FJFI, ČVUT, Prague
Monte Carlo Methods

Zikkurat method for generating random numbers

Daniel Wohlrath

Winter* 2020/21

Contents

*Translated into english years later

1 Introduction

The Ziggurat method is an algorithm for generating pseudorandom numbers. Belonging to the class of rejection methods, this algorithm depends on a generator of uniformly distributed numbers, typically a pseudorandom number generator or precomputed tables.

The algorithm is primarily designed to generate values from a monotonically decreasing probability density. It can also be applied to unimodal symmetric densities, such as the density of the normal distribution, see subsection 3.1. With slight modification, however, it can be used for more complex distributions, see subsection 3.2. The method was developed in the 1980s by George Marsaglia and Wai Wan Tsang. The term *Ziggurat* is associated with the procedure of the algorithm, in which the respective probability density is gradually covered with rectangles that resemble a Babylonian structure - zikkurat.

2 Zikkurat Method

2.1 Creation of Rectangles

The method is based on covering the area under the decreasing probability density with a set of rectangles of equal area and a "bottom bar", consisting of a rectangle of smaller area and the tail of the distribution, see Figure 1.

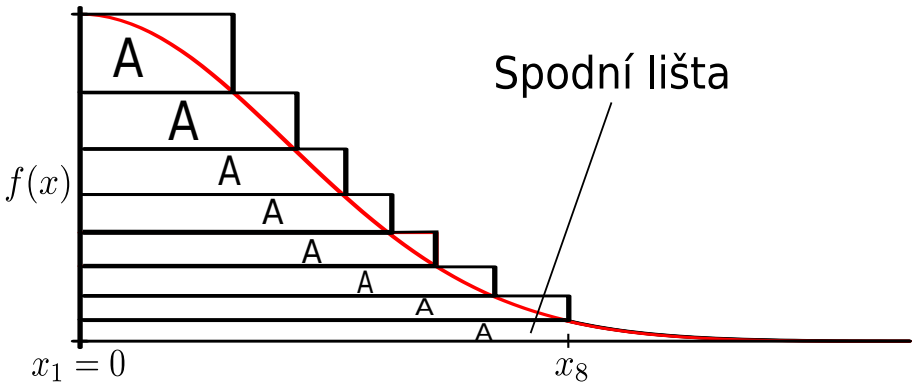


Figure 1: Coverage of the decreasing function f (in red) by 7 rectangles of equal area of size A and a bottom bar with an area of A

In practice, 63, 127, or 255 rectangles of area A are used. Therefore, if we choose, for example, $n = 127$ and the end point x_{128}, then we calculate the required area of the rectangles A as

$$A = f(x_{128}) \cdot x_{128} + T\,, \tag{1}$$

where T denotes the area under the curve for $x > x_{128}$. Denoting $f(x_i) = f_i$, we can then determine the respective boundary points of the rectangles from the relationships

$$f_i = \frac{A}{x_{i+1}} \cdot f_{i+1}\,,$$
$$x_i = f^{-1}(f_i)$$

for $i \in \{1, \ldots, 127\}$ successively. The task is then to find[1] such x_{128} that $x_1 = 0$.

[1] e.g., by the bisection method

2.2 Random Number Generation

Once we have a suitable set of points $\{x_1, \ldots, x_{128}\}$ available, we can proceed to the second part of the algorithm, namely to generating random numbers from this density.

First, number the individual rectangles (layers) from top to bottom 1 to 128, so the first layer is the narrowest and also the highest, the 128. layer touches the x-axis and can be of infinite length. Then we generate uniformly distributed numbers i from the set $\{1, \ldots, 128\}$ and U from the interval $(0, 1)$. For $i \neq 128$, if

$$U \cdot x_{i+1} < x_i \,, \tag{2}$$

then $U \cdot x_{i+1}$ is a realization of the random variable with the given density. If inequality (2) does not occur[2], we generate another number Z from the uniform distribution on the interval $(0, 1)$, and set the auxiliary variable $\tilde{y}$ equal to:

$$\tilde{y} = Z \cdot (f_i - f_{i+1}) \,.$$

If

$$\tilde{y} < f(U \cdot x_{i+1}) \,,$$

then $U \cdot x_{i+1}$ is the desired realization, otherwise, we generate a new i representing the examined layer.

2.2.1 Bottom Layer and Fall-back Algorithm

Now let's discuss the case when $i = 128$, i.e., we have selected the bottom layer adjacent to the x-axis. The bottom layer can be divided into a rectangular *main part* and a tail, having, while maintaining the notation in expression (1), respective areas $f_{128} \cdot x_{128}$ and T. If we define an auxiliary variable $x_{257} = A/f_{256}$, we can then partially use the previous procedure:

We generate a number $\hat{U} \sim R(0, 1)$[3] and if

$$\hat{U} \cdot x_{257} < x_{256} \,, \tag{3}$$

then $\hat{U} \cdot x_{257}$ is the sought realization. If inequality (3) does not occur, we are forced to generate a random number from the tail, using the so-called *fall-back algorithm*, which depends on the specific shape of the distribution.

