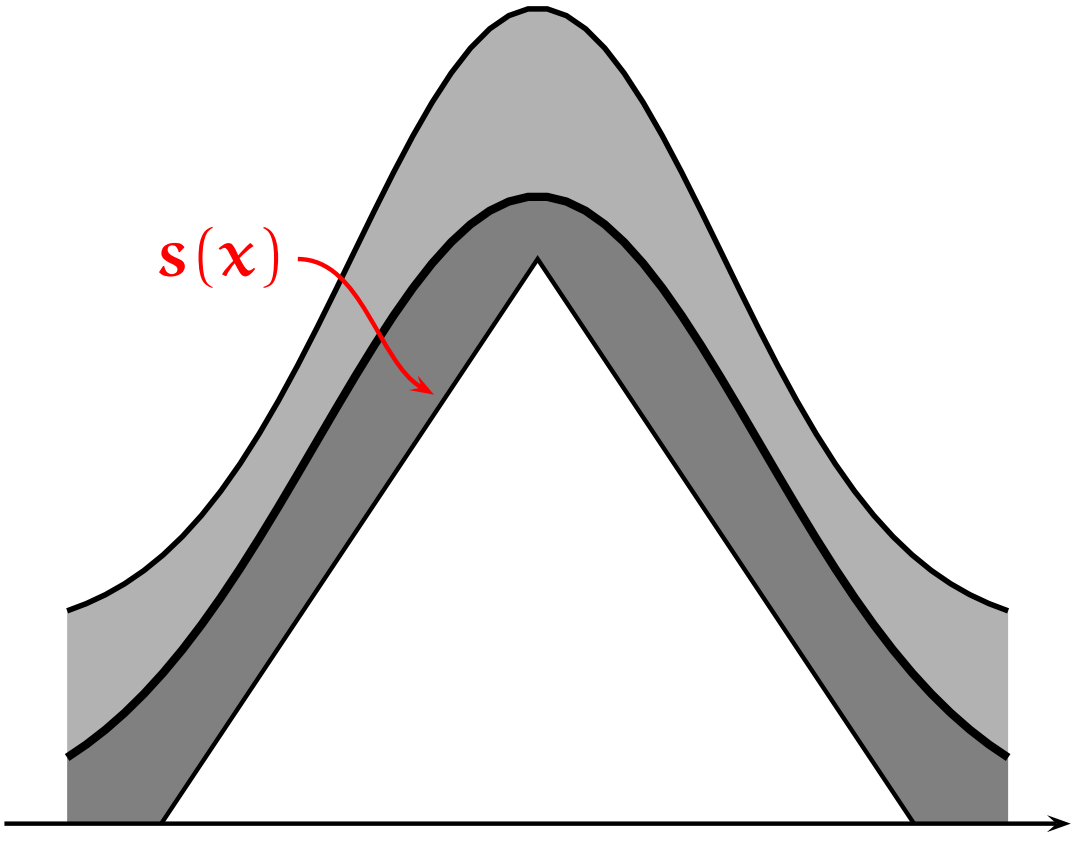
Rejection Method



Required:

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

Rejection Method



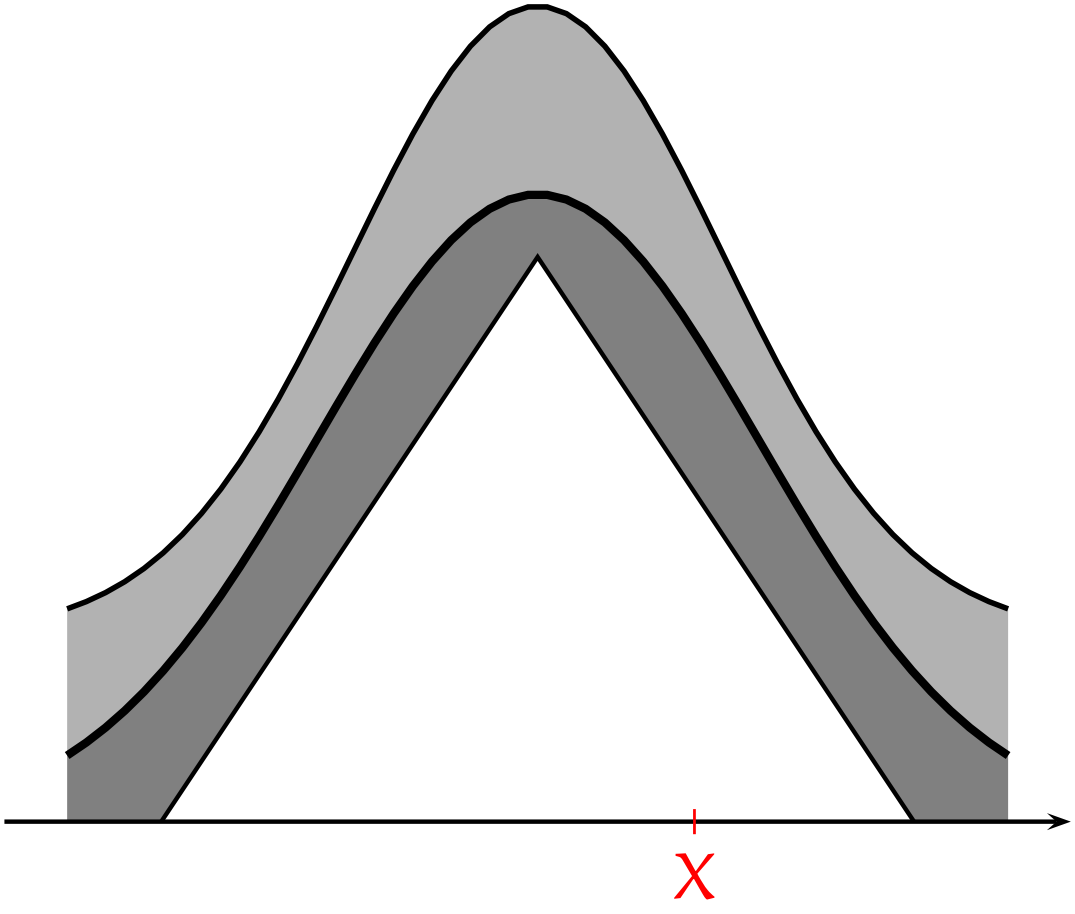
Required:

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

Optional:

- squeeze $s(x) \leq f(x)$

Rejection Method // Algorithm



- Generate $X \sim$ hat.

