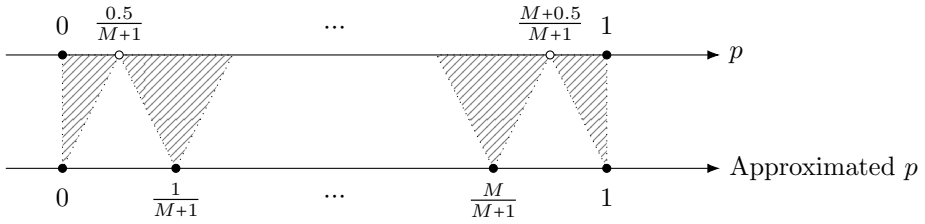
Notation. In this paper we will write $\lfloor\cdot\rfloor$ for the floor function, and $[\![x]\!] = \lfloor x + 0.5\rfloor$ for the rounding function. The "boxed" $\boxed{\text{quantities}}$ in the algorithms can be pre-computed to speed up calculations.

Assumption 1. In all of the algorithms we assume that all integers live in $\mathbb{Z}/(M+1)$, where M is some positive integer.

1 Bernoulli and Discrete Uniform distributions

Before we proceed to more complicated algorithms, a couple of words on generating events with a prescribed probability p, $0 \leqslant p \leqslant 1$. Suppose $X \sim \text{Uniform}\{0, 1, \cdots, M\}$. Note, that each value of X occurs with probability $1/(M+1)$. Rounding p to the nearest multiple of $1/(M+1)$ would be ideal, with the incurred error begin upper-bounded (sharply) by $0.5/(M+1)$:



Unfortunately, this mapping yields $M+2$ possible values, leading to technical complications, since we are restricted to $\mathbb{Z}/(M+1)$. To address the issue, we round p to the nearest multiple of $1/M$ instead.

Notation. For any p, $0 \leqslant p \leqslant 1$, we will write

$$p^\star = \begin{cases} [\![M\,p]\!] & \text{if } 0 \leqslant p \leqslant 0.5, \\ M - [\![M(1-p)]\!] & \text{if } 0.5 < p \leqslant 1. \end{cases}$$

Events with probability p may be approximated with either $\{X < p^\star\}$ or $\{X \leqslant p^\star\}$.

Target probability	$p^\star$	$\mathsf{P}(X < p^\star)$	$\mathsf{P}(X \leqslant p^\star)$
$0 \leqslant p < 0.5/M$	0	0	$1/(M+1)$
$0.5/M \leqslant p < 1.5/M$	1	$1/(M+1)$	$2/(M+1)$
...			
$(M-1.5)/M < p \leqslant (M-0.5)/M$	$M-1$	$(M-1)/(M+1)$	$M/(M+1)$
$(M-0.5)/M < p \leqslant 1$	M	$M/(M+1)$	1

It is not hard to see that this will lead to errors not exceeding $1.5/M$.

Algorithm (Bernoulli).

1. Let p, $0 \leqslant p \leqslant 1$, be the parameter of Bernoulli distribution. If $p = 1$, set $\boxed{S} = 0$ and $\boxed{H} = 1$.
2. If $p < 1$, set $\boxed{S} = 1$ and $\boxed{H} = p^\star$.
3. Generate $\hat{U} \sim \text{Uniform}\{0, 1, \cdots, M\}$. If $\boxed{S} \cdot \hat{U} < \boxed{H}$ return *true*, otherwise return *false*.

The algorithm above has $1/M$ as an upper bound for the probabilities of generated events. Now that we have a Bernoulli(p) generator, we will proceed to generation of $\text{Uniform}\{a, a+1, \cdots, b\}$, where $a \leqslant b$ are integers. Unlike the algorithm above, this generation can be done precisely with a rejection method. To this end, we will partition the set $\{0, 1, \cdots, M\}$ into blocks of size $(b-a+1)$ with possibly one block of a smaller size. If the smaller block is selected—which happens with probability $((M+1) \mod (b-a+1))/(M+1)$—we start over.

Algorithm (Discrete Uniform).

