*mathematics*

MDPI

*Article*

# Critical Analysis of Beta Random Variable Generation Methods

**Elena Almaraz Luengo [1,*] and Carlos Gragera [2]**

[1] Department of Statistics and Operational Research, Faculty of Mathematical Science, Complutense University of Madrid, 28040 Madrid, Spain
[2] Faculty of Mathematical Science, Complutense University of Madrid, 28040 Madrid, Spain; cgrage01@ucm.es
* Correspondence: ealmaraz@ucm.es

**Abstract:** The fast generation of values of the beta random variable is a subject of great interest and multiple applications, ranging from purely mathematical and statistical ones to applications in management and production, among others. There are several methods for generating these values, with one of the essential points for their design being the selection of random seeds. Two interesting aspects converge here: the use of sequences as inputs (and the need for them to verify properties such as randomness and uniformity, which are verified through statistical test suites) and the design of the algorithm for the generation of the variable. In this paper, we analyse, in detail, the algorithms that have been developed in the literature, both from a mathematical/statistical and computational point of view. We also provide empirical development using R software, which is currently in high demand and is one of the main novelties with respect to previous comparisons carried out in FORTRAN. We establish which algorithms are more efficient and in which contexts, depending on the different values of the parameters, allowing the user to determine the best method given the experimental conditions.

**Keywords:** beta random variable; dieharder; hypothesis testing; NIST; pseudo-random number; simulation; statistical tests suite; TestU01

**MSC:** 11K45; 65C10



**Citation:** Almaraz Luengo, E.; Gragera, C. Critical Analysis of Beta Random Variable Generation Methods. *Mathematics* **2023**, *11*, 4893. https://doi.org/10.3390/math11244893

Academic Editor: Antonio Di Crescenzo

Received: 28 October 2023
Revised: 18 November 2023
Accepted: 20 November 2023
Published: 6 December 2023



## 1. Introduction

The study of different algorithms for generating random variables is currently a topic of great interest (see, for example, [1–5], among others).

The rapid generation of beta random variables is essential for simulating real models in a diverse group of disciplines. It is possible to define random variable (r.v.) generation as the process of obtaining a realisation of a r.v. from a target distribution.

The generation of random numbers or numbers that behave as such is crucial for obtaining sequences of values that come from other distributions, such as normal, exponential, or beta distributions. To this end, the simulation process begins with the generation of values that can be considered sequences of independent numbers which come from a uniform variable in the interval (0,1) (or, if working at bit level, 0–1 independent uniformly distributed values) that will constitute the starting point for the algorithmic generation of values of the desired probability distribution. It is therefore essential to start from quality values (i.e., values that really verify the desired properties of randomness and uniformity). It is at this point that we will take a closer look at some of the most important concepts related to random and pseudo-random sequences and the tests that must be carried out to check the quality of the sequences. In general, it is possible to speak of true random numbers and pseudo-random numbers (see, for example, [6–11] among others). The former are based on physical sources and do not need an initial input to be generated. They are not expected to show any cyclical patterns or correlations between the generated data. The latter are sets of values that, although generated through algorithms and an initial seed, resemble values from independent realisations of a uniform r.v. in the interval $(0,1)$ ($U(0,1)$). As they

are generated via deterministic algorithms, they are reproducible. Generators that produce the first type of numbers are called true random number generators (TRNGs), while the second type is generated by pseudo-random number generators (PRNGs). Given the high cost, both material and temporal, of using TRNGs, since it is necessary to work with an entropy source and a processing function, it is more common to work with PRNGs, since they are much faster and computationally more efficient. One line of research related to this generation is based on the creation of more powerful and secure algorithms that allow these sequences to be obtained while also providing a certain degree of security that prevents the data obtained from allowing knowledge of the underlying algorithm (in this sense, the study of the so-called cryptographically secure pseudo-random number generators is of interest). For more details on the generation of pseudo-random numbers, see [12].

Once the sequence of pseudo-random numbers has been obtained, and before being used for other purposes such as the simulation of a system or the generation of values of other random variables, it is necessary to check whether the properties of uniformity and randomness (and in the case of working in cryptographic applications, unpredictability) are verified. For this purpose, different hypothesis tests are used to test them from different perspectives. The aim is to measure whether the results obtained with the samples under study are compatible with those that should be obtained in the case of working with sequences that are random. By carrying out numerous tests on different sequences of a generator, it will be possible to decide on the goodness of the generator, and, therefore, it will be possible to decide whether it is appropriate to use the sequences in question for subsequent analyses. It is important to emphasise this aspect because various algorithms for generating beta random variables will be discussed below, starting from certain inputs that assume a certain prior distribution for which, in the event that this is not being verified, the results of such algorithms would not be as expected.

In order to check sequences effectively, different sets of tests known as test suites or test batteries are used. There are many suites in the literature, such as NIST STS 800-22 [13], Diehard [14], Dieharder [15], ENT [16], FIPS [17–19], and TestU01 [20]. The most currently used suites are NIST STS 800-22, Dieharder, and TestsU01. Once the sequences have been checked and found to be of quality, they can then be used in the downstream processes where they need to be used.

In this paper, we will focus on analysing the existing algorithms in the literature for the generation of beta random variables. The study will be carried out from a critical point of view and will analyse the mathematical/statistical and computational properties of such algorithms under the prior assumption that the inputs of such algorithms have been previously checked by means of a suite of statistical tests. This is an exhaustive and more complete review that includes all the methods developed, classified by typology, and provides an overview of the different methods used in the development of the methods. Additionally, we will carry out an empirical study to analyse the performance of the different algorithms using R software, which is one of the most demanded programs at present, both in the academic and research fields, and which also provides free access.

The outline of this work is as follows: in Section 2, the basic concepts about the distribution under study are explained; in Section 3, the main existing methods in scientific research for the generation of the beta r.v. are discussed in detail; in Section 4, a computational study on the efficiency of the previously explained methods is carried out; and finally in Section 5, the main conclusions of this work are given.

## 2. Preliminaries

A r.v. is a measurable function $f : (\Omega, \mathcal{A}, P) \to (\mathbb{R}, \mathcal{B})$, where $(\Omega, \mathcal{A}, P)$ is a probability space ($\Omega$ is the sample space, $\mathcal{A} \subset \mathcal{P}(\Omega)$ is a $\sigma$-algebra, and $P$ is the probability measure defined on $(\Omega, \mathcal{A})$), and $\mathcal{B}$ is Borel's $\sigma$-algebra over $\mathbb{R}$.

A beta r.v. with the parameters $\alpha > 0$ and $\beta > 0$ (denoted by $X \sim beta(\alpha, \beta)$) is a continuous r.v. with a density

$$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)},\ 0 \leq x \leq 1 \tag{1}$$

where $B(\alpha,\beta) = \int_0^1 u^{\alpha-1}(1-u)^{\beta-1}du = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}$.

The density of this distribution can take a number of interesting different forms depending on its parameters. This versatility in the forms it can take is what makes the beta r.v. so important in modelling real-world phenomena (often after re-scaling the defining interval to the interval $[a, b]$ via the transformation $a + (b-a)X$). In addition, special cases include the uniform r.v. when $\alpha = \beta = 1$; the gamma r.v. on the boundary when $\alpha \to \infty$, $\beta \to \infty$, and $\alpha/\beta$ remains constant; and the normal r.v. when $\alpha \to \infty$, $\beta \to \infty$, and $\alpha = \beta$ among the most relevant ones. Also, small transformations of the beta distribution provide many of the families of the Pearson distribution system [21]. By providing a relationship between these important distributions, the beta r.v. can be a suitable model where the more commonly used distributions fail. This versatility is important in simulation modelling, as it is often used when closed-form results are more difficult to obtain, such as when component lifetimes do not follow a normal, exponential or Erlang distribution [21]. This is why beta r.v. is widely used in Bayesian statistics [22], simulation [23,24], management and production models [25–27], control systems [28], and in the study of genome structures [29], machine learning [30], among many others.

## 3. Methods for Generating Beta Random Variables

In this section, we will describe the main methods for generating beta random variables, from the most general methods (i.e., they are not specifically designed to generate values of beta random variables, and they can be difficult to adapt and are, in general, more computationally inefficient as a result) to more specific and concrete methods.

### *3.1. Inverse Transform Method*

Let $X$ be a continuous r.v. with a cumulative distribution function $F$ which is continuous and strictly increasing in $(0,1)$. Let $F^{-1}$ be the inverse of $F$. The method starts by generating $U \sim U(0,1)$ and afterward takes $X = F^{-1}(U)$. To check the consistency of the algorithm, let us note the following for $x \in \mathbb{R}$: $P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x)$. For more details about this method and improvements, see, for example, [31].

One of the main characteristics of this method is its simplicity, as it allows the generation of random variables from only one uniform r.v. and the calculation of the inverse distribution function $F^{-1}$. Moreover, it preserves the structural properties of the underlying uniform pseudo-random number generator. This makes it possible to generate correlated random variables, generate order statistics efficiently, and obtain samples from truncated distributions or marginal distributions in a simple way [32]. But this method also has disadvantages, since it is only computationally efficient and accurate if $F^{-1}$ is known. For a beta r.v., this only happens when either of its two parameters is equal to one (see Algorithm 1). Regrettably, it is not always possible to obtain $F^{-1}$ (see [33,34]), and thus other approaches are employed, such as numerical methods (even if they are approximate). Among the most well-known methods of numerical inversion are the "root-finding algorithms", among which are Newton's method, the false position method [35], and the bisection method [32], but these algorithms are generally characterised by their time-consuming nature. Other methods use the interpolation of the tabulated values of $F$, as in [33]. Finally, in [31], another interesting algorithm was designed (NINIGL) that allowed continuous random variables such as beta ones to be generated by numerical inversion.

**Algorithm 1** Inverse transform method for generating $beta(\alpha, 1)$

**Require:** $\alpha$
**Output:** $X \sim beta(\alpha, 1)$
1: Generate $U \sim U(0,1)$
2: $X \leftarrow U^{\frac{1}{\alpha}}$
3: **return** $X$

In [31], a new method of inversion is proposed. It is the first algorithm of its kind based on error control, which can be applied to all smooth and bounded densities. The authors use polynomial interpolation techniques of inverse CDF and Gauss–Lobatto integration in their development. They apply their algorithm not only to beta variable generation, but also to normal, gamma, and t-distributions. This is a very fast algorithm with accuracy close to machine precision. This research line is currently being pursued, and we can highlight very recently published works that are based on the previous one, for example [36], where the author presents an inversion method designed in the varying parameter case, if a suitable density transformation can be found to avoid running the configuration for each parameter.

### *3.2. Composition Method*

This method is applied when the distribution function $F(x)$ of the variable to be generated can be expressed as a combination of other distributions $F_1, F_2, \ldots$ that are easier to sample than the original one. That is, $F(x) = \sum_{j=1}^{\infty} p_j F_j(x)$ with $p_j \geq 0$, $\sum_{j=1}^{\infty} p_j = 1$ and $F_j(\cdot)$ a cumulative distribution function of a r.v. $\forall\, j$. To verify the consistency of this method, note that: $P(X \leq x) = \sum_{j=1}^{\infty} P(J = j) P(X \leq x \mid J = j) = \sum_{j=1}^{\infty} p_j F_j(x) = F(x)$.

Algorithm 2 shows this method, it is suitable when the distribution we want to generate is itself a mixture. For other types of distributions, the application of this method can be complex. Therefore, this method is often used in conjunction with other principles.

**Algorithm 2** Composition method for generating $beta(\alpha, \beta)$ r.v. [37]

**Require:** Decomposition of the distribution function $F(x) = \sum_{j=1}^{\infty} p_j F_j(x)$
**Output:** Random sample $x$ from the $beta(\alpha, \beta)$ r.v.
1: Generate a random index $J$ such that $P(J = j) = p_j$ for $j = 1, 2, \ldots$
2: Generate $x$ with distribution $F_J(\cdot)$
3: **return** $x$

### *3.3. Acceptance-Rejection (A–R) Method*

This method was proposed by John von Neumann in 1951 [38]. It is a flexible and efficient method for generating continuous random variables. The concept of acceptance–rejection consists of generating random samples from a given distribution and discarding some of them so that the remaining samples follow the desired distribution. This idea, together with the results described hereafter (whose proofs can be seen in [39]), will enable the method to be established and its consistency to be justified.