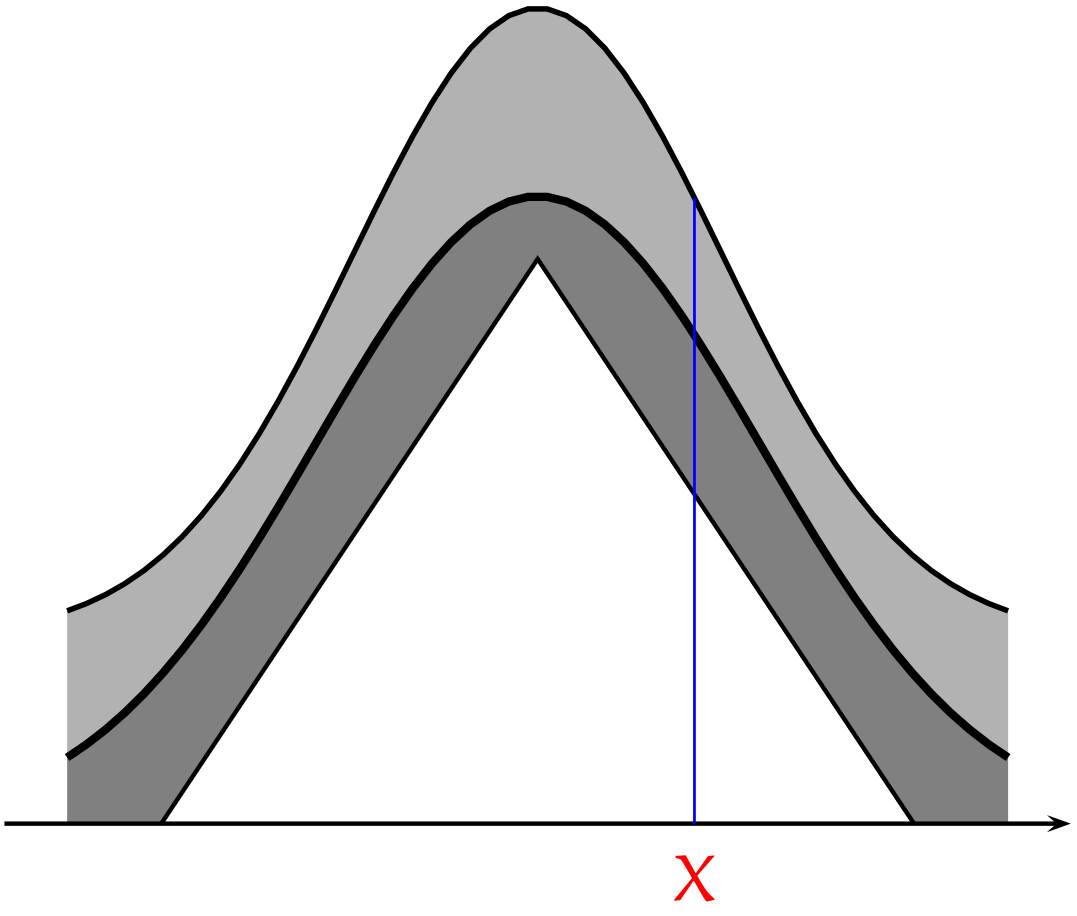
  Generate $U \sim U(0,1)$.

  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 $X \sim$ hat.

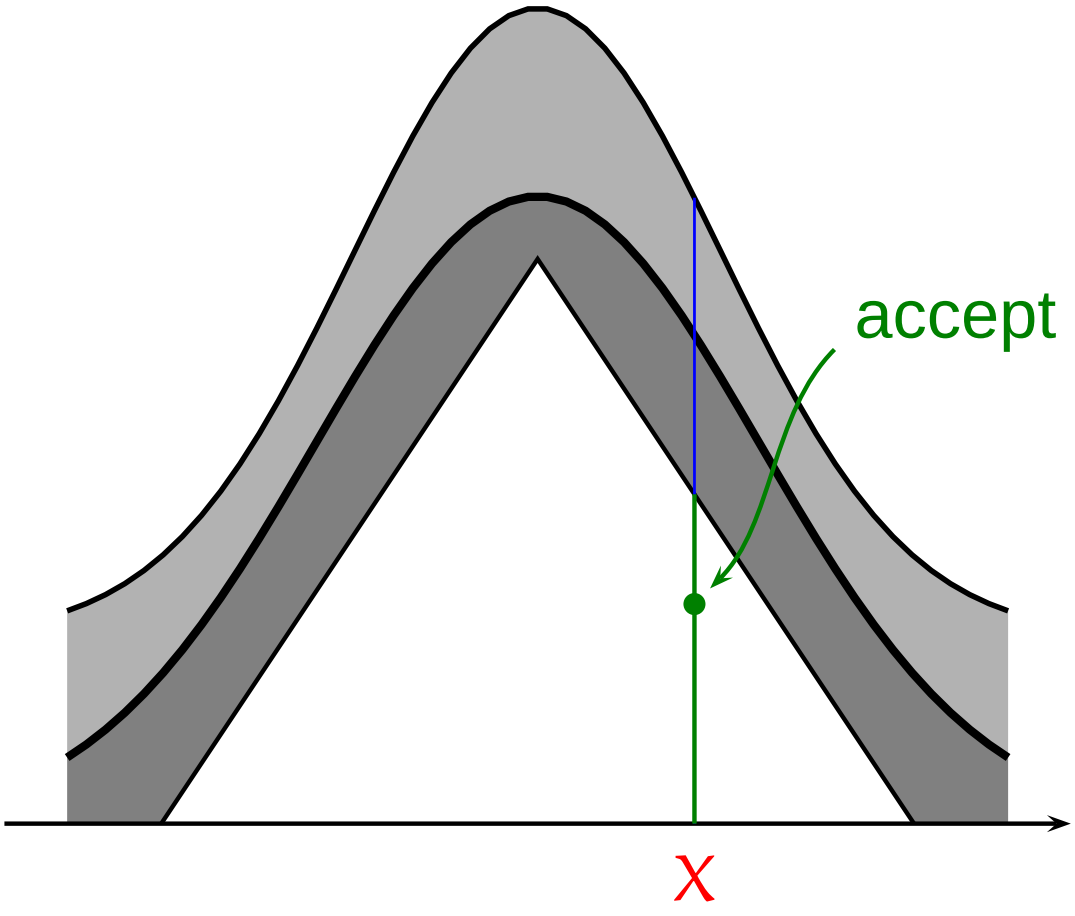
- Generate $U \sim U(0,1)$.