1. Let $a \leqslant b$ be the parameters of Uniform distribution, and set $\boxed{N} = b - a$.
2. If $\boxed{N} = 0$, set $\boxed{B} = M$.
3. If $\boxed{N} \neq 0$, set $\boxed{B} = 1 + \lfloor (M - \boxed{N})/(\boxed{N} + 1) \rfloor$.
4. Generate $\hat{U} \sim \text{Uniform}\{0, 1, \cdots, M\}$, and set $K = \lfloor \hat{U} / \boxed{B} \rfloor$ to be the block index where $\hat{U}$ landed. Note that $K \mid \left\{K \leqslant \boxed{N}\right\}$ is conditionally Uniform $\left\{0, 1, \cdots, \boxed{N}\right\}$.
5. If $K \leqslant \boxed{N}$, accept $a + K$.
6. If $K > \boxed{N}$, return to step 4.

2 Discrete distributions with finite support

2.1 Overview

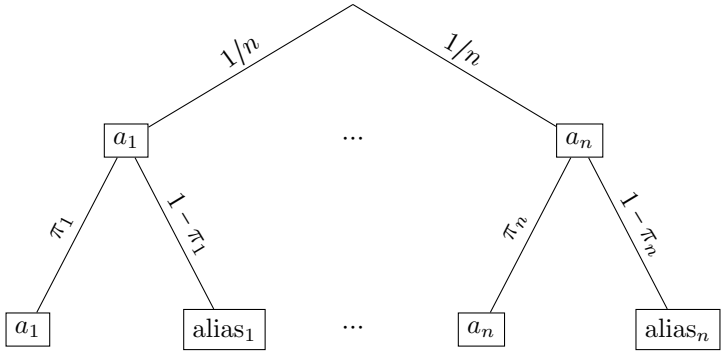
In this section we will consider the so-called "alias" algorithm for generating random variates from discrete distributions with finite support, based off of a paper by Vose [1991].

2.2 Formal Setup

Suppose we are dealing with a random variable X with a discrete distribution with finite support over $n \geqslant 1$ values:

$$\mathsf{P}(X = a_i) = p_i, \qquad 1 \leqslant i \leqslant n,$$

with $p_1 + \cdots + p_n = 1$ and $p_i > 0$, $1 \leqslant i \leqslant n$. The algorithm is two-stage: at the first stage we select one of the elements with probability $(1/n)$; at the second stage we either take the chosen item, or its alias.



The idea behind setting up probabilities correctly is an iterative one, where at each step we eliminate one branch. More specifically, select j-th branch such that $p_j \leqslant 1/n$. Then setting $\pi_j = n\,p_j$ and ensuring that j-th element will not appear as an alias at future steps will take care of the left sub-branch. Choosing the j-th alias as any element, say k-th, with $p_k > 1/n$, and updating $p_k = p_k - (1 - \pi_j)/n$, will take care of the right sub-branch.

2.3 Alias Algorithm

Algorithm (Continuous generators).

1. Create an array of conditional probabilities, $\boxed{\pi_i}$, $1 \leqslant i \leqslant n$, and "aliases", $\boxed{b_i} = \boxed{a_i}$, $1 \leqslant i \leqslant n$.

2. Categorize each probability as either "small" or "large":

$$S = \{i \,|\, p_i \leqslant 1/n\}\,, \quad L = \{i \,|\, p_i > 1/n\}\,.$$

3. While $S \neq \varnothing$ and $L \neq \varnothing$:
   (a) Take $j \in S$ (index of a small element) and $k \in L$ (index of a large element).
   (b) Set $\boxed{b_j} = \boxed{a_k}$ (pair up the small element with the large element).
   (c) Set $\boxed{\pi_j} = np_j \leqslant 1$ (record the cutoff value for the small element).
   (d) Since this large element is now an "alias" for the small element, we will adjust the probability of it being selected in the future. Let $\delta_j = 1/n - p_j \geqslant 0$ be the probability of element k being selected in the j-th branch.
   (e) Overwrite $p_k = p_k - \delta_j$, making sure that once we reach k-th branch, its chances of being selected will be smaller.
   (f) If $p_k \leqslant 1/n$, remove k from L and place it in S (what used to be large is now small).
   (g) Remove j from S.
4. For each $j \in S$ set $\boxed{\pi_j} = 1$; for each $k \in L$ set $\boxed{\pi_k} = 1$. This will mitigate some rounding errors.
5. The steps above are the initialization stage and will have to be completed only once. The generation itself proceeds as follows:
   (a) Generate $K \sim \text{Uniform}\,\{1, \cdots, n\}$.
   (b) Generate $U \sim \text{Uniform}[0, 1)$.
   (c) If $U < \boxed{\pi_K}$, accept $\boxed{a_K}$.
   (d) If $U \geqslant \boxed{\pi_K}$, accept $\boxed{b_K}$.