**Theorem 1.** *Let $f(x)$ be a density function and $c$ be a nonnegative constant. If the random pair $(X, Y)$ is uniformly distributed in $G_{cf} = \{(x, y) : 0 < y \leq cf(x)\}$, then $X$ is a r.v. with a density $f(x)$.*

**Theorem 2.** *Let $(X_1, Y_1), (X_2, Y_2), (X_3, Y_3), \ldots$ be a sequence of independent and identically distributed (i.i.d.) random pairs distributed uniformly in a set $A$. Then, for a (measurable) subset $B \subset A$ with $P((X, Y) \in B) > 0$, the subsequence of all pairs $(X_i, Y_i) \in B$ is a sequence of i.i.d. pairs uniformly distributed in $B$.*

Theorem 2 establishes that all accepted pairs are uniformly distributed over the region $G_f$ between the density function $f$ and the $x$ axis. Under Theorem 1, one can then ensure that the $x$ coordinate of these accepted pairs follows a distribution with a density $f$. Furthermore, the following theorem shows the reciprocal idea to that of Theorem 1:

**Theorem 3.** *Let $X$ be a r.v. with a density $g(x)$ and $Y$ be a uniform r.v. in the interval $(0, cg(X))$. Then, for some constant $c > 0$, $(X, Y)$ is uniformly distributed in $G_{cf} = \{(x, y) : 0 < y \leq cf(x)\}$.*

For the proof, see [32]. This method uses a density function $f(x)$ and a dominant function $t(x) \geq f(x)$. The original description of the method used the maximum of the density as the dominant function $t(x) = \max_x f(x)$. For $\alpha, \beta > 1$, it is possible to take $t(x) = f\left(\frac{p-1}{p+q-2}\right)$. This function $t(x)$ is not really a density function since $c = \int_{-\infty}^{\infty} t(x)dx \geq \int_{-\infty}^{\infty} f(x)dx = 1$, but the function $g(x) = \frac{t(x)}{c}$ is. Therefore, a common method for generating samples of a $t(\cdot)$ function for a density $f(\cdot)$ is to choose another density function $g(\cdot)$ from which samples can be easily generated and find a constant $c > 0$ such that $cg(x) \geq f(x)$ for all $x$ in the domain of $f(\cdot)$. A constant $c$ can be found by taking $c = \max_{x>0}\left\{\frac{f(x)}{g(x)}\right\}$. By setting $t(x) = cg(x)$, it is possible to obtain Algorithm 3.

**Algorithm 3** Acceptance-rejection method for generating a $beta(\alpha, \beta)$ r.v.

**Require:** Density of a beta r.v. $f(x)$, constant $c$ and density $g(x)$ such that $cg(x) \geq f(x)$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
1: **repeat**
2: Generate $x$ with density $g(\cdot)$
3: Generate $u \sim U(0, 1)$
4: **until** $u \leq \frac{f(x)}{cg(x)}$
5: **return** $x$

The execution time of Algorithm 3 depends on (1) the time taken to generate $x$, (2) the time taken for the comparison $u \leq \frac{f(x)}{cg(x)}$ and (3) the expected number of iterations $c$ until the required number of values of $X$ is achieved. The loop condition requires constantly evaluating the function $f(x)$ and is not always easy, especially in the case of a beta r.v. By adding an intermediate step to Algorithm 3, it is possible to accelerate the process. If there is a simpler way to evaluate and minorise the function $s(x)$ for the density $f(x)$ (i.e., $s(x) \leq f(x)\ \forall x \in [0, 1]$), then it is not necessary to calculate the density function $f$ to accept the sample. If the pair $(x, ucg(x))$ lies below the curve $s(x)$, then $x$ is accepted directly. This modification is called the squeeze method [40], which is illustrated in Algorithm 4.

**Algorithm 4** The squeeze method

**Require:** Density $f(x)$ of a beta r.v., constant $c$ and density $g(x)$ such that $cg(x) \geq f(x)$, minorising function $s(x)$
**Output:** Random sample $x$ coming from $beta(\alpha, \beta)$ r.v.
1: **repeat**
2: Generate $x$ with density $g$
3: Generate $u \sim U(0, 1)$
4: **if** $u \leq \frac{s(x)}{cg(x)}$ **then go to** 6
5: **until** $u \leq \frac{f(x)}{cg(x)}$
6: **return** $x$

#### 3.3.1. Jöhnk's Algorithm

This method shows that, starting from two independent $U(0, 1)$ random variables $Y$ and $Z$, if $Y^{1/\alpha} + Z^{1/\beta} \leq 1$, then $X = Y^{1/\alpha}/(Y^{1/\alpha} + Z^{1/\beta}) \sim beta(\alpha, \beta)$ (see [41]). The method is valid for all beta distributions with parameters $\alpha > 0$ and $\beta > 0$, but it requires on average $\frac{1}{P(Y+Z\leq 1)} = \frac{\Gamma(\alpha+\beta+1)}{\Gamma(\alpha+1)\Gamma(\beta+1)}$ iterations (see [32]), and these grow rapidly with $\alpha$ and $\beta$; that is, when $\alpha$ and $\beta$ are large parameters, the computational cost is high, and thus it is recommended to restrict their use to $beta(\alpha, \beta)$ such that $\alpha, \beta < 1$. This algorithm is not used anymore because it is not as efficient as other methods. The pseudo-code is in Algorithm 5.

**Algorithm 5** Jöhnk's algorithm [41]

**Require:** Parameters $\alpha$ and $\beta$
**Output:** Random sample $x$ coming from $beta(\alpha, \beta)$ r.v.
1: **repeat**
2: Generate $u, v \sim U(0,1)$
3: $y \leftarrow u^{1/\alpha}$
4: $z \leftarrow v^{1/\beta}$
5: **until** $y + z \leq 1$
6: **return** $x = \frac{y}{y+z}$

3.3.2. Forsythe's Method

The basic method is described in [42], and some extensions and applications can be seen in [43,44]. It consists of two parts: a first step where an interval is selected to generate the $X$ r.v. and a second step where the exact value within the interval is determined by the A–R technique. This method allows the generation of random variables with a density $f(x) = Ce^{-B(x)}$, where $B(x)$ is an increasing function of $x$ in $(0, \infty)$ and $C$ is a real constant. The successive intervals of $x$ have limits $q_0, q_1, \ldots$, verifying $B(q_k) - B(q_{k-1}) \leq 1$. For each interval of $x$, two constants are calculated: $r_k = \int_0^{q_k} f(x)dx$ and $d_k = q_k - q_{k-1}$ $(k = 1, \ldots, K)$. The number of intervals $K$ is chosen so that $r_K$ exceeds the largest representative number less than one. In addition, to sample within the interval, it is appropriate to define the following function: $G_k(x) = B(q_{k-1} + x) - B(q_{k-1})$.

Algorithm 6 is divided into two parts: a first loop that allows us to choose the interval $[q_{k-1}, q_k)$ to which $x$ will belong and a second one that will determine the value of $x$ in that interval.

**Algorithm 6** Forsythe's algorithm [42]

**Require:** Functions $B(X), G(X)$, constants $q_k, r_k, d_k$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
SELECTION OF THE INTERVAL
1: $k \leftarrow 1$
2: Generate $u \sim U(0,1)$
3: **if** $u <= r_k$ **then go to** 5
4: **else** $k \leftarrow k + 1$ **go to** 3
GENERATION OF $x$
5: Generate $u \sim U(0,1)$
6: $w \leftarrow ud_k$
7: $t \leftarrow G_k(w)$
8: Generate $v \sim U(0,1)$
9: **if** $v \geq t$ **then**
10: **return** $x \leftarrow q_{k-1} + w$
11: **else**
12: Generate $u \sim U(0,1)$
13: **if** $u < v$ **then** $t \leftarrow u$ **go to** 8
14: **else go to** 5

For a beta r.v. with parameters $\alpha$ and $\beta$ it follows that

$$B(x) = -(\alpha - 1)\log(x) - (\beta - 1)\log(1 - x) \tag{2}$$

$$G_k(x) = -(\alpha - 1)\log\left(1 + \frac{x}{q_{k-1}}\right) - (\beta - 1)\log\left(1 - \frac{x}{1 - q_{k-1}}\right) \tag{3}$$

As for the number of intervals, in [45], it is proposed to control their width by successively solving $G_k(x) = g_{max}$, finding it satisfactory to take a value of 0.2 for $g_{max}$. The constants $r_k$ are obtained directly by integrating the density.

The main disadvantage of this method is that it requires continuous calculation of the constants $q_k, d_k$ and $r_k$, which reduces its efficiency.

In [45], a comparative analysis (in $\mu$s) of the performance of Jöhnk's and Fosythe's algorithms for the case of $\alpha = \beta < 1$ was performed by programming them in FORTRAN on two computers: a Cyber 73-14 and an IBM 360/65. In this case, Forsythe's algorithm turned out to be more efficient than Jöhnk's algorithm for $\alpha = \beta < 1$ analysed on the Cyber 73-14 and IBM 360/65, except for the case where $\alpha = 0.1$. As the value of $\alpha$ increased, the execution time in Forsythe's algorithm remained approximately the same, whereas in the case of Jöhnk's algorithm, it tended to be little more than a half as fast as Forsythe's. In this paper, the case $\alpha = \beta > 1$ is also analysed, and Forsythe's method is compared to other methods which will be described later.

#### 3.3.3. Ahrens and Dieter's Methods

These methods are particular cases of the A–R method based on the density of a normal r.v. The beta distribution has as its domain the interval $[0, 1]$, and when its parameters take values greater than one, its density function is bounded. This is why the density function of a beta distribution can be increased using a normal r.v. and not have any problem with tails. In [46], this idea was used to modify the A–R method so that samples from a beta distribution of the parameters $\alpha, \beta > 1$ could be generated by using a majorising function proportional to the density of a normal r.v. To perform this method, the beta density function is taken, without loss of generality, as $f(x) = \left(\frac{x}{A}\right)^A \left(\frac{1-x}{B}\right)^B C^C$, where $A = \alpha - 1, B = \beta - 1$ and $C = A + B = \alpha + \beta - 2$. As proven in [46], $f(x) \leq \exp\left(-2C\left(x - \frac{A}{C}\right)^2\right)$, $\forall x \in [0, 1]$. The left-hand side of the inequality corresponds to the density of a beta r.v., while the right-hand side is proportional to the density of a normal r.v. with a mean $\mu = \frac{A}{C}$ and standard deviation $\sigma = \frac{1}{(2C^{\frac{1}{2}})}$.

In Algorithm 7, the pseudo-code of this variant is presented.

**Algorithm 7** BN algorithm ([46], $\alpha, \beta > 1$)

**Require:** Parameters $\alpha > 1$ and $\beta > 1$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
1: $A \leftarrow \alpha - 1, B \leftarrow \beta - 1, C \leftarrow A + B, L \leftarrow C \log C, \mu \leftarrow \frac{A}{C}, \sigma \leftarrow \frac{0.5}{C^{\frac{1}{2}}}$
2: Generate $z \sim N(0, 1)$
3: $x \leftarrow z\sigma + \mu$
4: **if** $x < 0$ **or** $x > 1$ **then go to** 2
5: Generate $u \sim U(0, 1)$
6: **if** $\log u > A \log\left(\frac{x}{A}\right) + B \log\left(\frac{1-x}{B}\right) + L + 0.5z^2$ **then go to** 2
7: **else**
8: **return** $x$

Generating samples of a uniform r.v. and a normal r.v. plus three logarithmic evaluations are the least efficient computations of this algorithm. The average number of iterations is $\frac{(\frac{1}{2}\pi)^{\frac{1}{2}} A^A B^B \Gamma(\alpha+\beta)}{C^C \Gamma(\alpha)\Gamma(\beta)} \approx \frac{1}{2}\frac{(A+B)}{(AB)^{\frac{1}{2}}}$. This approximation is based on Stirling's formula [46] and is valid except for small values of $\alpha$ and $\beta$. This method reaches its maximum efficiency in the symmetric case $\alpha = \beta$. For this case, the version of Algorithm 7 developed thus far can be improved. If $\alpha = \beta$, then $A = B, C = 2A$ and step 13 tests whether $u \leq (4x(1-x))^A e^{\frac{z^2}{2}}$ is true. For $x = z\sigma + \mu$, this condition can be written as $u \leq q(z)$ with $q(z) = \left(1 - \frac{\frac{1}{2}z^2}{A}\right)^A e^{\frac{z^2}{2}}$. In addition, this function $q(\cdot)$ can be bounded to speed up the algorithm such that $\frac{1-z^4}{(8\alpha-12)} \leq q(z) \leq \frac{1-z^4}{(8\alpha-8)} + \frac{1}{2}\left(\frac{z^4}{(8\alpha-8)}\right)^2$. The pseudo-code can be seen in Algorithm 8.

In [46], the authors made a comparison in $\mu$s of the BN and BS algorithms programmed in FORTRAN and found that the computational time stabilised quickly for 150 µs in the symmetric case. In [45], a comparison of the performance (in $\mu$s) of the BN algorithm, general rejection method, Forsythe's method and a method based on order statistics was carried out in FORTRAN on two computers: a Cyber 73-14 and an IBM 360/65. For the case where $\alpha = 2$, the general rejection method was faster than the BN algorithm but slower than Forsythe's algorithm. For the rest of the $\alpha$ values analysed, it was slowest and inefficient in the Cyber 73-14 and one of the slowest in the IBM 360-65. For values of $\alpha > 3$, the BN algorithm provided the second fastest algorithm, with Forsythe's method being the fastest for all integers $\alpha$ except two.