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 $X \sim$ hat.

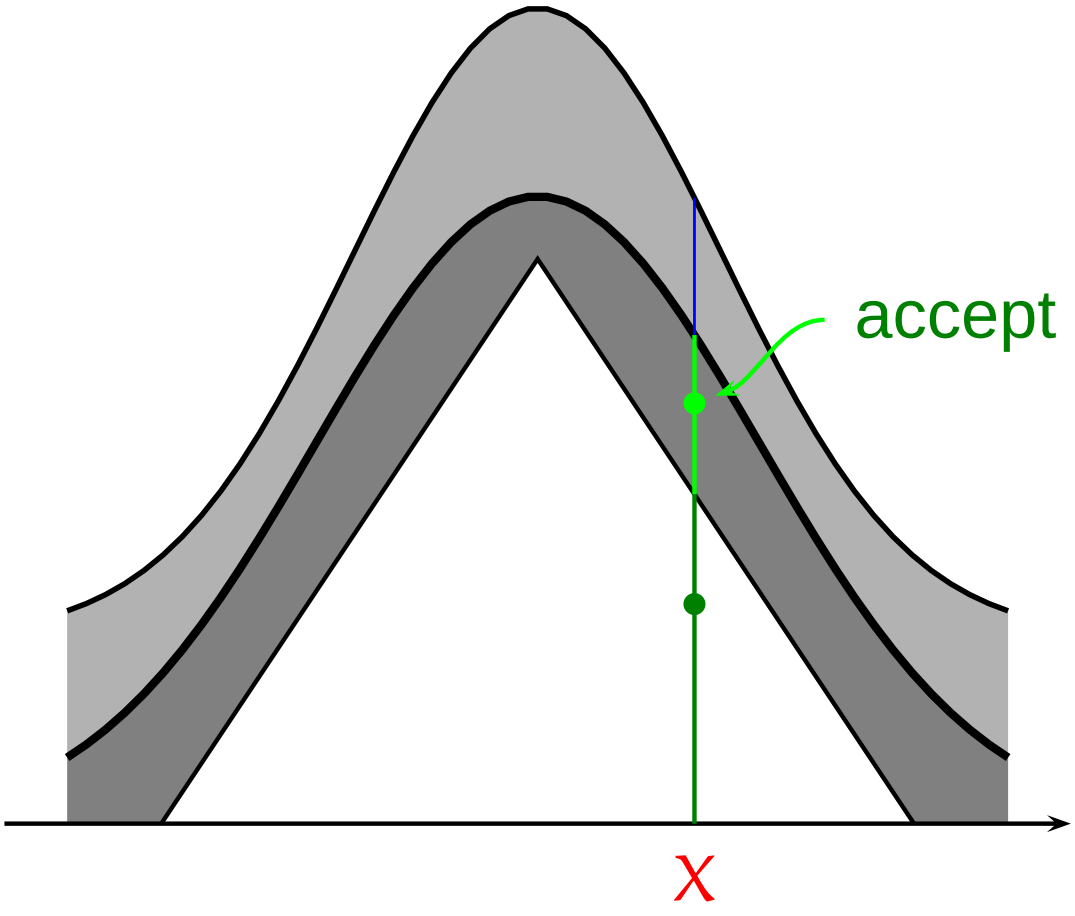
Generate $U \sim U(0,1)$.

- 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 $X \sim$ hat.

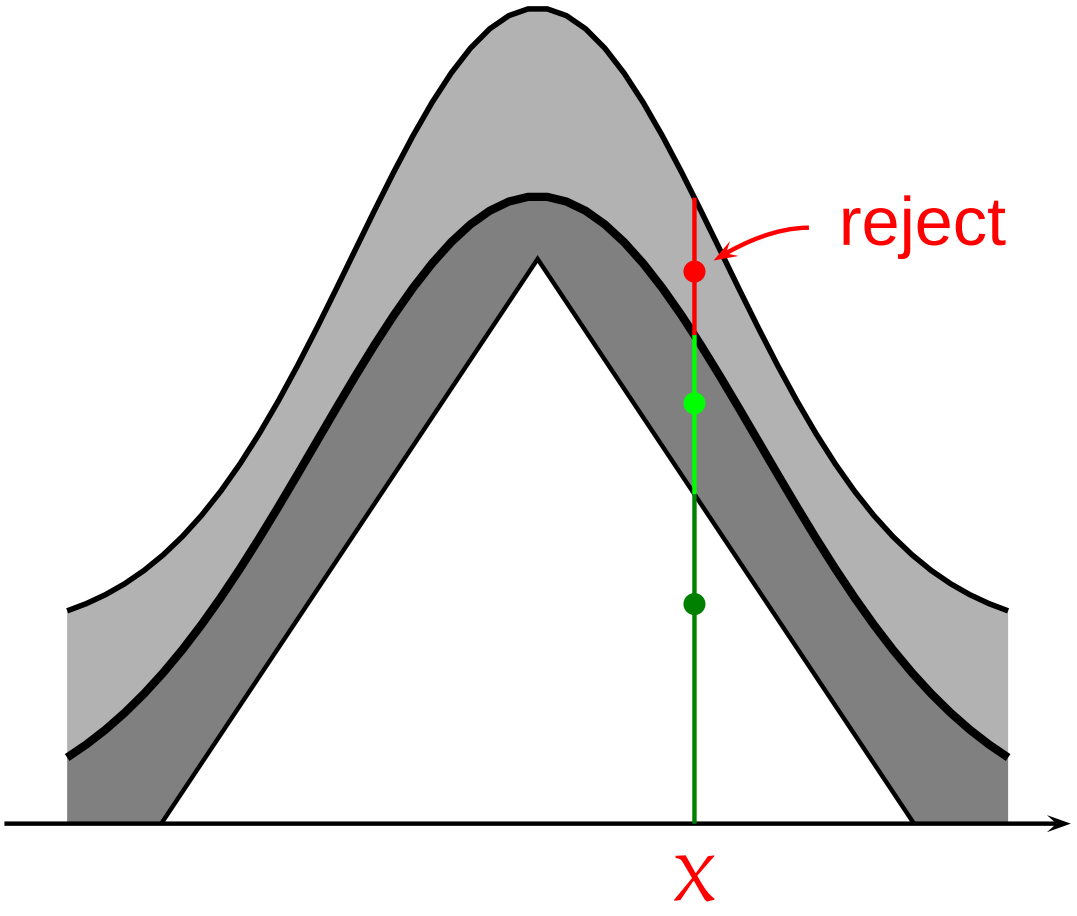
Generate $U \sim U(0,1)$.