Algorithm (Discrete uniform integer generators).

1. Create an array of cutoff values, $\boxed{\hat{\pi}_i}$, $1 \leqslant i \leqslant n$, and "aliases", $\boxed{b_i} = \boxed{a_i}$, $1 \leqslant i \leqslant n$.
2. Categorize each probability as either "small" or "large":

$$S = \{i \,|\, np_i \leqslant 1\}\,, \quad L = \{i \,|\, np_i > 1\}\,.$$

3. While $S \neq \varnothing$ and $L \neq \varnothing$:
   (a) Take $j \in S$ (index of a small element) and $k \in L$ (index of a large element).
   (b) Set $\boxed{b_j} = \boxed{a_k}$.
   (c) Set $\boxed{\hat{\pi}_j} = (np_j)^\star \leqslant M$. If $np_j = 1$, then set $\boxed{b_j} = \boxed{a_j}$.
   (d) Set $\delta_j = 1 - np_j \geqslant 0$.
   (e) Overwrite $np_k = np_k - \delta_j$.
   (f) If $np_k \leqslant 1$, remove k from L and place it in S.
   (g) Remove j from S.
4. For each $j \in S$ set $\boxed{b_j} = \boxed{a_j}$; for each $k \in L$ set $\boxed{b_k} = \boxed{a_k}$.
5. The steps above are the initialization stage and will have to be completed only once. The generation itself proceeds as follows:
   (a) Generate $K \sim \text{Uniform}\,\{1, \cdots, n\}$.
   (b) Generate $\hat{U} \sim \text{Uniform}\,\{0, \cdots, M\}$.
   (c) If $\hat{U} < \boxed{\hat{\pi}_K}$, accept $\boxed{a_K}$.
   (d) If $\hat{U} \geqslant \boxed{\hat{\pi}_K}$, accept $\boxed{b_K}$.

3 Unimodal absolutely continuous distributions

3.1 Overview

The aim of this section is two-fold: first, to give an overview of the Ziggurat algorithm Marsaglia and Tsang [undated] for unimodal absolutely continuous distributions; second, to provide one implementation if the algorithm. One of the beauties of the Ziggurat algorithm lies in the fact that while almost all the time it is a very efficient version of the classical rejection method, it allows to handle densities with infinite support—assuming it is known how to sample from the tail (in the normal case see, e.g., Marsaglia [1964]).

3.2 Formal Setup

In what follows, let f denote the density function; θ denote the mode of the distribution (argument where f achieves its maximum); and T denote its survival function:

$$T(x) = \int_x^\infty f(t)\,dt.$$

We will start with the monotone decreasing density case; the monotone increasing case can be handled similarly.

The idea behind the Ziggurat algorithm is to cover the density function with $n \geqslant 2$ horizontal layers of the same area, where all the layers except for the bottom one are rectangular. More formally, we start with a partition of the vertical interval

$$0 = f_0 < f_1 < \cdots < f_{n-1} < f_n = f(\theta).$$

For $1 \leqslant k \leqslant n-1$ let

$$a_k = \inf\big\{x : f(x) > f_k\big\}, \quad b_k = \sup\big\{x : f(x) > f_k\big\};$$

put $a_n = b_n = \theta$, and define the layers by

$$\begin{cases} L_0 = \big\{(x,y) : 0 \leqslant y \leqslant \min\{f_1, f(x)\}\big\}, \\ L_k = [a_k, b_k] \times [f_k, f_{k+1}], \quad 1 \leqslant k \leqslant n-1. \end{cases}$$

This layering is illustrated in Figure 1.