**Algorithm 8** BS algorithm ([46], $\alpha = \beta > 1.5$)

**Require:** Parameter $\alpha > 1.5$
**Output:** Random sample $x$ with $beta(\alpha, \alpha)$ distribution
$A \leftarrow \alpha - 1$
$t \leftarrow (A + A)^{\frac{1}{2}}$
Generate $z \sim N(0,1)$
$x \leftarrow \frac{1}{2}(1 + \frac{z}{t})$
**if** $x < 0$ *or* $x > 1$ **then go to** 3
Generate $u \sim U(0,1)$
**if** $u \leq 1 - \frac{z^4}{(8\alpha - 12)}$ **then**
**return** $x$
**else if** $u \geq 1 - \frac{z^4}{(8\alpha - 8)} + \frac{1}{2}\left(\frac{z^4}{(8\alpha - 8)}\right)^2$ **then go to** 3
**else if** $\log u > A \log(4x(1-x)) + \frac{z^2}{2}$ **then go to** 3
**else**
**return** $x$

### 3.3.4. Switching Algorithms

In [47], a new procedure was developed to generate beta random variables with both parameters or at least one of them being less than one for densities of the form

$$f(x) = c f_1(x) f_2(x) \tag{4}$$

However, it also combines the composition and inversion methods.

Let $g(x) = k_1 f_1(x)$ be a density function with an associated distribution function $G$. If $X$ is a r.v. with density $g(x)$ and $U \sim U(0,1)$, then we can deduce using the A–R method that $X$ is accepted if $U \sup f_2 \leq f_2(X)$. The average number of iterations required is $\frac{c \sup f_2}{k_1}$. This method is particularly useful for unbounded densities. In [47], two cases were developed: Case 1, with both parameters being less than one, and Case 2, with one parameter being less than one and the other being higher than one.

In Case 1, the density is not bounded when $x = 0$ or $x = 1$. To solve the difficulties this implies in the rejection algorithm, a composition is employed so that the range $[0,1]$ to which $x$ belongs is divided into two: $[0,t]$ and $(t,1]$. When $x$ takes values greater than $t$, the roles of $f_1$ and $f_2$ of the density function in Equation (4) are switched. For this reason, this algorithm is named the *"switching algorithm"*. Starting from the density of Equation (4), then following is taken:

$$\begin{aligned} f_1(x) &= \alpha x^{\alpha - 1} \\ f_2(x) &= \beta (1-x)^{\beta - 1} \end{aligned} \tag{5}$$

To generate a sample $x$ from a beta r.v., we have that $x \in [0,t]$ with a probability $p$, while $x \in (t,1]$ with a probability $1 - p$. If $0 \leq x \leq t$, then $x$ is generated from the density $\{{}^{\alpha x^{\alpha-1}}/_{t\alpha},\ 0 \leq x \leq t\}$ through the inversion method, while if $t < x \leq 1$, then $x$ is generated from $\{{}^{\beta(1-z)^{\beta-1}}/_{(1-t)^{\beta}},\ t < z \leq 1\}$. From the calculations described in detail in [47], the probability value $p$ and the optimal value of $t$ are obtained as a function of the parameters of the distribution $beta(\alpha, \beta)$ to be generated:

$$t = \frac{\sqrt{\alpha(1-\alpha)}}{\sqrt{\beta(1-\beta)} + \sqrt{\alpha(1-\alpha)}} \tag{6}$$

$$p = \frac{\beta t}{\alpha(1-t) + \beta t} \tag{7}$$

At first, this new algorithm requires generating an exponential r.v. with a parameter 1, $E \sim exp(1)$ and two uniform random variables in $[0,1]$. The first uniform r.v. $U$ is used in the $U < p$ test that dictates whether to generate $x \in [0,t]$ or $x \in (t,1]$, while the second uniform $V$ is needed to generate $x$ itself through the inversion method. However, with the properties of the uniform variable, if $U < p$, then ${}^{U}/_{p}$ also follows a uniform distribution. Therefore, it is not necessary to generate $V$, and it is sufficient to generate $x$ from ${}^{U}/_{p}$ or ${}^{(1-U)}/_{(1-p)}$, depending on whether the test $U < p$ is verified or not. In Algorithm 9, a pseudo-code for this method is described.

Algorithm 9 has an acceptance rate $1/e$, with

$$e = \frac{\Gamma(\alpha+1)\Gamma(\beta+1)}{\Gamma(\alpha+\beta)} \frac{t^{(1-\alpha)}(1-t)^{(1-\beta)}}{\beta t + \alpha(1-t)}$$

being the efficiency [47].

Case 2 does not differ much from the previous one as far as the procedure is concerned. Here, $f_1$ and $f_2$ are taken as in Equation (5), and $x$ is generated from the same densities. However, the calculation of $p$ and the optimal value of $t$ varies. Now, $p = \frac{\beta t}{\beta t + \alpha(1-t)^\beta}$ and $t$ must satisfy $h(t) = \beta t + (\alpha-1)(1-t)^\beta - \beta t(1-t)^{(\beta-1)} = 0$. Except for particular cases, this equation cannot be solved analytically, but it is easily solved numerically. By solving it in this way, Atkinson and Whittaker [47] concluded that the value closest to the optimum one is obtained when $t = \frac{(1-\alpha)}{(\beta+1-\alpha)}$.

**Algorithm 9** Switching algorithm [47,48] (SW2 $(\alpha, \beta < 1)$)

**Require:** Parameters $\alpha < 1$ and $\beta < 1$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$

$t \leftarrow \frac{\sqrt{\alpha(1-\alpha)}}{\sqrt{\beta(1-\beta)}+\sqrt{\alpha(1-\alpha)}}$
$p \leftarrow \frac{\beta t}{\alpha(1-t)+\beta t}$
Generate $U \sim U(0,1)$
Generate $E \sim exp(1)$
**if** $U > p$ **then**
    $X = 1 - (1-t)\left(\frac{1-U}{1-p}\right)^{1/\beta}$
    **if** $(1-\alpha)\log\left(\frac{X}{t}\right) \leq E$ **then**
        **return** $X$
    **else go to** 3
**else**
    $X = t\left(\frac{U}{p}\right)^{1/\alpha}$
    **if** $(1-\beta)\log\left(\frac{1-X}{1-t}\right) \leq E$ **then**
        **return** $X$
    **else go to** 3

By applying this modification in Algorithm 9, Algorithm 10 is obtained. The efficiency of Algorithm 10 is

$$e = \frac{\Gamma(\alpha+1)\Gamma(\beta+1)}{\Gamma(\alpha+\beta)} \frac{1}{\beta t^\alpha + \alpha(1-t)^\beta t^{\alpha-1}}$$

See [47] for more information.

**Algorithm 10** Switching algorithm [47,48] (SW1 $(\alpha < 1, \beta > 1)$)

**Require:** Parameters $\alpha < 1$ and $\beta > 1$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$

$t \leftarrow \frac{(1-\alpha)}{(\beta+1-\alpha)}$
$p \leftarrow \frac{\beta t}{\alpha(1-t)+\beta t}$
Generate $U \sim U(0,1)$
Generate $E \sim exp(1)$
**if** $U > p$ **then**
    $X = 1 - (1-t)\left(\frac{1-U}{1-p}\right)^{1/\beta}$
    **if** $(1-\alpha)\log\left(\frac{X}{t}\right) \leq E$ **then**
        **return** $X$
    **else go to** 3
**else**
    $X = t\left(\frac{U}{p}\right)^{1/\alpha}$
    **if** $(1-\beta)\log(1-X) \leq E$ **then**
        **return** $X$
    **else go to** 3

A comparative analysis of the performance (in $\mu$s) of the SW1 and SW2 algorithms by programming them in FORTRAN on a Cyber 73-14 is presented in [47]. In particular, in Case 1, the Jöhnk's and switch algorithms were compared. If $\alpha + \beta < 1$, then Jöhnk's algorithm is preferable, while if $\alpha + \beta > 1$, then the switching algorithm is faster. In Case 2, Whittaker suggested an A–R algorithm without decomposition, and its pseudo-code can be seen in Algorithm 11. This method was compared to the switch algorithm, resulting in the last one being faster in all studied parameter combinations.

**Algorithm 11** Whittaker's algorithm [47] ($\alpha < 1, \beta > 1$)

**Require:** Parameters $\alpha < 1$ and $\beta > 1$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
1: Generate $U \sim U(0,1)$ and $E \sim exp(1)$
2: Set $X = U^{1/\alpha}$
3: **if** $(1-\beta)\log(1-X) \leq E$ **then** accept $X$. Otherwise **go to** 1

### 3.3.5. Cheng's Methods

The method presented in [49] is based on the A–R method by applying it to second-type beta variables $beta_2(\alpha, \beta)$. (The density of a $beta_2(\alpha, \beta)$ r.v. is $f(X) = \frac{x^{\alpha-1}}{B(\alpha,\beta)(1+x)^{\alpha+\beta}}, x > 0$, where $B(\alpha, \beta)$ is the beta function). If $Y \sim beta_2(\alpha, \beta)$, then $Z = \frac{Y}{1+Y} \sim beta(\alpha, \beta)$ [49].

Based on Algorithm 3, for generating values of a $B_2(\alpha, \beta)$ r.v., we must select $f(x)$ as a density of a $beta_2(\alpha, \beta)$ r.v. and $g(x) = \lambda\mu x^{\lambda-1}(\mu + x^\lambda)^{-2}$, where $\mu$ and $\lambda$ are parameters set to obtain a small value for $c$. Cheng suggested [49] $\mu = (\frac{\alpha}{\beta})^\lambda$ and

$$\lambda = \begin{cases} \min(\alpha, \beta) & , if\ \min(\alpha, \beta) \leq 1 \\ \sqrt{\frac{2\alpha\beta - (\alpha+\beta)}{\alpha+\beta-2}} & , if\ \min(\alpha, \beta) > 1 \end{cases} \tag{8}$$

Additionally, $c = \max(f/g)$ is taken as a function of the given parameters $c = \frac{4\alpha^\alpha \beta^\beta}{\lambda B(\alpha,\beta)(\alpha+\beta)^{\alpha+\beta}}$. In Algorithm 12 the pseudo-code of this method is presented.

**Algorithm 12** BA algorithm [49]

**Require:** Parameters $\alpha$ and $\beta$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
INITIALISATION
1: $a \leftarrow \alpha + \beta$
2: **if** $\min(\alpha, \beta) \leq 1$ **then** $b \leftarrow \max(\alpha^{-1}, \beta^{-1})$
3: **else** $b \leftarrow \sqrt{\frac{a-2}{2\alpha\beta - a}}$
4: $g \leftarrow \alpha + b^{-1}$
GENERATION
5: Generate $u_1, u_2 \sim U(0,1)$
6: $v \leftarrow b\log\left(\frac{u_1}{(1-u_1)}\right)$
7: $w \leftarrow \alpha e^v$
8: **if** $a\log\left(\frac{a}{(\beta+w)}\right) + gv - 1.3862944 < \log(u_1^2 u_2)$ **then go to** 5 ▷ Constant is log(4)
9: **return** $x = \frac{w}{(\beta+w)}$

The time required for this algorithm depends mainly on the number of iterations $c$ needed to obtain a sample. It is interesting to distinguish the behaviour of this quantity in two different cases: (1) when $\min(\alpha, \beta) > 1$, $c$ does not rise approximately above 1.47, and (2) when $\min(\alpha, \beta) \leq 1$, $c$ reaches 4. This duality behaviour suggests modifying Algorithm 12 with the goal of increasing its speed for each of the cases. In this way, Cheng [49] developed (1) Algorithm BB for $\min(\alpha, \beta) > 1$ and (2) Algorithm BC for $\min(\alpha, \beta) \leq 1$.

Algorithm 12 requires four logarithmic evaluations to be performed between steps 6 and 8. This can be computationally demanding. When the rejection rate is low ($\min(\alpha, \beta) > 1$), the logarithmic evaluation of step 6 cannot be avoided. However, a preliminary test can be performed to avoid the evaluations of step 8. The idea behind this test resides in the property that a state as long as $\log z$ is a

concave function, and its tangent always lies above the curve [49]. Therefore, $\theta z - \log(\theta) - 1 \geq z$, $\forall z, \theta > 0$. Let $z = \beta + W$. If the test

$$m \log(m) - \log 4 + aV - m\{\theta(\beta + W) - \log(\theta) - 1\} \geq \log(U_1^2 U_2) \tag{9}$$

is satisfied, then the value $X = \frac{W}{\beta+W}$ will be accepted.

Regarding the choice of $\theta$, in [49], it was concluded that taking $\theta = 1/(\alpha + \beta)$ is the best option to maximise the conditional probability of satisfying the test in Equation (9) given that the algorithm is accepted. This value for $\theta$ causes a drawback: it must be ensured that $\alpha < \beta$. However, since if $X \sim beta(\alpha, \beta)$, then $1 - X \sim beta(\beta, \alpha)$ [37], it is possible to take without loss of generality $\alpha$ and $\beta$ as the minimum and maximum, respectively, of the original parameters $\alpha_0$ and $\beta_0$, and if an exchange occurs between the original parameters and those taken for the algorithm, then $X = \frac{b}{(b+W)}$ is returned instead of $X = \frac{W}{(b+W)}$ [49].

It is also possible to avoid evaluating $\log(U_1^2 U_2)$ in the preliminary test (Equation (9)) as discussed in [49]. The exact value of $\theta$ is not critical for this test, and $\theta = 5$ can be taken independently of the values of $\alpha$ and $\beta$. These modifications are given in Algorithm 13.