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 $X \sim$ 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.

ARVAG

Rejection Constant

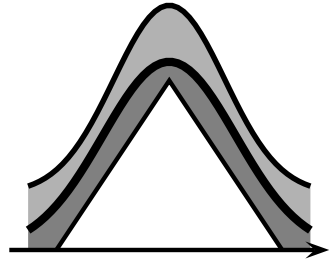
$$\alpha = \frac{\int h(x)\,dx}{\int f(x)\,dx} = \frac{\text{area below hat}}{\text{area below density}}$$

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

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.



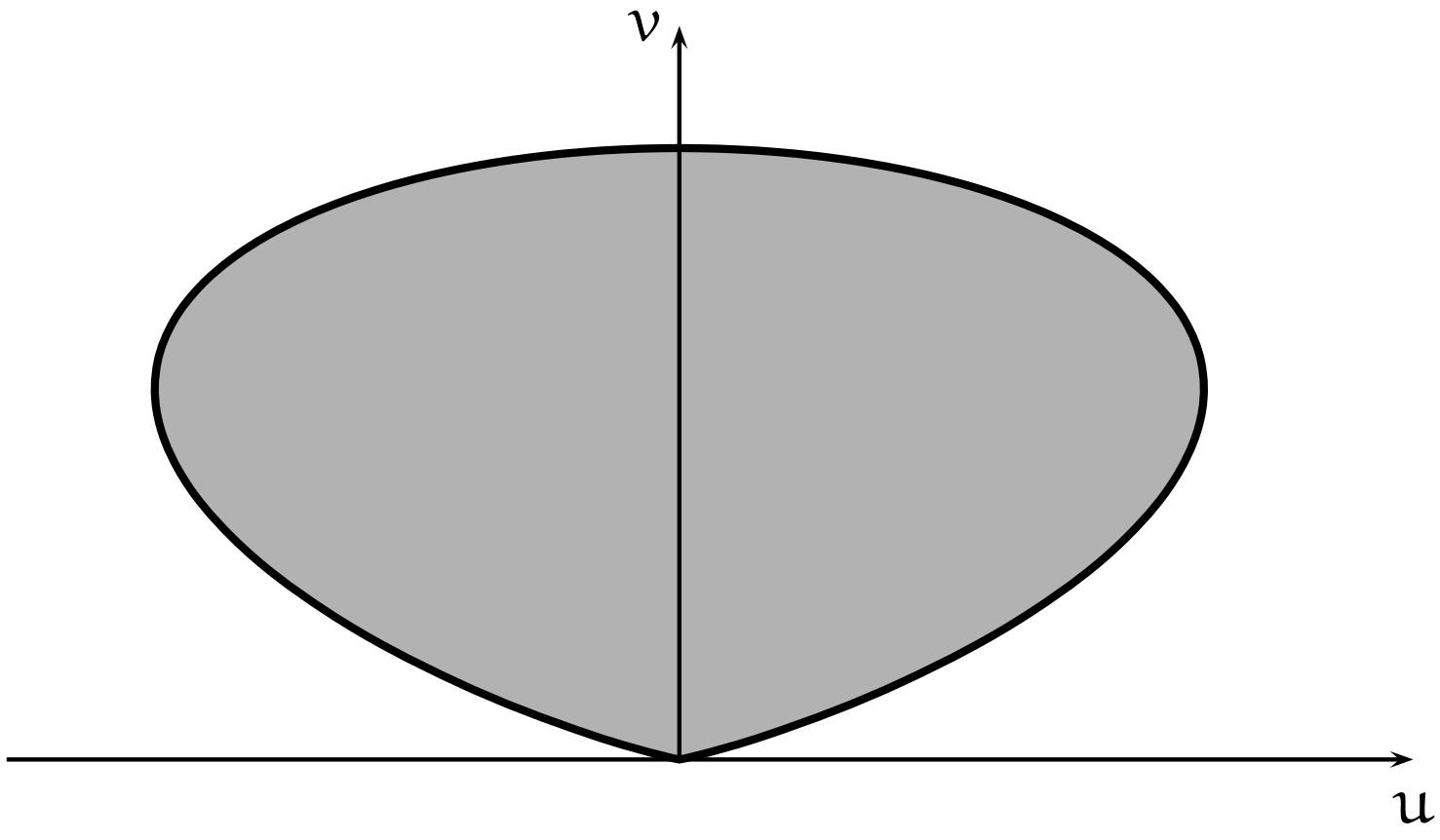
In practice more useful:

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

Notice that f need not integrate to 1 and $\int f$ need not be known.

Ratio-of-Uniforms

Transform acceptance region: $(x, y) \mapsto (u, v) = (x\sqrt{y}, \sqrt{y})$



Normal distribution

Composition Method

Required: Composition $f(x) = \sum p_i f_i(x)$

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

  Generate random variate X with density f_J.

  Return X.

Composition Method

Required: Composition $f(x) = \sum p_i f_i(x)$

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

  Generate random variate X with density f_J.

  Return X.

Can be done in constant time (independent of N) by means of *Indexed Search* (Chen and Asau, 1974).

Composition Method

Required: Composition $f(x) = \sum p_i f_i(x)$

Generate random variate $J \in \mathbb{Z}$ with prob. vector (p_i).

- Generate random variate X with density f_J.

Return X.

Composition Method

Required: Composition $f(x) = \sum p_i f_i(x)$

Generate random variate $J \in \mathbb{Z}$ with prob. vector (p_i).

Generate random variate X with density f_J.

- Return X.

ARVAG

Special Algorithms

vs.

Universal Methods

Algorithms for Particular Distributions

Goal for design:

- **fast** generator and/or
- **simple** code and/or
- little storage requirements
- **Structural** properties are hardly investigated.
  (In opposition to uniform random number generators.)
- Expert required for making generator for new distribution.

Vast literature on generation methods for standard distributions; eg. Book: Devroye (1986),

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 makes them also attractive for generating from standard distributions.
Even for sampling from Gaussian distributions.

E.g., their structural properties are known.

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.