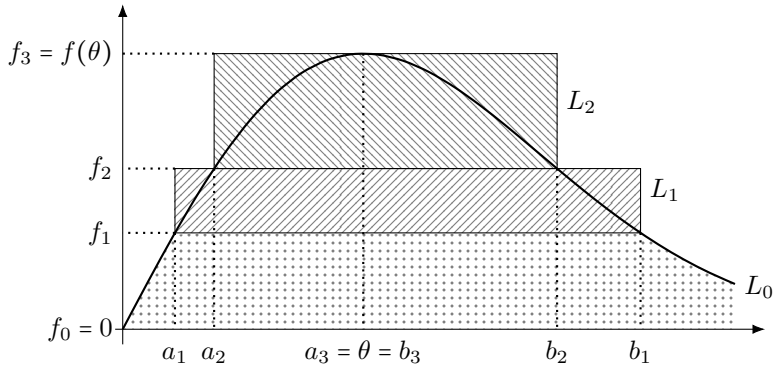


Figure 1: Unimodal Ziggurat algorithm with 3 layers.

We want to choose $f_1, \cdots, f_{n-1}$ in such a way as to make sure that the area of each layer is the same:

$$|L_0| = |L_1| = \cdots = |L_{n-1}| = V.$$

Thus, one arrives at a (non-linear) system of n equations with n unknowns:

$$\begin{cases} V = (1 - T(a_1)) + (b_1 - a_1)\, f_1 + T(b_1) \\ V = (b_k - a_k)\,\big(f_{k+1} - f_k\big), \quad 1 \leqslant k \leqslant n-1. \end{cases} \tag{1}$$

Proposition 1. *System (1) has a unique solution.*

Having set up the basics, let us review the principles of the Ziggurat algorithm.

- The first step is to select one layer (uniformly) at random.
- The second step is to generate a (uniform) random point inside the selected layer. If the point happens to be under the graph of the density function, we accept it. Otherwise, we start over from the first step.

To facilitate these steps, note that every layer has a rectangular subset that lies entirely under the density curve. Specifically, all

$$B_k = [a_{k+1}, b_{k+1}] \times [f_k, f_{k+1}] \subseteq L_k, \quad 0 \leqslant k \leqslant n-1,$$

lie under the graph of f (for the top layer the box B_{n-1} is trivial). In light of this, the second step of the algorithm becomes substantially simpler if the point sampled from layer L_k falls inside B_k. The probability of this happening, which we will refer to as the *simple coverage probability*, is

$$p_k = \frac{|B_k|}{|L_k|}.$$

It is convenient to write these probabilities in terms of the widths of the layers (the term "width" is accurate for all layers but the bottom one, in which case we interpret it as the width of a rectangle of the same height and area). Let

$$a_0 = a_1 - (1 - T(a_1))/f_1 \quad \text{and} \quad b_0 = b_1 + T(b_1)/f_1,$$

and set

$$\begin{cases} w_k = b_k - a_k, & 0 \leqslant k \leqslant n-1, \\ w_n = 0. \end{cases} \tag{2}$$

Then the simple coverage probabilities become

$$p_k = \frac{w_{k+1}}{w_k} \quad \text{for } 0 \leqslant k \leqslant n-1. \tag{3}$$

Furthermore, let α_k and β_k, $0 \leqslant k \leqslant n-1$, denote the probabilities of landing to the left and right of B_k, respectively. Then

$$\begin{cases} \alpha_k = (a_{k+1} - a_k)/w_k, \\ \beta_k = (b_k - b_{k+1})/w_k, \end{cases} \quad \text{for } 0 \leqslant k \leqslant n-1. \tag{4}$$

Note, that in the case of monotone densities, either all $\alpha_k = 0$ or all $\beta_k = 0$, $0 \leqslant k \leqslant n-1$. It is not difficult to see that

$$\mathsf{P}\big(\text{simple coverage}\big) = \frac{1}{n}\sum_{k=0}^{n-1} \mathsf{P}\big(\text{simple coverage} \mid \text{layer } k\big) = \frac{p_0 + \cdots + p_{n-1}}{n},$$

$$\mathsf{P}\big(\text{rejection}\big) = \frac{1}{n}\sum_{k=0}^{n-1} \mathsf{P}\big(\text{rejection} \mid \text{layer } k\big) = 1 - \frac{1}{nV}.$$

3.3 Ziggurat Algorithm

The algorithm of Marsaglia and Tsang [undated] and Doornik [1997] for $n \geqslant 2$ layers is summarized below.

Algorithm (Continuous generators).

1. Generate $K \sim \text{Uniform}\{0, 1, \cdots, n-1\}$, the zero-based layer index.
2. Generate $U \sim \text{Uniform}[0, 1)$, the relative "horizontal position" inside L_K.