For the case where $\min(\alpha, \beta) \leq 1$, rejects must be detected beforehand by performing preliminary tests between steps 6 and 8 of Algorithm 12. For this purpose, the following results are used [49]:

**Lemma 1.** *If* $U_1 < \frac{1}{2}$ *and* $\alpha \geq \beta$, *then* $R(x) = \frac{f(x)}{Cg(x)}$ *verifies* $R(U_1) \leq k_1(1 - 2U_1)^{-2}$, *where* $k_1 = \frac{\delta(1+3\beta)}{4(18(\frac{\alpha}{\beta})-14)}$.

**Lemma 2.** *If* $U_1 \geq \frac{1}{2}$ *and* $min(\alpha, \beta) = \beta \leq 1$, *then* $U_1^{-2} \leq 4R(U_1) \leq k_2 U_1^{-2}$, *where* $k_2 = \frac{1+(2+\delta^{-1})\beta}{4}$ *with* $\delta = 1 + \alpha - \beta$.

**Algorithm 13** BB algorithm [49] ($\min(\alpha_0, \beta_0) > 1$)

**Require:** Parameters $\alpha_0$ and $\beta_0$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
INITIALISATION
$\alpha \leftarrow \min(\alpha_0, \beta_0)$
$\beta \leftarrow \max(\alpha_0, \beta_0)$
$a \leftarrow \alpha + \beta$
$b \leftarrow \sqrt{\frac{a-2}{2\alpha\beta - a}}$
$g \leftarrow \alpha + b^{-1}$
GENERATION
Generate $u_1, u_2 \sim U(0, 1)$
$v \leftarrow b \log\left(\frac{u_1}{1-u_1}\right)$
$w \leftarrow \alpha e^v$
$z \leftarrow u_1^2 u_2$
$r \leftarrow gv - 1.3862944$ ▷ Constant is $\log 4$
$s \leftarrow \alpha + r - w$
**if** $s + 2.609438 \geq 5z$ **then go to** 16 ▷ Constant is $1 + \log 5$
$t \leftarrow \log z$
**if** $s \geq t$ **then go to** 16
**if** $r + a \log\left(\frac{a}{\beta+w}\right) < t$ **then go to** 6
**if** $\alpha = \alpha_0$ **then**
    **return** $x = \frac{w}{b+w}$
**else**
    **return** $x = \frac{b}{b+w}$

The choice of $k_1$ and $k_2$ is covered in detail in [49].

Lemma 1 allows one to create a preliminary test for rejection, while Lemma 2 can be applied to create preliminary tests for both acceptance and rejection. Algorithm 14 includes these tests.

In [49], a comparison of the methods proposed in different situations was made through programming in FORTRAN and running on a CDC 7600 computer. The case where $\min(\alpha, \beta) > 1$

was first analysed, and the BA, BB and BN algorithms were compared, leaving aside Jöhnk's algorithm since it was substantially slower than the other methods in this case. In this situation, the BB algorithm was uniformly faster than the BA algorithm, which was uniformly faster than the BN algorithm. While Cheng argued that the speed of the BN algorithm could be improved by applying a better normal r.v. generator, he in fact said that the BN algorithm's times would be reduced to reach the performance of the BA or BB algorithms when $\alpha$ and $\beta$ were approximately equal and large. However, Cheng pointed out two aspects showing why the BA and BB algorithms were better than the BN algorithm. First, the performance of the BA and BB remained approximately constant for any values of $\alpha$ and $\beta$, while the performance of the BN algorithm worsened when $\alpha$ and $\beta$ were close to one or when their values were quite different. Second, in the case where $\alpha$ and $\beta$ varied and had to be recalculated at each call, the BA and BB algorithms took only 5.6 µs more time per variable, while the BN algorithm required an additional 9.9 µs. The case where $\min(\alpha, \beta) \leq 1$ was also analysed. Here, the BA, BC, switch (1 and 2) and Jöhnk's algorithms were compared. None of them substantially dominated over the others. If $\alpha$ and $\beta$ were fixed and less than or equal to one, and if $\alpha + \beta \leq 1$, then Jöhnk's method would be the best performer, while if $\alpha + \beta > 1$, then the BC or first switch method were dominant. If $\alpha$ and $\beta$ were fixed, and either of them were greater than one, then the second switch method was slightly faster than the BC algorithm if the other parameter was small, while the BC algorithm was slightly faster in the other cases. Here, Jöhnk's method slowed down rapidly as $\alpha$ or $\beta$ increased. In the case where $\alpha$ and $\beta$ were not fixed and were updated on each call, the Switch methods were not as efficient, and it was Jöhnk's method that provided better performance when $\alpha$ or $\beta$ was small or if $\alpha + \beta \leq 1.5$. Outside of this region, the BA algorithm was better if $\alpha, \beta \approx 1$; otherwise, the BC algorithm was better.

**Algorithm 14** BC algorithm [49] ($\min(\alpha_0, \beta_0) \leq 1$)

**Require:** Parameters $\alpha_0$ and $\beta_0$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
INITIALISATION
$\alpha \leftarrow \max(\alpha_0, \beta_0)$
$\beta \leftarrow \min(\alpha_0, \beta_0)$
$a \leftarrow \alpha + \beta$
$b \leftarrow \beta^{-1}$
$\delta \leftarrow 1 + \alpha - \beta$
$k_1 \leftarrow \frac{\delta(0.25+0.75\beta)}{18\alpha b - 14}$
$k_2 = 0.25 + (0.5 + 0.25\delta^{-1})\beta$
GENERATION
Generate $u_1, u_2 \sim U(0,1)$
**if** $u_1 \geq \frac{1}{2}$ **then go to** 14
$y \leftarrow u_1 u_2$
$z \leftarrow u_1 y$
**if** $0.25u_2 + z - y \geq k_1$ **then go to** 8
**else go to** 20
$z \leftarrow u_1^2 u_2$
**if** $z \leq 0.25$ **then**
    $v \leftarrow b \log\left(\frac{u_1}{1-u_1}\right)$
    $w \leftarrow \alpha e^v$
    **go to** 23
**if** $z \geq k_2$ **then go to** 8
$v \leftarrow b \log\left(\frac{u_1}{1-u_1}\right)$
$w \leftarrow \alpha e^v$
**if** $a\left[\log\left(\frac{a}{\beta+w}\right) + v\right] - 1.3862944 < z$ **then go to** 8
**if** $\alpha = \alpha_0$ **then**
    **return** $x = \frac{w}{\beta+w}$
**else**
    **return** $x = \frac{\beta}{\beta+w}$

#### 3.3.6. BNM Algorithm

In [50], another technique for generating beta distributions was developed based on the one already developed in [46]. This new technique assumes that the inflection points of $f(x)$ fall at points of the form

$$x = \frac{\left[A \pm \left(\frac{AB}{C-1}\right)^{\frac{1}{2}}\right]}{C} \tag{10}$$

if these values are real numbers between 0 and 1. If there are no inflection points, then the density is concave. If there are two inflection points, then the density is concave between these points and convex in the rest of the space. If there is only one point, then the density is concave in the direction of the mode and convex in the opposite direction. The mode is found in $\frac{A}{C}$ when $A, B > 0$. For this method, it is essential to define the following points (see [50]):

$$\begin{aligned}
x_1 &= x_2 - \frac{x_2(1-x_2)}{A - Cx_2} \\
x_2 &= \begin{cases} \frac{\left[A-\left(\frac{AB}{C-1}\right)^{\frac{1}{2}}\right]}{C} & \text{, if it is a real number in [0,1]} \\ 0 & \text{, in other case.} \end{cases} \\
x_3 &= \frac{A}{C} \\
x_4 &= \begin{cases} \frac{\left[A+\left(\frac{AB}{C-1}\right)^{\frac{1}{2}}\right]}{C} & \text{, if it is a real number in [0,1]} \\ 0 & \text{, in other case.} \end{cases} \\
x_5 &= x_4 - \frac{(x_4(1-x_4))}{A - Cx_4}
\end{aligned} \tag{11}$$

where $x_2$ and $x_4$ are the inflection points in the case where they exist, $x_3$ is the mode and $x_1$ and $x_5$ are the points at which tangents passing through points $x_2$ and $x_4$ intersect the $X$ axis, respectively. In addition, this method performs a preliminary test with a minorant function $b_1(x) \leq f(X)\ \forall x \in [0,1]$ defined by

$$b_1(x) = \begin{cases} 0 & \text{, if } 0 \leq x \leq x_1 \\ \frac{(x-x_1)}{(x_3-x_1)} & \text{, if } x_1 < x \leq x_3 \\ \frac{(x_5-x)}{(x_5-x_3)} & \text{, if } x_3 < x \leq x_5 \\ 0 & \text{, if } x_5 < x \leq 1 \end{cases} \tag{12}$$

For more details, see [50]. To write Algorithm 15, it is enough to define the points (Equation (11)) in Algorithm 7 and add a step between steps 5 and 6 that allows the following test to be performed:

$$\text{if } u \exp\left(\frac{-z^2}{2}\right) \leq b_1(x), \textbf{ Return } x.$$

Although Algorithm 15 is more extensive and includes numerous conditionals, the existence of the preliminary test reduces the number of logarithmic evaluations required, which should make it more computationally efficient.

**Algorithm 15** BNM [50]

**Require:** Parameters $\alpha > 1$ and $\beta > 1$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
$A \leftarrow \alpha - 1$
$B \leftarrow \beta - 1$
$C \leftarrow A + B$
$L \leftarrow C \log C$
$\mu \leftarrow {}^{A}/_{C}$
$\sigma \leftarrow {}^{0.5}/_{C^{\frac{1}{2}}}$
$x_2 \leftarrow \frac{\left\lceil A-\left(\frac{AB}{C-1}\right)^{\frac{1}{2}}\right\rceil}{R}$
**if** $x_2$ **not in** $[0,1]$ **or not** real **then** $x_2 = 0$
$x_4 \leftarrow \frac{\left\lceil A+\left(\frac{AB}{C-1}\right)^{\frac{1}{2}}\right\rceil}{R}$
**if** $x_4$ **not in** $[0,1]$ **or not** real **then** $x_4 = 1$
$x_1 = x_2 - \frac{x_2(1-x_2)}{A-Cx_2}$
$x_5 = x_4 - \frac{x_4(1-x_4)}{A-Cx_4}$
$x_3 = \mu$
Generate $z \sim N(0,1)$
$x \leftarrow z\sigma + \mu$
**if** $x < 0$ **or** $x > 1$ **then go to** 14
Generate $u \sim U(0,1)$
**if** $0 \le x \le x_1$ **or** $x_5 < x \le 1$ **then** $s \leftarrow 0$
**else if** $x_1 < x \le x_3$ **then** $s \leftarrow \frac{x-x_1}{x_3-x_1}$
**else if** $x_3 < x \le x_5$ **then** $s \leftarrow \frac{x_5-x}{x_5-x_3}$
**if** $ue^{-z^2/2} \le s$ **then**
    **return** $x$
**if** $\log u > A\log\left(\frac{x}{A}\right) + B\log\left(\frac{1-x}{B}\right) + L + 0.5z^2$ **then go to** 14
**else**
    **return** $x$

### 3.3.7. B2P and B4P Algorithms

Similar to the A–R methods discussed in Section 3.3.3, by taking the density function of a beta r.v. with the parameters $\alpha, \beta > 1$ such that

$$f(x) = \left(\frac{x}{A}\right)^{A}\left(\frac{1-x}{B}\right)^{B} C^{C} \tag{13}$$

where $A = \alpha - 1, B = \beta - 1$ and $C = A + B$, Schmeiser and Shalaby [50] developed three methods based on the A–R method for generating samples of a $beta(\alpha, \beta)$ r.v. The first one is the BNM algorithm, and the other two methods will be described below. Starting from the points defined in Equation (11), this method uses a majorising piecewise function $t_1(x)$:

$$t_1(x) = \begin{cases} \frac{xf(x_2)}{x_2} & , if\, 0 \le x \le x_2 \\ 1 & , if\, x_2 < x \le x_4 \\ \frac{(1-x)f(x_4)}{1-x_4} & , if\, x_4 < x \le 1 \end{cases} \tag{14}$$

This method is called the 2-points technique, since $t_1$ requires evaluating $f(x)$ at two points: $x_2$ and $x_4$. It can easily be proven that $t_1(x) \ge f(x)\ \forall x \in [0,1]$ [50]. For $x \in [0, x_2]$ or $x \in [x_4, 1]$, $t_1$ is a straight line joining two points of the function $f(x)$, and this itself is convex, while for

$x \in [x_2, x_4]$, $t_1(x) = 1 = f(x_3) = \max_x f(x)$, and hence $t_1(x) \geq f(x)$. This method also employs a minorising function:

$$b_2(x) = \begin{cases} 0 & ,if\, 0 \leq x \leq x_1 \\ \frac{(x-x_1)f(x_2)}{x_2-x_1} & ,if\, x_1 < x \leq x_2 \\ f(x_2) + \frac{(x-x_2)(1-f(x_2))}{x_3-x_2} & ,if\, x_2 < x \leq x_3 \\ f(x_4) + \frac{(x_4-x)(1-f(x_4))}{x_4-x_3} & ,if\, x_3 < x \leq x_4 \\ \frac{(x_5-x)f(x_4)}{x_5-x_4} & ,if\, x_4 < x \leq x_5 \\ 0 & ,if\, x_5 < x \leq 1 \end{cases} \tag{15}$$

It can also be proven that $b_2(x) \leq f(x)\ \forall x \in [0,1]$ (see [50]). For $x \in [x_2, x_4]$, $f(x)$ is concave, and $b_2(x)$ represents two straight lines connecting $x_2$ ,$x_3$ and $x_4$, and thus $b_2(x) \leq f(x)$. For $x \in [x_1, x_2]$ or $x \in [x_4, x_5]$, $f(x)$ is concave, and $b_2(x) \leq f(x)$ because $b_2(x)$ is tangent to $f(x)$ in $x_2$ and $x_4$. By using $t_1, s$ it is possible to develop Algorithm 16.