Two Examples

We make a glimpse at two such algorithms:

(**TDR**)

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

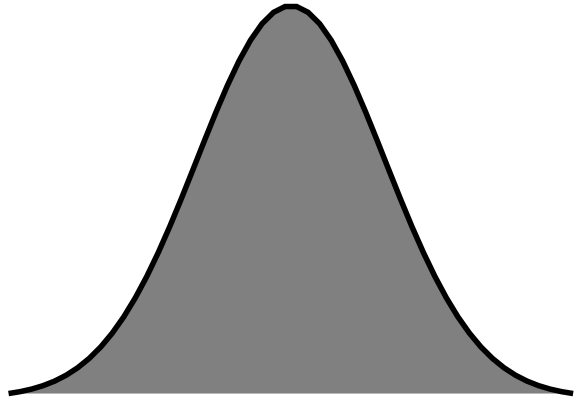
(**HINV**)

A numerical *inversion* method using Hermite interpolation. (Hörmann and Leydold, 2003)

All respective constants for hat, squeeze or interpolating polynome are computed automatically.

Transformed Density Rejection

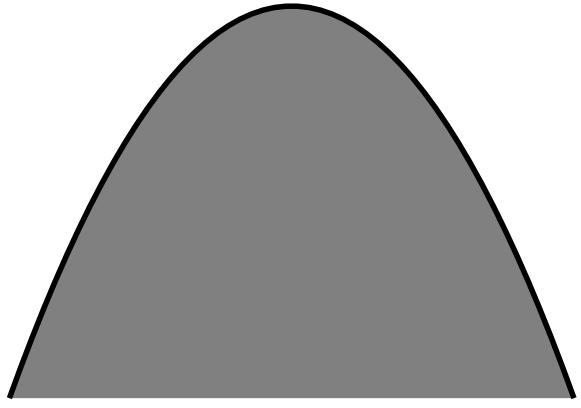
Gaussian distribution, $T(x) = \log(x)$



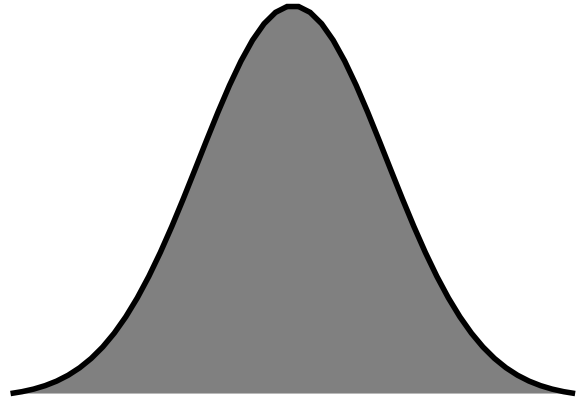
Density with hat and squeeze

Transformed Density Rejection

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



Transformed density

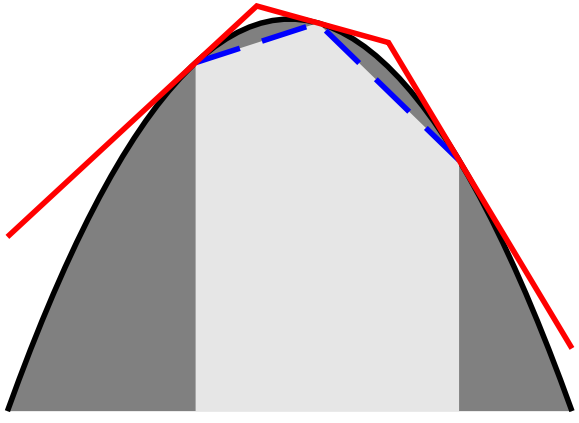


Density with hat and squeeze

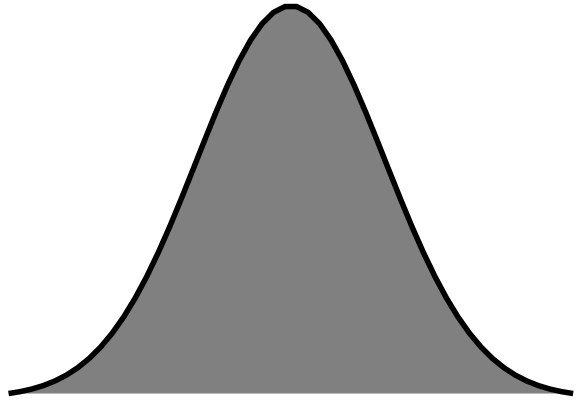
Find a monotone differentiable transformation T, such that the transformed density $T(f(x))$ is concave.

Transformed Density Rejection

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



Transformed density

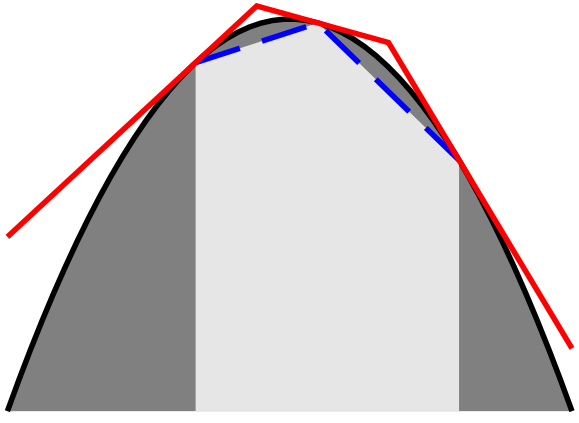


Density with hat and squeeze

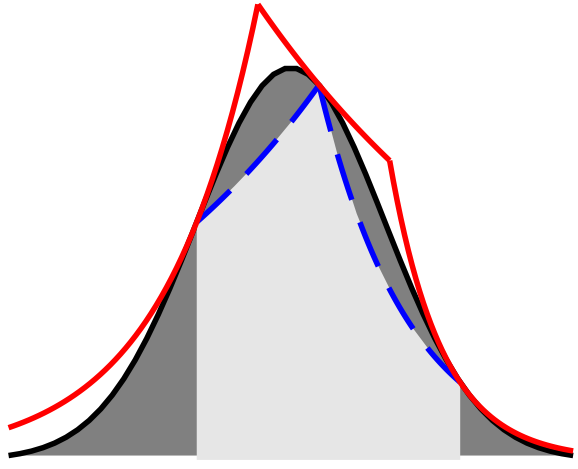
Use tangents and secants to construct hat and squeeze for the transformed density.