3. If $K = 0$:
   (a) If $U < \boxed{\alpha_0}$, accept a variate from the left tail.
   (b) If $U \geqslant \boxed{1-\beta_0}$, accept a variate from the right tail.
   (c) If $\boxed{\alpha_0} \leqslant U < \boxed{1-\beta_0}$, accept $\boxed{a_0} + U \cdot \boxed{w_0}$, since the random point landed inside B_0.
4. If $K \neq 0$:
   (a) Let $X = \boxed{a_K} + U \cdot \boxed{w_K}$ be the abscissa of the random point inside L_K.
   (b) If $\boxed{\alpha_K} \leqslant U < \boxed{1-\beta_K}$, accept X, since the point landed inside B_K.
   (c) Generate $W \sim \text{Uniform}[0,1)$, the relative vertical position inside L_K.
   (d) If $\boxed{f_K} + W \cdot \boxed{(f_{K+1} - f_K)} < f(X)$, accept X, since the point landed under the graph of f.
   (e) Return to step 1.

In practice one usually has to deal with discrete uniform integer generators. For simplicity of presentation, we will assume that the possible values are $\{0, 1, \cdots, M\}$; typically, M would be a Mersenne number. The previous algorithm can be adjusted to take advantage of integer arithmetic in the following way:

Algorithm (Discrete uniform integer generators).

1. Generate $K \sim \text{Uniform}\{0, 1, \cdots, n-1\}$, the zero-based layer index.
2. Generate $\hat{U} \sim \text{Uniform}\{0, 1, \cdots, M\}$, the up-scaled relative "horizontal position" inside L_K.
3. If $K = 0$:
   (a) If $\hat{U} < \boxed{[\![(M^\star+1)\alpha_0]\!]}$, accept a variate from the left tail.
   (b) If $\hat{U} > \boxed{[\![(M^\star+1)(1-\beta_0)]\!]}$, accept a variate from the right tail.
   (c) If $\boxed{[\![(M^\star+1)\alpha_0]\!]} \leqslant \hat{U} \leqslant \boxed{[\![(M^\star+1)(1-\beta_0)]\!]}$, accept $\boxed{a_0} + \hat{U} \cdot \boxed{w_0/(M+1)}$, since the random point landed inside B_0.
4. If $K \neq 0$:
   (a) Let $X = \boxed{a_K} + \hat{U} \cdot \boxed{w_K/(M+1)}$ be the abscissa of the random point inside L_K.
   (b) If $\boxed{[\![(M^\star+1)\alpha_K]\!]} < \hat{U} < \boxed{[\![(M^\star+1)(1-\beta_K)]\!]}$, accept X, since the point landed inside B_K.
   (c) Generate $\hat{W} \sim \text{Uniform}\{0, 1, \cdots, M\}$, the up-scaled relative vertical position inside L_K.
   (d) If $\boxed{f_K} + \hat{W} \cdot \boxed{(f_{K+1} - f_K)/(M+1)} < f(X)$, accept X, since the point landed under the graph of f.
   (e) Return to step 1.

Note, that if we know in advance that $\alpha_0 = \cdots = \alpha_{n-1} = 0$ or $\beta_0 = \cdots = \beta_{n-1} = 0$, certain steps in the algorithm can be simplified (or skipped entirely).

Remark 1. In most cases, we think it is reasonable to make the following assumptions.

- If $\alpha_j = 0$ for some j, then all $\alpha_k = 0$, $0 \leqslant k \leqslant n-1$.
- If $\beta_j = 0$ for some j, then all $\beta_k = 0$, $0 \leqslant k \leqslant n-1$.

Remark 2. Also note, that only $\lceil \log_2(n) \rceil$ bits are necessary to generate K; the remaining bits may be used, e.g., to implement vectorized Ziggurat versions, or store the random index for repeated iterations.

References

Jurgen A. Doornik. An improved ziggurat method to generate normal random samples. *Communications in Statistics – Theory and Methods*, 26(5):1253–1268, December 1997.

George Marsaglia. Generating a variable from the tail of the normal distribution. *Technometrics*, 6(1): 101–102, February 1964. ISSN 00401706.

George Marsaglia and Wai Wan Tsang. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5(i08), undated.

Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17(9):972–975, September 1991.