For the 4-points method, it is necessary to evaluate $f(x)$ in $x_1$, $x_2$, $x_4$ and $x_5$ and consider the majorising function:

$$t_2(x) = \begin{cases} \frac{xf(x_1)}{x_1} & ,if\, 0 \leq x \leq x_1 \\ f(x_1) + \frac{(x-x_1)(f(x_2)-f(x_1))}{x_2-x_1} & ,if\, x_1 < x \leq x_2 \\ 1 & ,if\, x_2 < x \leq x_4 \\ f(x_5) + \frac{(x_5-x)(f(x_4)-f(x_5))}{x_5-x_4} & ,if\, x_4 < x \leq x_5 \\ \frac{(1-x)f(x_5)}{1-x_5} & ,if\, x_5 < x \leq 1 \end{cases} \tag{16}$$

The implementation of the 4-points method differs from the 2-points method in that four rectangular regions with a probability of rejection of zero have been created, while the rest is the same. These methods are efficient when both $\alpha$ and $\beta$ are relatively small. However, when either parameter is large, the majorising functions $t_1(x)$ and $t_2(x)$ do not fit particularly well, causing the B2P and B4P algorithms to be less efficient. In [21], two algorithms were developed by replacing the majorising function in the tails with a better-fitting exponential dominant function.

In [50] a performance comparison of the BNM, B2P, B4P, BN, BB, Jöhnk's and ratio of gammas (RG) algorithms was performed on a CYBER 72 computer using a FORTRAN compiler. The study concluded that the BNM algorithm is more efficient than the BN algorithm (in a marginal runtime), but for most parameter values, the B2P and B4P algorithms are faster, with the B4P algorithm being less sensitive to large parameter values. For its part, the BB method was faster when the distribution was quite skewed. Jöhnk's method was more inefficient than the others, and the RG algorithm was slow because it requires the generation of two gamma random variables.

**Algorithm 16** B2P algorithm [50]

**Require:** Parameters $\alpha, \beta > 1$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
INITIALISATION
$A \leftarrow \alpha - 1, B \leftarrow \beta - 1, C \leftarrow A + B, L \leftarrow C \log C, x_1 \leftarrow x_2 \leftarrow 0, x_3 \leftarrow \frac{A}{C}, x_4 \leftarrow x_5 \leftarrow 1, f_2 \leftarrow f_4 \leftarrow 0$
**if** $C \leq 1$ **then go to** 10
$D \leftarrow \frac{(AB/(C-1))^{1/2}}{C}$
**if** $D \geq x_3$ **then go to** 7
**else**
    $x_2 \leftarrow x_3 - D, x_1 \leftarrow x_2 - \frac{x_2(1-x_2)}{(A-Cx_2)}, f_2 \leftarrow \exp\left(A \log\left(x_2/A\right) + B \log\left((1-x_2)/B\right) + L\right)$
**if** $x_3 + D \geq 1$ **then go to** 10
**else**
    $x_4 \leftarrow x_3 + D, x_5 \leftarrow x_4 - \frac{x_4(1-x_4)}{(A-Cx_4)}, f_4 \leftarrow \exp\left(A \log\left(x_4/A\right) + B \log\left((1-x_4)/B\right) + L\right)$
$p_1 \leftarrow x_3 - x_2$ , $p_2 \leftarrow (x_4 - x_3) + p_1, p_3 \leftarrow f_2 x_2/2 + p_2, p_4 \leftarrow f_4(1-x_4)/2 + p_3$
GENERATION
Generate $u \sim U(0,1)$
$u \leftarrow u p_4$
Generate $w \sim U(0,1)$
**if** $u > p_1$ **then go to** 19
**else**
    $x \leftarrow x_2 + w(x_3 - x_2), v \leftarrow \frac{u}{p_1}$
    **if** $v \leq f_2 + w(1 - f_2)$ **then go to** 41
    **else go to** 37
**if** $u > p_2$ **then go to** 24
**else**
    $x \leftarrow x_3 + w(x_4 - x_3), v \leftarrow \frac{u-p_1}{(p_2-p_1)}$
    **if** $v \leq 1 - \frac{(1-f_4)}{w}$ **then go to** 41
    **else go to** 37
Generate $w_2 \sim U(0,1)$
**if** $w_2 > w$ **then** $w \leftarrow w_2$
**if** $u > p_3$ **then go to** 33
**else**
    $x \leftarrow w x_2$
    $v \leftarrow \frac{u-p_2}{(p_3-p_2)} w f_2$
    **if** $x \leq x_1$ **then go to** 37
    **else if** $v \leq \frac{f_2(x-x_1)}{(x_2-x_1)}$ **then** go to 41
    **else go to** 37
$x \leftarrow 1 - w(1 - x_4)$
$v \leftarrow \left(\frac{u-p_3}{(p_4-p_3)}\right)\left(\frac{(1-x)f_4}{(1-x_4)}\right)$
**if** $x \geq x_5$ **then go to** 37
**else if** $v \leq \frac{f_4(x_5-x)}{(x_5-x_4)}$ **then go to** 41
$P \leftarrow \log(v)$
**if** $P \geq -(x - x_3)^2(C + C)$ **then go to** 11
**if** $P > A \log(x/A) + B \log((1-x)/B) + L$ **then go to** 11
**else**
    **return** $x$

### 3.3.8. B2PE and B4PE Algorithms

In [21,51], two extensions of the B2P and B4P algorithms were proposed—the B2PE and B4PE algorithms—that are valid for $\alpha, \beta > 1$. Both algorithms are similar, with B4PE being more cumbersome but faster.

First, the new functions $t_1(x)$ and $t_2(x)$ are defined by taking the curve of an exponential for the tails [52], and $b_2(x)$ is taken as shown in Equation (15). The generation of each variable $X$ needed for the B2PE algorithm consists of taking a probability $p_j$, $j = 1, 2, 3$ with a region from the three existing ones defined, generating a point $(x, v)$ uniformly distributed over the selected region and accepting or rejecting it, depending on whether it is above or below the $t_1(x)$ and $b_2(x)$ [21] functions. To generate the points $(x, v)$ in the region $i$, we take

$$x = \begin{cases} x_2 + (x_4 - x_2) * v & , if\ i = 1 \\ x_2 + \log(u)/\lambda_2 & , if\ i = 2 \\ x_4 - \log(u)/\lambda_4 & , if\ i = 3 \end{cases} \quad (17)$$

where $\lambda_2 = {}^{A}/_{x_2} - {}^{B}/_{(1-x_2)}$, $\lambda_4 = {}^{B}/_{(1-x_4)} - {}^{A}/_{x_4}$ and $u \sim U(0,1)$. By computing $-\log(u)$, exponential values are generated for regions 2 and 3 [21]. For the $v$ coordinate, this is generated as $U(0, t_1(x))$, but it is not necessary to evaluate $t_1(x)$ explicitly. For $i = 1$, $t_1(x) = 1$, while for regions 2 and 3, $t_1(x)$ involves an exponential function, but by using Schmeiser's method [52], the exponential operation can be eliminated. For the B4PE algorithm, $t_2(x)$ is taken. The number of regions increases to 10, but the underlying idea is the same. The pseudo-codes of these methods are shown in Algorithms 17 and 18.

**Algorithm 17** B2PE algorithm (Schmeiser and Babu [21])

**Require:** Parameters $\alpha, \beta > 1$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
INITIALISATION
$A \leftarrow \alpha - 1$, $B \leftarrow \beta - 1$, $C \leftarrow A + B$, $L \leftarrow C \log C$, $x_2 \leftarrow f_2 \leftarrow f_4 \leftarrow 0$, $x_3 \leftarrow \frac{A}{C}$, $x_4 \leftarrow 1$
**if** $C \leq 1$ **then go to** 10
$D \leftarrow \frac{(AB/(C-1))^{1/2}}{C}$
**if** $D \geq x_3$ **then go to** 7
**else**
    $x_2 \leftarrow x_3 - D$, $\lambda_2 \leftarrow {}^{A}/_{x_2} - {}^{B}/_{(1-x_2)}$, $f_2 \leftarrow \exp\left(A \log\left({}^{x_2}/_{A}\right) + B \log\left({}^{(1-x_2)}/_{B}\right) + L\right)$
**if** $x_3 + D \geq 1$ **then go to** 10
**else**
    $x_4 \leftarrow x_3 + D$, $\lambda_4 \leftarrow {}^{B}/_{(1-x_4)} - {}^{A}/_{x_4}$, $f_4 \leftarrow \exp\left(A \log\left({}^{x_4}/_{A}\right) + B \log\left({}^{(1-x_4)}/_{B}\right) + L\right)$
$p_1 \leftarrow x_4 - x_2$
$p_2 \leftarrow {}^{f_2}/_{\lambda_2} + p_1$
$p_3 \leftarrow {}^{f_4}/_{\lambda_4} + p_2$
GENERATION
Generate $u \sim U(0,1)$
$u \leftarrow u p_3$
Generate $v \sim U(0,1)$
**if** $u > p_1$ **then go to** 22
**else** ▷ Region 1
    $x \leftarrow x_2 + u$
    **if** $x < x_3$ **and** $v < f_2 + \frac{(x-x_2)(1-f_2)}{(x_3-x_2)}$ **then go to** 40
    **else if** $x \geq x_3$ **and** $v < f_4 + \frac{(x_4-x)(1-f_4)}{(x_4-x_3)}$ **then go to** 40
    **else go to** 36
**if** $u > p_2$ **then go to** 30
**else** ▷ Region 2
    $u \leftarrow \frac{(u-p_1)}{(p_2-p_1)}$
    $x \leftarrow x_2 + \frac{\log(u)}{\lambda_2}$
    **if** $v < \frac{\lambda_2(x-x_2)+1}{u}$ **then go to** 40
    **if** $x \leq 0$ **then go to** 13
    **else**
        $v \leftarrow v f_2 u$ **then go to** 36
$u \leftarrow \frac{(u-p_2)}{(p_3-p_2)}$ ▷ Region 3
$x \leftarrow x_4 - \frac{\log(u)}{\lambda_4}$
**if** $v < \frac{\lambda_4(x_4-x)+1}{u}$ **then go to** 40
**if** $x \geq 1$ **then go to** 13
**else**
    $v \leftarrow v f_4 u$
$P \leftarrow \log(v)$
**if** $P > -(x-x_3)^2(C+C)$ **then go to** 13
**if** $P > A \log({}^{x}/_{A}) + B \log({}^{(1-x)}/_{B}) + L$ **then go to** 13
**else**
    **return** $x$