Transformed Density Rejection

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



Transformed density



Density with hat and squeeze

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

Hermite Interpolation

Idea:
Approximate function $g(x)$ on some interval $[b_0, b_1]$ by polynomial $h(x)$ of order $2n+1$ such that

$$g^{(k)}(b_i) = h^{(k)}(b_i) \quad \text{for } k = 0, \ldots, n \text{ and } i = 0, 1$$

Approximation error is $O(||g^{2n+2}(x)||_\infty \cdot (b_1 - b_0)^{2k+2})$.

Consequently, we can make approximation error as small es desired by splitting interval in shorter subintervals.

$n = 0 \quad \rightarrow \quad$ linear interpolation
$n = 1 \quad \rightarrow \quad$ cubic interpolation
$n = 2 \quad \rightarrow \quad$ quintic interpolation

ARVAG

A First Conclusion

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

- Only one piece of code.
- Performance controlled by a simple parameter.
- Sampling from truncated distributions.
- The marginal generation time does not depend on the density.
- They can be used for variance reduction techniques.
- The quality of the generated random numbers only depends on the underlying uniform random number generator.

ARVAG

Random Vectors

Random Vector Generation

Exact methods

- Conditional distribution method (Generalized inversion)
- Rejection Method

Markov chain sampler

- Metropolis-Hastings algorithm
- Gibbs sampler, Hit-and-Run sampler
- HITRO algorithm

Multivariate Rejection Method

Rejection principle exactly the same as for univariate distributions.

Problems:

- Construction of hat functions much more difficult.
- Curse of Dimensionality: Acceptance probability deteriorates quickly with dimension.

Example: Rejection from hypercube to generate uniform vectors in a ball:

dimension	2	3	10	50
probability	0.785	0.524	0.0025	10^{-28}

ARVAG

Markov Chain Sampler

Run a Markov chain whose stationary distribution is the required distribution.

Advantages:

- Algorithms are much simpler.
- More generally applicable.

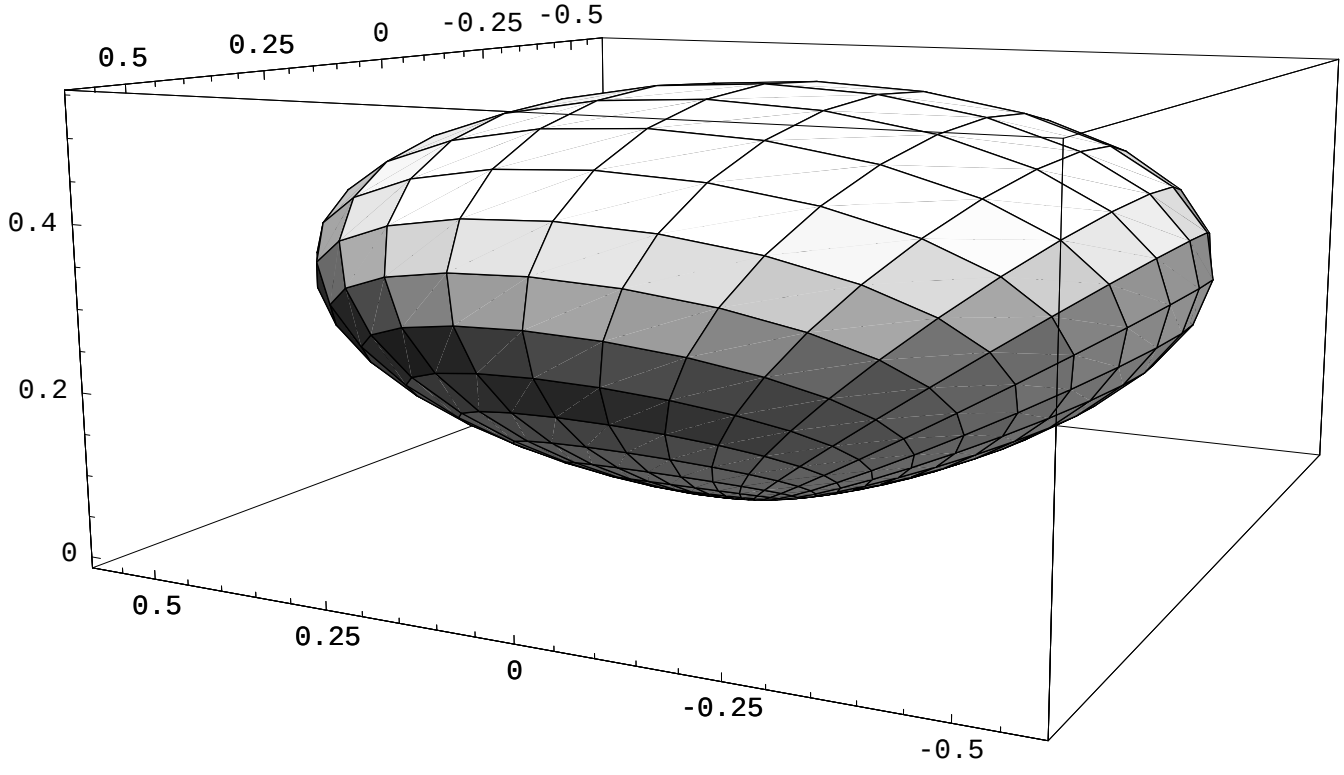
Disadvantages:

- No IID random vectors.
- Rate of convergence of the Markov chain is a problem.
  Only heuristic rules for convergence exist.

“Optimistic Algorithm” (Hörmann)

HITRO

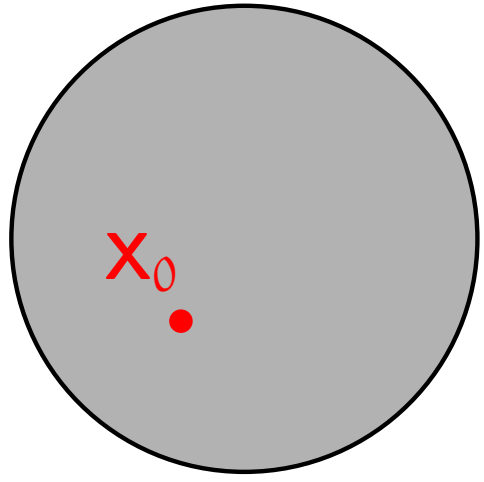
Combines the Hit-and-Run sampler with the multivariate Ratio-of-Uniforms method.