3 Implementation of the Method

3.1 Normal Distribution

When generating random numbers from the standardized normal distribution, we divide the density into 2 parts, see Figure 2, and apply the Ziggurat method to its right wing. We take advantage of the fact that the method does not require the function to be normalized[4], so we will work only with the function $f(x) = \exp(\frac{-x^2}{2})$. The correct distribution of 128 rectangles is obtained by the initial choice of $x_{128} = 3.442619855899$, which corresponds to the area of the rectangle A approximately 0.0099. If we generate from the bottom layer 128, the fall-back algorithm according to [1] is as follows:

Generate $U_1, U_2 \sim R(0, 1)$, then

1. set $x = -\ln(U_1)/x_{128}$
2. set $y = -\ln(U_2)$
3. if $2y > x$, then $x + x_1$ is the sought realization
4. otherwise, return to step 1.

Finally, we individually decide about the respective realizations, whether they lie in the left or right wing of the density. Generating random numbers from an alternative distribution, we multiply our previously obtained realization by a factor of -1 with a probability of $1/2$ and do not multiply it by anything with the complementary probability, also $1/2$.

[2] For layer no. 1 it never occurs
[3] R denotes uniform distribution
[4] That is $\int f = 1$

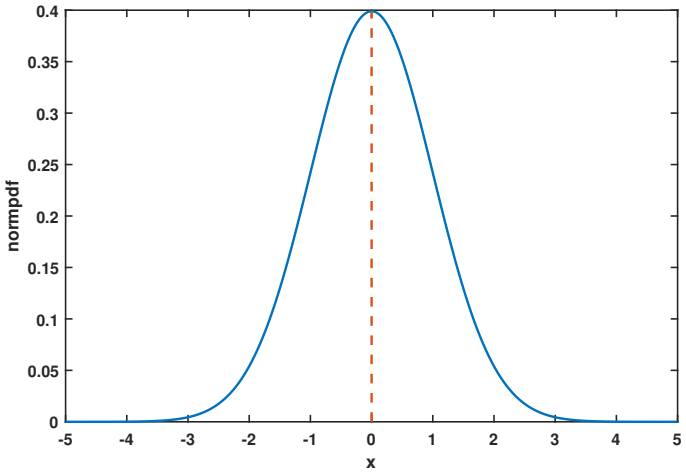


Figure 2: Division of the density of the normal distribution into 2 parts

When generating 10^6 random numbers using the above-described method, the Lilliefors test was conducted, which did not reject the normality of the generated numbers with a p-value less than 0.12. The Kolmogorov-Smirnov test also did not reject the hypothesis, with a p-value of 0.36. Finally, Pearson's χ^2 test also did not reject the null hypothesis, with a p-value of 0.35. Figure 3 illustrates a good match between the distribution of generated numbers and the standardized normal distribution.

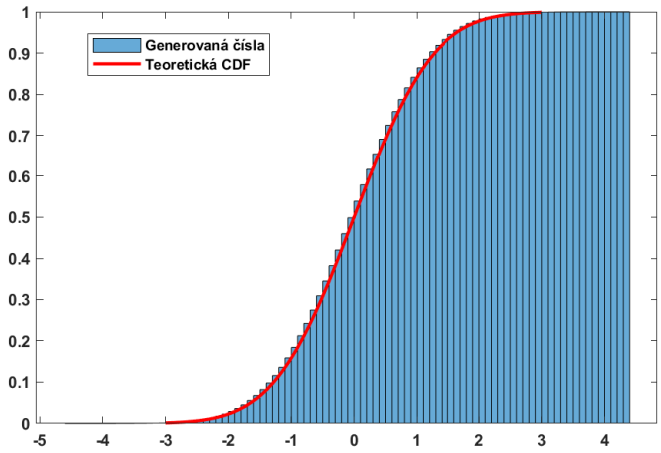


Figure 3: Agreement of theoretical and empirical distribution functions when generating random numbers from the normal distribution