**Algorithm 18** B4PE algorithm (Schmeiser and Babu [21])

**Require:** Parameters $\alpha, \beta > 1$
**Output:** Random sample $x$ with distribution $beta(\alpha, \beta)$
INITIALIZATION
$A \leftarrow \alpha - 1, B \leftarrow \beta - 1, C \leftarrow A + B, L \leftarrow C \log C, x_1 \leftarrow x_2 \leftarrow 0, x_3 \leftarrow \frac{A}{C}, x_4 \leftarrow x_5 \leftarrow 1, f_1 \leftarrow f_2 \leftarrow f_4 \leftarrow f_5 \leftarrow 0$
**if** $C \leq 1$ **then** go to 18
$D \leftarrow \frac{(AB/(C-1))^{1/2}}{C}$
**if** $D \geq x_3$ **then go to** 11
**else**
    $x_2 \leftarrow x_3 - D$
    $x_1 \leftarrow x_2 - \frac{x_2(1-x_2)}{(A-Cx_2)}$
    $\lambda_1 \leftarrow \frac{A}{x_1} - \frac{B}{(1-x_1)}$
    $f_1 \leftarrow \exp\left(A \log\left(x_1/A\right) + B \log\left((1-x_1)/B\right) + L\right)$
    $f_2 \leftarrow \exp\left(A \log\left(x_2/A\right) + B \log\left((1-x_2)/B\right) + L\right)$
**if** $x_3 + D \geq 1$ **then go to** 18
**else**
    $x_4 \leftarrow x_3 + D$
    $x_5 \leftarrow x_4 - \frac{x_4(1-x_4)}{(A-Cx_4)}$
    $\lambda_5 \leftarrow \frac{B}{(1-x_5)} - \frac{A}{x_5}$
    $f_4 \leftarrow \exp\left(A \log\left(x_4/A\right) + B \log\left((1-x_4)/B\right) + L\right)$
    $f_5 \leftarrow \exp\left(A \log\left(x_5/A\right) + B \log\left((1-x_5)/B\right) + L\right)$
$p_1 \leftarrow f_2(x_3 - x_2)$
$p_2 \leftarrow f_4(x_4 - x_3) + p_1$
$p_3 \leftarrow f_1(x_2 - x_1) + p_2$
$p_4 \leftarrow f_5(x_5 - x_4) + p_3$
$p_5 \leftarrow (1 - f_2)(x_3 - x_2) + p_4$
$p_6 \leftarrow (1 - f_4)(x_4 - x_3) + p_5$
$p_7 \leftarrow (f_2 - f_1)(x_2 - x_1)/2 + p_6$
$p_8 \leftarrow (f_4 - f_5)(x_5 - x_4)/2 + p_7$
$p_9 \leftarrow \frac{f_1}{\lambda_1} + p_8$
$p_{10} \leftarrow \frac{f_5}{\lambda_5} + p_9$
GENERATION
Generate $u \sim U(0,1)$
$u \leftarrow u p_{10}$
**if** $u > p_4$ **then go to** 42
**else**
    **if** $u > p_1$ **then go to** 35
    **else** ▷ Region 1
        **return** $x \leftarrow x_2 + u/f_2$
    **if** $u > p_2$ **then go to** 38
    **else** ▷ Region 2
        **return** $x \leftarrow x_3 + (u - p_1)/f_4$
    **if** $u > p_3$ **then go to** 41
    **else** ▷ Region 3
        **return** $x \leftarrow x_1 + (u - p_2)/f_1$
    **return** $x \leftarrow x_4 + (u - p_3)/f_5$ ▷ Region 4
Generate $w \sim U(0,1)$
**if** $u > p_5$ **then go to** 49
**else** ▷ Region 5
    $x \leftarrow x_2 + w(x_3 - x_2)$
    **if** $\frac{(u-p_4)}{(p_5-p_4)} \leq w$ **then go to** 86
    **else**
        $v \leftarrow f_2 + \frac{u-p_4}{(x_3-x_2)}$ **then go to** 82
**if** $u > p_6$ **then go to** 55
**else** ▷ Region 6
    $x \leftarrow x_3 + w(x_4 - x_3)$
    **if** $\frac{(p_6-u)}{(p_6-p_5)} \geq w$ **then go to** 86
    **else**
        $v \leftarrow f_4 + \frac{u-p_5}{(x_4-x_3)}$ **then go to** 82
**if** $u > p_8$ **then go to** 69
**else** ▷ Region 7
    Generate $w_2 \sim U(0,1)$
    **if** $w_2 > w$ **then** $w \leftarrow w_2$
    **if** $u > p_7$ **then go to** 65
    **else**

61: $x \leftarrow x_1 + w(x_2 - x_1)$
62: $v \leftarrow f_1 + \frac{2w(u-p_6)}{(x_2-x_1)}$
63: **if** $v < f_2 w$ **then go to** 86
64: **else go to** 82
65: $x \leftarrow x_5 - w(x_5 - x_4)$ ▷ Region 8
66: $v \leftarrow f_5 + \frac{2w(u-p_7)}{(x_5-x_4)}$
67: **if** $v \leq f_4 w$ **then go to** 86
68: **else go to** 82
69: **if** $u > p_9$ **then go to** 76 ▷ Region 9
70: $u \leftarrow \frac{p_9-u}{(p_9-p_8)}$
71: $x \leftarrow x_1 + \frac{\log(u)}{\lambda_1}$
72: **if** $x \leq 0$ **then go to** 28
73: **if** $w \leq \frac{\lambda_1(x-x_1)+1}{u}$ **then go to** 86
74: **else**
75: $v \leftarrow w f_1 u$ **then go to** 82
76: $u \leftarrow \frac{p_{10}-u}{(p_{10}-p_9)}$ ▷ Region 10
77: $x \leftarrow x_5 - \frac{\log(u)}{\lambda_5}$
78: **if** $x \geq 1$ **then go to** 28
79: **if** $w \leq \frac{\lambda_5(x_5-x)+1}{u}$ **then go to** 86
80: **else**
81: $v \leftarrow w f_5 u$
82: $P \leftarrow \log(v)$
83: **if** $P > -(x - x_3)^2(C + C)$ **then go to** 28
84: **if** $P > A\log(x/A) + B\log((1-x)/B) + L$ **then go to** 28
85: **else**
86: **return** $x$

In [21], a comparative study of the B2P, B4P, B2PE, B4PE and BB algorithms on a CDC CYBER 72 computer was carried out by programming them in FORTRAN. As a result, the B2PE and B4PE algorithms dominated the B2P and B4P algorithms, respectively. Furthermore, both the B2PE and B4PE algorithms dominated the BB algorithm, with the B4PE algorithm being approximately twice as fast as the BB algorithm.

### *3.4. Other Methods of Generation*

There are other methods for generating beta random variables. Some of them are based on certain statistical properties of random variables, order statistics, stratified rejection methods or stochastic search procedures, among others.

The method based on gamma random variables is one of the most used methods. It is based on the following result (see [32]). If $Y$ and $Z$ are two independent standard gamma random variables with parameters $\alpha$ and $\beta$, respectively, then $X = Y/(Y+Z) \sim beta(\alpha, \beta)$. Given this property, it is possible to state that any existing method for simulating gamma random variables can be adopted and used to generate a beta distribution. Algorithm 19 shows this method.

**Algorithm 19** Method for generating a beta r.v. based on a gamma r.v. [37]

**Require:** Parameters $\alpha$ y $\beta$
**Output:** R.v. $X \sim beta(\alpha, \beta)$
Generate $Y \sim Gamma(\alpha)$
Generate $Z \sim Gamma(\beta)$
$X \leftarrow \frac{Y}{Y+Z}$
**return** $X$

Another relational property of the beta r.v. allows generating samples of this distribution from the order statistics of a sample of a $U(0,1)$ r.v. as explained in [53]. If $0 < U_{(1)} < \cdots < U_{(\alpha+\beta-1)} < 1$ is an ordered sample of a size $(\alpha + \beta - 1)$ from a $U(0,1)$ distribution, then $U_{(\alpha)} \sim beta(\alpha, \beta)$. Thus, a sample from a $beta(\alpha, \beta)$ r.v. with $\alpha, \beta \in \mathbb{N}$ can be generated by taking the $\alpha$th smallest value from a uniform sample of a size $(\alpha + \beta - 1)$.

Algorithm 20 requires generating a whole sample from a uniform distribution to obtain a single value of the desired beta distribution, which is computationally cumbersome, especially if large parameters are considered. In addition, standard ordering of the sample to find the the smallest $\alpha$th

value may not be too difficult for small parameters, but when taking large $\alpha$ or $\beta$ values, the cost rises. As a solution to this problem, other faster sorting algorithms such as SELECT [54] or SORT [55] can be used as a solution to this problem.

**Algorithm 20** Method based on order statistics [32]

**Require:** Parameters $\alpha$ and $\beta$
**Output:** Random sample $x$ of a $beta(\alpha, \beta)$ r.v.
Generate a sample of size $(\alpha + \beta - 1)$ from a $U(0, 1)$ r.v.
$x \leftarrow \alpha$th order statistics
**return** $x$

In relation to stratified rejection methods, the B00, B01 and B11 algorithms developed in [56], in which envelopes and exponentials are applied to slices in the centres and tails of the desired beta distribution, respectively, stand out. Some improvements on the traditional A–R algorithms are the BPRS and BPRB algorithms developed in [57], which improve the acceptance and rejection at the centre of the beta r.v. for simulation using the idea of patch rejection. When $\alpha, \beta < 1$, the B00 algorithm has the fastest computational generation time among all compared algorithms. When $\alpha < 1 < \beta$ or $\beta < 1 < \alpha$ , the B01 algorithm is the fastest. When $\alpha, \beta > 1$, if one parameter is close to one and the other is large, then the B4PE algorithm has the shortest computer generation time; otherwise, the BPRS algorithm has the shortest computer generation time.

Another procedure is the stochastic search method developed in [58] that asymptotically generates beta random variables. This method has some drawbacks and was studied and improved upon in [59] (MK algorithm). In this paper, the authors indicate the parameter values of the beta r.v. that allow it to be generated with Kennedy's algorithm and perform a study on the optimal parameter selection. The MK algorithm proposed is faster than all the compared algorithms for generating the beta r.v. when $\alpha, \beta < 1$, and $1.2 < \alpha + \beta < 2$. In [60], a universal generator is proposed for absolute-continuous and integer-valued random variables. This algorithm is based on the previous work [61] and is a generalization of the A-R method. There are other recent techniques, which also include analysis through neutrosophic statistics (see [62]), analysing the generation of beta distributions using the neutrosophic acceptance–rejection method (see [63]).

## 4. Computational Development

This section compares the previously described methods to find which one is best for simulating beta random variables. The computational results were derived from the implementation of the different algorithms in R, the generation of 5 beta random samples of a size of 10,000 for each of them and the calculation of the average execution times of these for a wide range of parameters on a laptop with an Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz (2.71 GHz) and 8 GB of RAM. The choices for the sample sizes and parameters were in relation to those made by the authors of the algorithms themselves in their comparisons [46,47,49,50]. The uniform pseudo-random numbers were generated by means of the function RUNIF [64]. In turn, the random samples of the normal, gamma and exponential distributions needed in some of the algorithms were also generated by their respective functions already existing in R: RNORM, RGAMMA and REXP [64]. The tables that stored the execution times always took $\alpha \leq \beta$. For the opposite case, it was enough to generate $1 - X$ instead of $X$ [37], and therefore the times hardly varied.

To begin with, for the algorithm based on the order statistics [53], we performed two different sorts of uniform random sampling: the first using the SELECT algorithm [54] and the second using the SORT algorithm [55], which is included in R (the command SORT). By looking at Table 1, it can be noted that when using SORT, the algorithm was not all that sensitive to the value taken by the parameters, the execution time remained between 400 and 600 ms at all times, unlike the algorithm with SELECT sorting, which was highly volatile. Using the SELECT [54] algorithm to sort the uniform random sample was quite interesting when the parameters $\alpha$ and $\beta$ were small; its execution times were much better than with SORT when $\alpha$ or $\beta$ did not take values greater than 10. Otherwise, the time needed to simulate a beta r.v. skyrocketed, taking up to 4 s for a $beta(100, 100)$. This is why it is not recommended to use SELECT sorting for large parameters.

In Table 1, we can also observe the execution times of Jöhnk's algorithm [41] and the algorithm based on the gamma r.v. [32] for the parameters $\alpha, \beta \geq 1$, while in Table 2, the times for these same algorithms but with $\alpha < 1$ and any $\beta$ value are collected. In Tables 3 and 4 we show the average number of iterations required for the simulations in BN, Cheng's and Jöhnk's algorithms and Cheng's, Jöhnk's and Switch algorithms respectively. When analysing the times used by Jöhnk's algorithm,

it is necessary to restrict the use of this algorithm to small parameters. For $\alpha, \beta = 5$, the algorithm already needed more than 3 s to generate the beta r.v., needing an average of 252 iterations as shown in Table 3. Similarly, if we tried to simulate a $beta(50, 100)$, an average number of iterations to the order of 40 was reached. However, neither this nor the algorithms based on order statistics were comparable to the method based on the gamma r.v. This method was faster for any type of parameter; it took no more than 5 ms to simulate any of the beta random variables studied. Moreover, it was 10 times more efficient than the best case of either of the other two methods. Therefore, if the objective is to simulate a beta r.v. through one of the methods based on relational properties, then the algorithm based on the gamma r.v. is recommended, as it was the fastest and took advantage of the function RGAMMA in R [64], which allowed us to generate random samples. Gamma is easily programmable, as it is only necessary to generate two samples and operate between them.

Next, we will compare the inversion algorithms: the bisection algorithm [32] based on the root-finding algorithms and the NINIGL algorithm [31]. The execution times for these methods are collected in Tables 5 and 6. Looking at the times for the bisection method, it is clear that this algorithm was slow, as we predicted, but it is interesting to note the stability of its times. Regardless of the parameters taken, it took around 0.7 or 0.8 s to simulate a beta r.v. On the other hand, we have the NINIGL algorithm. When studying the times of this algorithm, it is highlighted that, in spite of the speed with which it generated a beta r.v. of any parameters, when $\alpha$ or $\beta$ were less than 0.5, the algorithm slowed down quite a lot and became invalid when both parameters were less than 0.5. This is because when a beta distribution takes on small parameters, its density reaches values close to zero, and this causes numerical problems in the algorithm. However, in spite of this, this method was clearly faster than the bisection method. In addition, this algorithm was programmed in the RUNURAN library, which is an adaptation of the C library of Universal Non-Uniform Random Number Generators UNU.RAN) under the name Polynomial Interpolation of Inverse CDF (PINV), which makes it more attractive for use in simulating beta random variables.