Hit-and-Run Sampler

Gerate a sample of random points uniformly distributed in some fixed but arbitrary bounded open set S:

- Choose a starting point $\mathbf{X}_0 \in S$ and set $k = 0$.
- Generate a random direction $\mathbf{d}_k$ with distribution $\mathcal{D}$.
- Generate λ_k uniformly distributed in $\Lambda_k = S \cap \{\mathbf{x} : \mathbf{x} = \mathbf{x}_k + \lambda \mathbf{d}_k\}$.
- Set $\mathbf{X}_{k+1} = \mathbf{X}_k + \lambda_k \mathbf{d}_k$ and $k = k + 1$.
- Repeat from Step 2

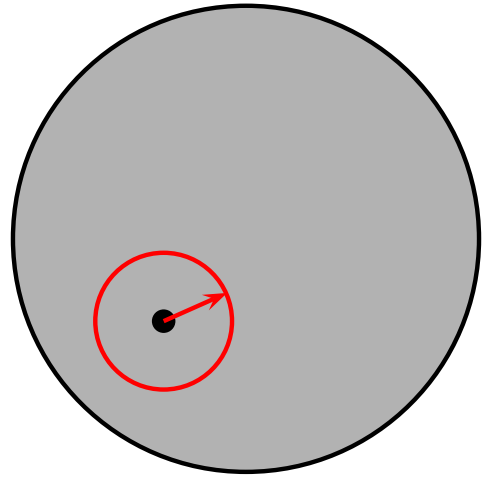


Smith (1984)

Hit-and-Run Sampler

Gerate a sample of random points uniformly distributed in some fixed but arbitrary bounded open set S:

- Choose a starting point $\mathbf{X}_0 \in S$ and set $k = 0$.
- Generate a random direction $\mathbf{d}_k$ with distribution $\mathcal{D}$.
- Generate λ_k uniformly distributed in $\Lambda_k = S \cap \{\mathbf{x} : \mathbf{x} = \mathbf{x}_k + \lambda \mathbf{d}_k\}$.
- Set $\mathbf{X}_{k+1} = \mathbf{X}_k + \lambda_k \mathbf{d}_k$ and $k = k + 1$.
- Repeat from Step 2

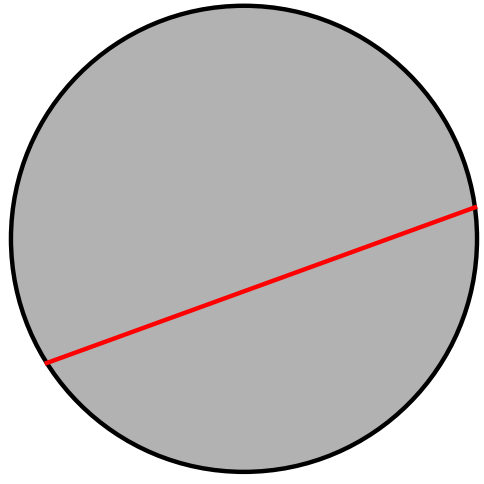


Smith (1984)

Hit-and-Run Sampler

Gerate a sample of random points uniformly distributed in some fixed but arbitrary bounded open set S:

- Choose a starting point $\mathbf{X}_0 \in S$ and set $k = 0$.
- Generate a random direction $\mathbf{d}_k$ with distribution $\mathcal{D}$.
- Generate λ_k uniformly distributed in $\Lambda_k = S \cap \{\mathbf{x} : \mathbf{x} = \mathbf{x}_k + \lambda \mathbf{d}_k\}$.
- Set $\mathbf{X}_{k+1} = \mathbf{X}_k + \lambda_k \mathbf{d}_k$ and $k = k + 1$.
- Repeat from Step 2

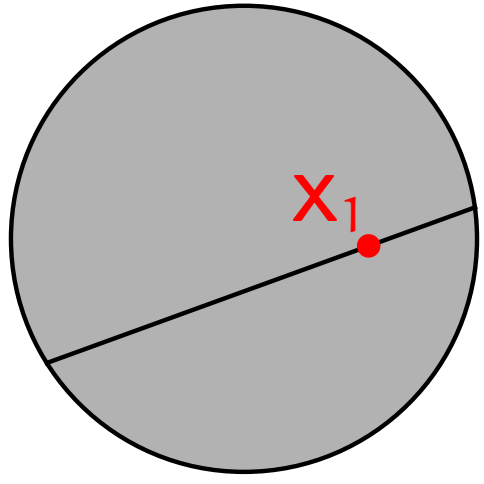


Smith (1984)

Hit-and-Run Sampler

Gerate a sample of random points uniformly distributed in some fixed but arbitrary bounded open set S:

- Choose a starting point $\mathbf{X}_0 \in S$ and set $k = 0$.
- Generate a random direction $\mathbf{d}_k$ with distribution $\mathcal{D}$.
- Generate λ_k uniformly distributed in $\Lambda_k = S \cap \{\mathbf{x} : \mathbf{x} = \mathbf{x}_k + \lambda \mathbf{d}_k\}$.
- Set $\mathbf{X}_{k+1} = \mathbf{X}_k + \lambda_k \mathbf{d}_k$ and $k = k + 1$.
- Repeat from Step 2

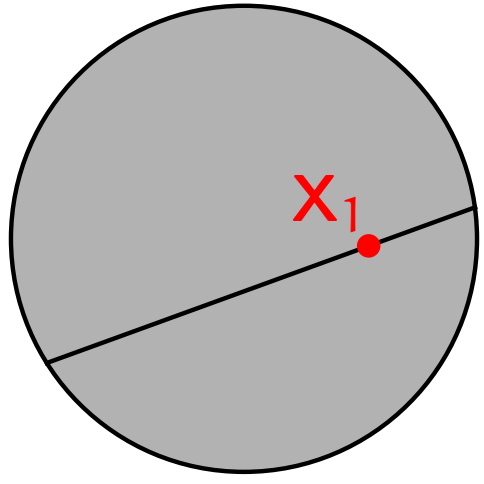


Smith (1984)

Hit-and-Run Sampler

Gerate a sample of random points uniformly distributed in some fixed but arbitrary bounded open set S:

- Choose a starting point $\mathbf{X}_0 \in S$ and set $k = 0$.
- Generate a random direction $\mathbf{d}_k$ with distribution $\mathcal{D}$.
- Generate λ_k uniformly distributed in $\Lambda_k = S \cap \{\mathbf{x} : \mathbf{x} = \mathbf{x}_k + \lambda \mathbf{d}_k\}$.
- Set $\mathbf{X}_{k+1} = \mathbf{X}_k + \lambda_k \mathbf{d}_k$ and $k = k + 1$.
- Repeat from Step 2



Smith (1984)

ARVAG

HITRO // Properties

The HITRO (Hit-and-run-Ratio-of-Uniforms) sampler is

- simple;
- easy to implement;
- relatively fast;
- works for many (not necessarily unimodal) distributions out of the box;
- performs similar to the Gibbs sampler
  but does not need a special and/or expensive generator for conditional distributions.