3.2 GIG Distribution

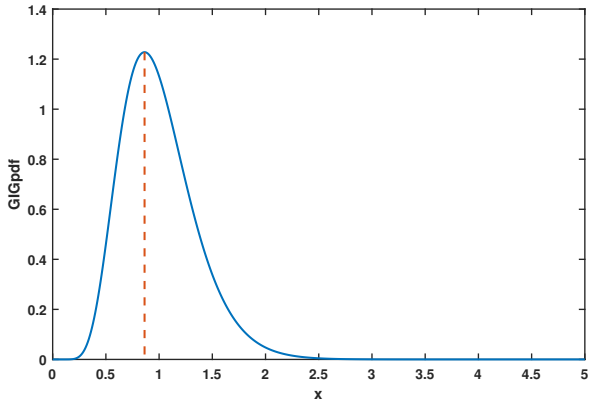
The Generalized Inverse Gaussian (GIG) distribution has a probability density function in the form

$$f(x) = \frac{(\frac{a}{b})^{p/2}}{2\mathcal{K}_p(\sqrt{ab})}\Theta(x)x^{p-1}\exp\left(-\frac{ax+\frac{b}{x}}{2}\right), \tag{4}$$

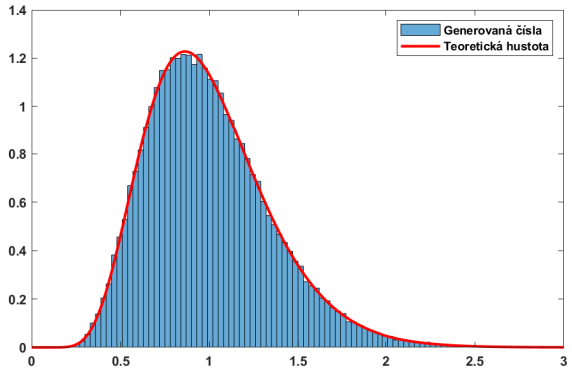
where $\Theta(x)$ is the Heaviside step function, $\mathcal{K}_p$ is the modified Bessel function of the second kind, and for the parameters $a, b > 0, , p \in \mathbb{R}$.

For a special choice of parameters $p = 6, b = 2$ and under the condition of mean value $\int xf(x), \mathrm{d}x = 1$, we subsequently obtain $a = 14.2655$. If we want to apply the Ziggurat method to this more complex distribution, it is necessary to divide it into two decreasing parts.

For our special case, the density reaches its maximum at the point $x_{\max} = 0.8634$. By substituting into the cumulative distribution function of GIG, we find that the probability of realizing a variable in the interval $(0, x_{\max})$ is equal to $p = 0.3912$. Therefore, always before performing the algorithm, we decide[5], whether the following random number will be generated from the right or left side of the distribution, see Figure 4a.



(a)



(b)

Figure 4: (a) Split of GIG density function into 2 parts (b) Comparison of histogram of generated samples and theoretical GIG density function. **The method works**.

The left wing of the density has a compact support, so no form of the fall-back algorithm will be implemented in this part, as was the case, for example, when generating from the normal distribution. The

[5]e.g., by generating random numbers from the uniform distribution $(0, 1)$

left wing was divided into 128 rectangles of equal area, and the first part of the Ziggurat algorithm (without fall-back) was performed.

The right wing of the density has an infinite tail, so it is necessary to design a certain version of the fall-back algorithm. In the text [2], a very general procedure is proposed, using knowledge of the cumulative distribution function and the quantile function, or the complementary cumulative distribution function and the inverse complementary cumulative distribution function[6]. The authors in their text justify that it is appropriate to generate a number from the right tail of the distribution by the following procedure:

1. generate $U \sim R(0, 1)$,
2. set $x^* = ICCDF(UCDF(x_n))$, where n is the number of Ziggurat layers, i.e., in our case 128,
3. x^* is then the sought realization.

Since we do not have ICCDF available, we approached the problem approximately. We looked for such x^* that $CCDF(x^*) = U * CCDF(x_{128})$.

When generating 10^6 random numbers, we used graphical comparison of the histogram and theoretical density, see Figure 4b, and Pearson's χ^2 test of good fit. The hypothesis that the generated numbers come from the GIG distribution with the given parameters was not rejected, with a p-value of 0.27.

4 Conclusion

We have presented the procedure of the Ziggurat method for generating random numbers. Initially, we demonstrated its classic use on the standardized normal distribution, with Kolmogorov-Smirnov, Lilliefors, and Pearson's χ^2 tests of good fit not rejecting the hypothesis of normality of the generated values at the 5% significance level.

Subsequently, we attempted to modify the method to generate random numbers from non-monotonic, asymmetric densities. The demonstrated modification method was shown on the GIG distribution for a special choice of parameters. The success of the demonstration was illustrated in the generation of 10^6 numbers by comparing the histogram and theoretical density and evaluating Pearson's χ^2 test at the 5% significance level.

[6] Respectively denoted as cdf and icdf or ccdf and iccdf

References

[1] Wai Wan Tsang George Marsaglia. The ziggurat method for generating random variables. *Journal of Statistical Software*, 2000.

[2] Mohammad A. Charsooghi Morteza Jalalvand. Generalized ziggurat algorithm for unimodal and unbounded probability density functions with zest. 2018.