**Table 1.** Execution time (in milliseconds) for relational property-based algorithms ($\alpha, \beta \geq 1$).

| Parameters | | Methods | | | |
|---|---|---|---|---|---|
| **Alpha** | **Beta** | **Gamma** | **SELECT** | **SORT** | **Jöhnk** |
| 1 | 1 | 3.00 | 39.29 | 439.84 | 91.16 |
| | 2 | 3.18 | 103.73 | 433.88 | 135.44 |
| | 5 | 2.80 | 166.32 | 445.73 | 282.73 |
| | 10 | 2.99 | 244.14 | 441.03 | 488.30 |
| | 50 | 2.59 | 646.80 | 476.25 | 2243.62 |
| | 100 | 2.59 | 1257.66 | 512.18 | * |
| 2 | 2 | 3.20 | 158.97 | 431.96 | 272.88 |
| | 5 | 2.59 | 222.41 | 449.40 | 938.62 |
| | 10 | 2.79 | 296.78 | 435.65 | 2761.89 |
| | 50 | 2.59 | 710.20 | 475.56 | * |
| | 100 | 2.19 | 1334.46 | 517.91 | * |
| 5 | 5 | 3.19 | 306.19 | 435.85 | * |
| | 10 | 2.20 | 367.11 | 445.19 | * |
| | 50 | 2.19 | 816.16 | 478.66 | * |
| | 100 | 2.39 | 1513.51 | 525.20 | * |
| 10 | 10 | 2.19 | 475.53 | 466.57 | * |
| | 50 | 2.00 | 958.83 | 478.22 | * |
| | 100 | 2.40 | 1736.97 | 521.37 | * |
| 50 | 50 | 1.78 | 1955.14 | 511.28 | * |
| | 100 | 1.80 | 2842.27 | 553.92 | * |
| 100 | 100 | 2.19 | 4028.12 | 587.14 | * |

* Excessive time.

**Table 2.** Execution time (in milliseconds) for relational property-based algorithms ($\alpha < 1$).

| Parameters | | Methods | | Parameters | | Methods | |
|---|---|---|---|---|---|---|---|
| **Alpha** | **Beta** | **Gamma** | **Jöhnk** | **Alpha** | **Beta** | **Gamma** | **Jöhnk** |
| 0.1 | 0.1 | 4.38 | 47.88 | 0.5 | 0.5 | 4.19 | 57.45 |
| | 0.3 | 4.80 | 50.87 | | 0.9 | 4.39 | 71.59 |
| | 0.5 | 3.99 | 52.57 | | 1 | 4.79 | 76.99 |
| | 0.9 | 4.18 | 56.96 | | 2 | 3.79 | 97.68 |
| | 1 | 2.99 | 55.85 | | 5 | 3.18 | 119.79 |
| | 2 | 3.20 | 55.65 | | 10 | 3.19 | 165.12 |
| | 5 | 2.99 | 61.04 | | 50 | 3.20 | 365.27 |
| | 10 | 2.79 | 63.72 | | 100 | 3.19 | 510.84 |
| | 50 | 2.60 | 72.91 | | | | |
| | 100 | 2.79 | 84.88 | | | | |
| 0.3 | 0.3 | 3.80 | 69.78 | 0.9 | 0.9 | 4.18 | 108.61 |
| | 0.5 | 4.09 | 56.85 | | 1 | 3.59 | 104.32 |
| | 0.9 | 4.38 | 61.88 | | 2 | 3.60 | 126.26 |
| | 1 | 3.60 | 59.94 | | 5 | 3.39 | 245.40 |
| | 2 | 3.39 | 71.03 | | 10 | 3.38 | 421.22 |
| | 5 | 2.99 | 86.98 | | 50 | 2.99 | 1578.84 |
| | 10 | 2.99 | 108.32 | | 100 | 3.19 | 2835.19 |
| | 50 | 3.19 | 218.72 | | | | |
| | 100 | 2.99 | 220.33 | | | | |

**Table 3.** Average number of iterations required ($\alpha, \beta \geq 1$).

| Parameters | | Methods | | |
|---|---|---|---|---|
| **Alpha** | **Beta** | **BN *** | **Cheng** | **Jöhnk** |
| 1 | 1 | 28.04 | 1 | 2.00 |
| | 2 | 2.489 | 1.185 | 3.00 |
| | 5 | 3.111 | 1.34 | 6.00 |
| | 10 | 4.148 | 1.402 | 11.00 |
| | 50 | 8.887 | 1.457 | 51.00 |
| | 100 | - | 1.464 | 101.00 |
| 2 | 2 | 1.329 | 1.061 | 6.00 |
| | 5 | 1.377 | 1.13 | 21.00 |
| | 10 | 1.689 | 1.18 | 66.00 |
| | 50 | 3.359 | 1.234 | 1326.00 |
| | 100 | 5.025 | 1.242 | 5151.00 |
| 5 | 5 | 1.09 | 1.101 | 252.00 |
| | 10 | 1.139 | 1.117 | 3003.00 |
| | 50 | 1.889 | 1.155 | $3.5 \times 10^6$ |
| | 100 | 2.588 | 1.162 | $9.7 \times 10^7$ |
| 10 | 10 | 1.041 | 1.114 | $1.8 \times 10^5$ |
| | 50 | 1.391 | 1.134 | $7.5 \times 10^{10}$ |
| | 100 | 1.809 | 1.141 | $4.7 \times 10^{13}$ |
| 50 | 50 | | 1.126 | $1.0 \times 10^{29}$ |
| | 100 | | | $2.0 \times 10^{40}$ |

* Take 1.001 instead 1 for $\alpha$ and $\beta$.

Thirdly, Tables 7 and 8 collect the time (in milliseconds) that was necessary to generate beta random variables from what we called specific methods. On the one hand, Table 7 stores the times of those methods applicable when both parameters were greater than one (i.e., the A–R algorithms from the density of a normal distribution [46], the BN, BNM and symmetric case BS algorithms [50], the 2-points B2P algorithm [50] together with its version using the tail of the exponential B2PE algorithm and the extension to the 4-points B4PE algorithm [21] and the BA and BB algorithms [49]). On the other hand, the execution times of the algorithms applicable when at least one of the parameters

was less than one (i.e., the "switching algorithm" [47] (SW1 and SW2) and Cheng's BA and BC algorithms [49]) are stored in Table 8. In addition, the runtimes of the function of R that generated random samples from a beta distribution RBETA are also collected in both tables.

**Table 4.** Average number of iterations required ($\alpha < 1$).

| Parameters | | Methods | | | Parameters | | Methods | | |
|---|---|---|---|---|---|---|---|---|---|
| **Alpha** | **Beta** | **Cheng** | **Jöhnk** | **Switch *** | **Alpha** | **Beta** | **Cheng** | **Jöhnk** | **Switch *** |
| 0.1 | 0.1 | 1.77 | 1.01 | 1.77 | 0.5 | 0.5 | 1.27 | 1.27 | 1.27 |
| | 0.3 | 2.49 | 1.04 | 1.53 | | 0.9 | 1.50 | 1.46 | 1.09 |
| | 0.5 | 2.70 | 1.06 | 1.35 | | 1 | 1.54 | 1.50 | 1.15 |
| | 0.9 | 2.84 | 1.09 | 1.09 | | 2 | 1.72 | 1.88 | 1.21 |
| | 1 | 2.86 | 1.10 | 1.03 | | 5 | 1.84 | 2.71 | 1.25 |
| | 2 | 2.94 | 1.16 | 1.05 | | 10 | 1.89 | 3.70 | 1.26 |
| | 5 | 2.99 | 1.25 | 1.07 | | 50 | 1.93 | 8.04 | 1.28 |
| | 10 | 3.01 | 1.33 | 1.08 | | 100 | 1.93 | 11.33 | 1.28 |
| | 50 | 3.02 | 1.56 | 1.09 | | | | | |
| | 100 | 3.02 | 1.67 | 1.09 | | | | | |
| 0.3 | 0.3 | 1.46 | 1.11 | 1.46 | 0.9 | 0.9 | 1.04 | 1.81 | 1.04 |
| | 0.5 | 1.72 | 1.17 | 1.34 | | 1 | 1.07 | 1.90 | 1.16 |
| | 0.9 | 1.95 | 1.28 | 1.10 | | 2 | 1.26 | 2.76 | 1.17 |
| | 1 | 1.98 | 1.30 | 1.09 | | 5 | 1.41 | 5.18 | 1.19 |
| | 2 | 2.13 | 1.50 | 1.14 | | 10 | 1.47 | 8.96 | 1.19 |
| | 5 | 2.23 | 1.87 | 1.18 | | 50 | 1.52 | 35.76 | 1.20 |
| | 10 | 2.27 | 2.27 | 1.20 | | 100 | 1.53 | 66.16 | 1.20 |
| | 50 | 2.29 | 3.62 | 1.21 | | | | | |
| | 100 | 2.30 | 4.44 | 1.21 | | | | | |

* Take $\beta = 1.001$ instead of $\beta = 1$.

**Table 5.** Execution times (in milliseconds) for inversion algorithms ($\alpha, \beta \geq 1$).

| Parameters | | Methods | |
|---|---|---|---|
| **Alpha** | **Beta** | **Bisection** | **NINIGL** |
| 1 | 1 | 687.40 | 0.60 |
| | 2 | 760.75 | 1.40 |
| | 5 | 761.26 | 1.00 |
| | 10 | 771.29 | 1.20 |
| | 50 | 765.05 | 1.00 |
| | 100 | 738.42 | 1.40 |
| 2 | 2 | 732.41 | 2.59 |
| | 5 | 803.14 | 2.39 |
| | 10 | 821.89 | 2.62 |
| | 50 | 838.48 | 2.39 |
| | 100 | 771.99 | 2.19 |
| 5 | 5 | 835.62 | 1.60 |
| | 10 | 819.31 | 1.20 |
| | 50 | 802.67 | 1.60 |
| | 100 | 795.96 | 1.60 |
| 10 | 10 | 792.23 | 1.00 |
| | 50 | 794.36 | 1.59 |
| | 100 | 810.24 | 1.60 |
| 50 | 50 | 756.47 | 1.20 |
| | 100 | 876.26 | 1.40 |
| 100 | 100 | 876.65 | 1.39 |

**Table 6.** Execution times (in milliseconds) for inversion algorithms ($\alpha < 1$).

| Parameters | | Methods | | Parameters | | Methods | |
|---|---|---|---|---|---|---|---|
| **Alpha** | **Beta** | **Bisection** | **NINIGL** | **Alpha** | **Beta** | **Bisection** | **NINIGL** |
| 0.1 | 0.1 | 743.93 | - | 0.5 | 0.5 | 739.83 | 3.39 |
| | 0.3 | 755.05 | - | | 0.9 | 795.65 | 3.99 |
| | 0.5 | 753.33 | 44.89 | | 1 | 744.47 | 3.06 |
| | 0.9 | 820.23 | 43.78 | | 2 | 758.80 | 4.39 |
| | 1 | 727.21 | 41.89 | | 5 | 756.77 | 4.59 |
| | 2 | 842.61 | 38.78 | | 10 | 782.66 | 4.79 |
| | 5 | 834.20 | 39.70 | | 50 | 747.51 | 4.79 |
| | 10 | 850.57 | 37.70 | | 100 | 745.92 | 4.29 |
| | 50 | 805.91 | 37.10 | | | | |
| | 100 | 813.52 | 37.10 | | | | |
| 0.3 | 0.3 | 759.32 | - | 0.9 | 0.9 | 738.04 | 3.39 |
| | 0.5 | 713.15 | 11.57 | | 1 | 704.67 | 2.79 |
| | 0.9 | 714.34 | 14.76 | | 2 | 790.74 | 3.59 |
| | 1 | 848.19 | 13.36 | | 5 | 817.37 | 3.39 |
| | 2 | 783.48 | 12.77 | | 10 | 827.93 | 3.10 |
| | 5 | 858.25 | 11.37 | | 50 | 807.46 | 3.59 |
| | 10 | 848.70 | 10.97 | | 100 | 813.41 | 2.99 |
| | 50 | 832.72 | 12.17 | | | | |
| | 100 | 798.93 | 11.17 | | | | |