ARVAG

Implementation:

UNU.RAN

ARVAG

UNU.RAN

We have implemented these (and other) methods in a library, called UNU.RAN
(**U**niversal **N**on**U**niform **Ran**dom variate generators).

Features:

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

Can be downloaded from
http://statistik.wu-wien.ac.at/unuran/

ARVAG

UNU.RAN // Design

- Create a **distribution** object.
- Set **data** for distribution: obligatory and optional data (depending on distribution type).
  PDF, CDF, domain, mode, . . .
- Choose a **sampling method**.
- Change default **parameters** for chosen method (if required).
- Initialize **generator** object.
- Draw a **sample**.

Random Variates for Standard Distributions

```
// The Gaussian random variate Generator is
// method in class TRandom.
x = gRandom->Gaus(0,1);

// Mean (0) and sigma (1) are arguments.
```

Generation of Non-Standard Distributions

```
// Describe distribution by its density.
// Use class TF1
TF1 *f = new TF1("f","(1-x)*exp(-x*x)",-10,1);

// Get a random number.
x = f->GetRandom();

// “Histogram method”:
//  - Truncated domain
//  - Rough approximation of density
```

UNU.RAN

```
// Create UNU.RAN continuous distribution
TF1 *f = new TF1("f","(1-x)*exp(-x*x)",-10,1);
TUnuranContDist dist(f);
dist.SetDomain(-INFINITY,1);

// Create UNU.RAN class
TUnuran unur;

// Initialize
retval = unur.Init(dist);
if (!retval) {... catch error ...}

// Get a random number
x = unur.Sample();
```

UNU.RAN

```
// Create UNU.RAN continuous distribution
TF1 *f = new TF1("f","(1-x)*exp(-x*x)",-10,1);
TUnuranContDist dist(f);
dist.SetDomain(-INFINITY,1);

// Create UNU.RAN class
TUnuran unur;

// Initialize
retval = unur.Init(dist, "arou");
if (!retval) {... catch error ...}

// Get a random number
x = unur.Sample();
```

UNU.RAN

```
// Create UNU.RAN continuous distribution
TF1 *f = new TF1("f","(1-x)*exp(-x*x)",-10,1);
TUnuranContDist dist(f);
dist.SetDomain(-INFINITY,1);

// Create UNU.RAN class
TUnuran unur;

// Initialize
retval = unur.Init(dist, "arou; usedars=on");
if (!retval) {... catch error ...}

// Get a random number
x = unur.Sample();
```

UNU.RAN

```
// Create UNU.RAN continuous distribution
TF1 *f = new TF1("f","(1-x)*exp(-x*x)",-10,1);
TUnuranContDist dist(f);
dist.SetDomain(-1,1);

// Create UNU.RAN class
TUnuran unur;

// Initialize
retval = unur.Init(dist, "arou");
if (!retval) {... catch error ...}

// Get a random number
x = unur.Sample();
```

UNU.RAN

```
// Create UNU.RAN continuous distribution
TF1 *f = new TF1("f","(1-x)*exp(-x*x)",-10,1);
TUnuranContDist dist(f);
TF1 *fc = new TF1("fc",
     "0.49+0.28*exp(-x*x)+0.49*Erf(x)",-10,1);
dist.SetCdf(fc);
dist.SetDomain(-INFINITY,1);
// Create UNU.RAN class and initialize
TUnuran unur;
retval = unur.Init(dist, "hinv");
if (!retval) {... catch error ...}
// Get a random number
x = unur.Sample();
```

Some Results with TRandom3

Distribution	Method	Time (ns)
Uniform	Random	30
Exponential	Random	170
Gaussian	Random	160
Gaussian	TF1.GetRandom	310
Gaussian	UNU.RAN arou	140
Maxwell	UNU.RAN arou	140
Maxwell	UNU.RAN hinv	190
MNormal (2)	UNU.RAN hitro	1750
MNormal (5)	UNU.RAN hitro	2150
MNormal (10)	UNU.RAN hitro	2820
MNormal (100)	UNU.RAN hitro	34790

ARVAG

UNU.RAN

Manual:

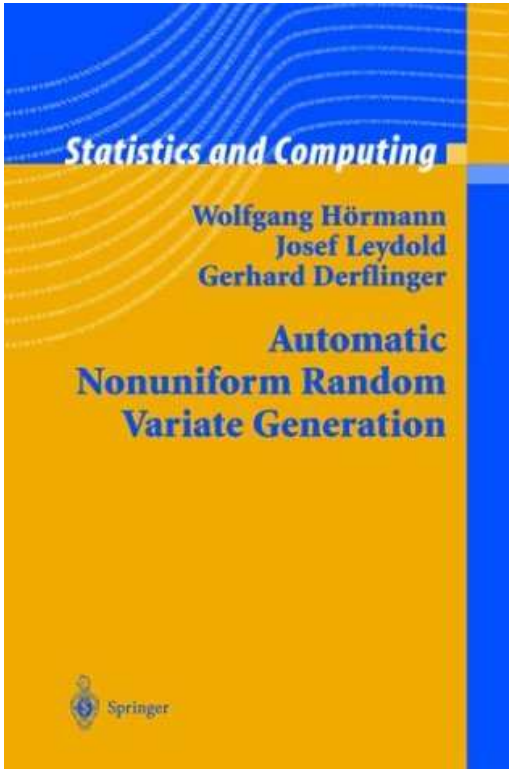
`http://statmath.wu-wien.ac.at/unuran/`

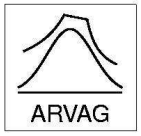
Request for help, Comments, Suggestions:

`unuran@statmath.wu-wien.ac.at`

Further Reading

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





Thank you

ARVAG

References

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

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

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.

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

Smith, R. L. (1984). Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research 32*, 1296–1308.