**Table 7.** Execution times (in milliseconds) for specific algorithms ($\alpha, \beta \geq 1$).

| Parameters | | Methods | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Alpha** | **Beta** | **B2P *** | **B2PE *** | **B4PE *** | **BA** | **BB** | **rbeta** | **BN *** | **BNM *** | **BS *** |
| 1 | 1 | 138.63 | 63.63 | 58.24 | 95.84 | 123.27 | 2.00 | 671.40 | - | 735.03 |
| | 2 | 264.49 | 113.70 | 107.12 | 111.30 | 140.42 | 2.19 | 107.11 | 126.67 | |
| | 5 | 378.99 | 77.59 | 59.83 | 123.48 | 147.61 | 2.59 | 144.81 | 169.95 | |
| | 10 | 721.07 | 69.22 | 59.64 | 140.02 | 168.14 | 2.60 | 185.90 | 227.98 | |
| | 50 | 3392.54 | 64.83 | 56.45 | 132.25 | 157.98 | 2.59 | 458.77 | 499.47 | |
| | 100 | 6830.86 | 64.02 | 59.45 | 133.24 | 156.58 | 2.59 | 555.32 | 700.73 | |
| 2 | 2 | 208.54 | 82.78 | 79.38 | 100.54 | 122.87 | 1.99 | 79.98 | 91.15 | 58.24 |
| | 5 | 187.99 | 75.40 | 60.84 | 105.71 | 125.46 | 2.00 | 79.00 | 106.72 | |
| | 10 | 287.77 | 75.40 | 60.44 | 110.31 | 132.45 | 2.19 | 92.35 | 169.94 | |
| | 50 | 1024.27 | 75.41 | 57.84 | 115.29 | 134.64 | 2.39 | 163.76 | 263.10 | |
| | 100 | 2094.05 | 77.39 | 66.42 | 115.70 | 134.45 | 2.19 | 223.61 | 334.71 | |
| 5 | 5 | 175.33 | 78.99 | 48.08 | 103.92 | 128.05 | 2.19 | 73.00 | 99.13 | 49.67 |
| | 10 | 203.26 | 67.63 | 47.87 | 106.31 | 125.47 | 2.00 | 75.80 | 98.34 | |
| | 50 | 619.75 | 70.21 | 49.87 | 107.31 | 126.26 | 2.19 | 113.50 | 169.15 | |
| | 100 | 1065.74 | 69.22 | 51.46 | 111.90 | 132.45 | 1.99 | 148.80 | 230.98 | |
| 10 | 10 | 210.44 | 68.21 | 48.27 | 105.52 | 130.65 | 2.20 | 70.30 | 76.20 | 45.08 |
| | 50 | 441.21 | 70.81 | 49.07 | 109.30 | 123.87 | 1.99 | 92.55 | 108.71 | |
| | 100 | 745.02 | 73.00 | 49.06 | 106.12 | 126.27 | 2.19 | 125.07 | 147.80 | |
| 50 | 50 | 391.35 | 66.23 | 46.88 | 107.11 | 128.25 | 2.59 | 65.83 | 72.21 | 45.07 |
| | 100 | 492.49 | 67.81 | 45.67 | 106.32 | 128.06 | 2.39 | 72.30 | 81.38 | |
| 100 | 100 | 515.02 | 66.62 | 49.07 | 105.32 | 136.04 | 2.39 | 70.60 | 75.20 | 45.48 |

* Take $\alpha = 1.001$ instead of $\alpha = 1$ and $\beta = 1.001$ instead of $\beta = 1$.

**Table 8.** Execution times (in milliseconds) for specific algorithms ($\alpha < 1$).

| Parameters | | Methods | | | | |
|---|---|---|---|---|---|---|
| **Alpha** | **Beta** | **BA** | **BC** | **rbeta** | **SW1 *** | **SW2** |
| 0.1 | 0.1 | 166.39 | 196.48 | 3.59 | | 112.49 |
| | 0.3 | 219.61 | 229.19 | 3.19 | | 97.94 |
| | 0.5 | 242.16 | 237.17 | 2.79 | | 76.80 |
| | 0.9 | 239.15 | 255.32 | 2.99 | | 68.41 |
| | 1 | 257.91 | 246.14 | 2.99 | 63.63 | |
| | 2 | 270.68 | 244.35 | 3.19 | 68.42 | |
| | 5 | 255.17 | 248.34 | 2.39 | 70.82 | |
| | 10 | 254.53 | 246.14 | 2.59 | 66.62 | |
| | 50 | 261.70 | 246.14 | 2.59 | 69.41 | |
| | 100 | 253.52 | 247.14 | 2.59 | 75.60 | |
| 0.3 | 0.3 | 142.62 | 164.56 | 2.79 | | 89.17 |
| | 0.5 | 153.19 | 179.12 | 2.39 | | 85.76 |
| | 0.9 | 169.95 | 203.65 | 2.79 | | 68.02 |
| | 1 | 173.73 | 191.09 | 2.59 | 63.23 | |
| | 2 | 185.50 | 202.46 | 2.79 | 69.01 | |
| | 5 | 192.09 | 207.84 | 2.39 | 71.81 | |
| | 10 | 192.08 | 206.05 | 2.59 | 73.44 | |
| | 50 | 196.27 | 208.44 | 2.59 | 74.99 | |
| | 100 | 199.07 | 210.04 | 2.39 | 75.60 | |
| 0.5 | 0.5 | 118.89 | 153.99 | 2.39 | | 76.20 |
| | 0.9 | 135.03 | 162.76 | 2.59 | | 64.23 |
| | 1 | 139.83 | 167.55 | 2.59 | 70.41 | |
| | 2 | 163.56 | 179.12 | 2.60 | 72.40 | |
| | 5 | 161.56 | 198.47 | 2.79 | 75.80 | |
| | 10 | 166.16 | 188.89 | 2.59 | 77.20 | |
| | 50 | 168.56 | 189.49 | 2.58 | 77.39 | |
| | 100 | 177.92 | 191.69 | 2.59 | 78.39 | |
| 0.9 | 0.9 | 99.54 | 129.65 | 2.19 | | 67.42 |
| | 1 | 101.92 | 135.64 | 2.20 | 73.80 | |
| | 2 | 118.48 | 146.21 | 2.39 | 72.21 | |
| | 5 | 128.86 | 160.76 | 2.40 | 74.20 | |
| | 10 | 132.45 | 165.36 | 2.79 | 73.61 | |
| | 50 | 137.43 | 169.75 | 2.40 | 75.79 | |
| | 100 | 135.84 | 174.13 | 2.59 | 75.00 | |

* Take $\alpha = 1.001$ instead of $\alpha = 1$ and $\beta = 1.001$ instead of $\beta = 1$.

If we begin by comparing Cheng's algorithms, it should be noted that although the BB and BC algorithms are supposed to be improvements upon the BA algorithm by reducing the number of logarithmic evaluations required, in our codes, the time spent on these evaluations was less than the time required to run the additional tests of the BB and BC algorithms, and therefore, the BA algorithm was uniformly faster than the other two. In addition, since it does not have the additional tests, it is simpler to code. When contrasting this algorithm with the "switching algorithm" of Atkinson and Whittaker [47] either when both parameters were less than one (SW2) or when only one was (SW1), we see that Atkinson and Whittaker's algorithm was faster than the BA algorithm for all parameters where both algorithms were valid. Furthermore, although the average number of iterations required when simulating symmetric beta random variables was the same, in general, Atkinson and Whittaker's algorithm [47] required fewer iterations, as shown in Table 4. It is also interesting to note the behaviour of the SW2 algorithm which, as the beta distribution became more asymmetric, became more efficient. The SW1 version, on the other hand, did not follow this pattern. Another algorithm that followed a pattern, even if it was the opposite of the SW2 algorithm's, was the 2-points B2P [50] algorithm. This algorithm became worse as some of the parameters took on larger values because the major function did not fit well in the tails, and it took almost 7 s to simulate a $beta(1, 100)$ r.v. Looking at Table 7 proves that, indeed, the supposed improvement of the 2-points algorithm proposed by Schmeiser and Babu, the B2PE algorithm [21], which uses exponentials to fit the majorising function better in the tails of the beta distribution, behaved as such. The B2PE

method is noteworthy for its constancy; the times required to simulate a beta r.v. were generally between 60 and 70 ms, which is very different from the behaviour of the B2P algorithm. However, this constancy of the B2PE method was also present in the 4-points version (B4PE) [21]. Between these two algorithms, it is complicated to establish which is the best. The B4PE algorithm was faster than the B2PE one, as shown in Table 7. However, the difference was minimal, not exceeding 30 ms, and the B4PE algorithm was more extensive at the time of programming. Therefore, this decision is left to the user's choice. Regardless of this choice, the adapted BS algorithm of Ahrens and Dieter [46] was faster and simpler than the previous two, but it is only valid for symmetric cases where $\alpha = \beta$ and the original BN algorithm [46], applicable for any beta r.v. with parameters greater than one, was not as efficient. Moreover, the supposed improvement of the BN algorithm proposed by Schmeiser and Shalaby (BNM) [50] was not computationally so. The computation of the $x_1, \ldots, x_5$ points and the additional step to evaluate the minorant function were more time-consuming than the evaluations of the original method they were trying to avoid. One remarkable thing about these three algorithms is how little efficiency they had for $\alpha = \beta = 1.001$; the BNM algorithm could not be applied directly, and the times needed for the BN and BS algorithms were far from those needed for the rest of the parameters. The reason for this behaviour is that for these parameters, it took around 28 iterations to generate a sample of the beta r.v., as shown in Table 3. However, the best option for simulating a beta r.v. among the algorithms present in Tables 7 and 8 was to use RBETA. When both parameters were greater than one, this function was about 25 times faster than those considered thus far to be more efficient. For the B2PE and B4PE or BS algorithms in the symmetric case and, similarly, when one of the parameters was less than one, the execution speed was at least 20 times faster than that of the "switching algorithm" (SW1 and SW2). The most interesting thing about RBETA is that, according to the documentation of R [64], this function follows the methods of Cheng [49], and yet it is significantly more efficient than the algorithms of Cheng which we programmed (BA, BB and BC). When taking this into account and that, for the tables with the rest of the methods, the algorithms that stood out for their speed could also be found implemented in R (NINIGL) or only needed to perform small transformations from R functions (algorithms based on gamma random variables), it can be stated that the best options for simulating a beta r.v. are already implemented in R.

Finally, we can still compare the three most efficient algorithms (gamma, NINIGL and RBETA) to obtain the best one. To accomplish this, Tables 9 and 10 collect the average execution times of these algorithms. On the one hand, when one of the parameters was less than one, it can clearly be observed that RBETA is the best choice; the execution times required for the gamma r.v.-based algorithm did not differ much from those required by RBETA, but they were slower, and the algorithm requires some programming. The NINIGL algorithm suffered a lot with small parameters and did not become a good choice in this situation. On the other hand, when both parameters were higher than one, the three algorithms were more in tune. While all three were efficient, the NINIGL algorithm stood out, in general, as the fastest one. Therefore, for each of the two situations, depending on the values taken by the parameters, the best options are those already mentioned. However, if we want an efficient algorithm for any parameter, then the best option for simulating beta random variables is to use RBETA. However, if we want to generate beta random samples through the implementation of any of the algorithms other than those already present in R, when reviewing the tables that collected the average execution times, the best option is the B4PE or B2PE algorithm if both parameters are greater than one, SW1 and SW2 algorithms if at least one parameter is less than one and the BA algorithm in case we are looking for one that is valid for any parameter.

**Table 9.** Execution times (in milliseconds) for algorithms implemented in R ($\alpha, \beta \geq 1$).

| Parameters | | Methods | | |
|---|---|---|---|---|
| **Alpha** | **Beta** | **Gamma** | **NINIGL** | **RBETA** |
| 1 | 1 | 3.00 | 0.60 | 2.00 |
| | 2 | 3.18 | 1.40 | 2.19 |
| | 5 | 2.80 | 1.00 | 2.59 |
| | 10 | 2.99 | 1.20 | 2.60 |
| | 50 | 2.59 | 1.00 | 2.59 |
| | 100 | 2.59 | 1.40 | 2.59 |
| 2 | 2 | 3.20 | 2.59 | 1.99 |
| | 5 | 2.59 | 2.39 | 2.00 |
| | 10 | 2.79 | 2.62 | 2.19 |
| | 50 | 2.59 | 2.39 | 2.39 |
| | 100 | 2.19 | 2.19 | 2.19 |
| 5 | 5 | 3.19 | 1.60 | 2.19 |
| | 10 | 2.20 | 1.20 | 2.00 |
| | 50 | 2.19 | 1.60 | 2.19 |
| | 100 | 2.39 | 1.60 | 1.99 |
| 10 | 10 | 2.19 | 1.00 | 2.20 |
| | 50 | 2.00 | 1.59 | 1.99 |
| | 100 | 2.40 | 1.60 | 2.19 |
| 50 | 50 | 1.78 | 1.20 | 2.59 |
| | 100 | 1.80 | 1.40 | 2.39 |
| 100 | 100 | 2.19 | 1.39 | 2.39 |

**Table 10.** Execution times (in milliseconds) for algorithms implemented in R ($\alpha < 1$).

| Parameters | | Methods | | |
|---|---|---|---|---|
| **Alpha** | **Beta** | **Gamma** | **NINIGL** | **RBETA** |
| 0.1 | 0.1 | 4.38 | - | 3.59 |
| | 0.3 | 4.80 | - | 3.19 |
| | 0.5 | 3.99 | 44.89 | 2.79 |
| | 0.9 | 4.18 | 43.78 | 2.99 |
| | 1 | 2.99 | 41.89 | 2.99 |
| | 2 | 3.20 | 38.78 | 3.19 |
| | 5 | 2.99 | 39.70 | 2.39 |
| | 10 | 2.79 | 37.70 | 2.59 |
| | 50 | 2.60 | 37.10 | 2.59 |
| | 100 | 2.79 | 37.10 | 2.59 |
| 0.3 | 0.3 | 3.80 | - | 2.79 |
| | 0.5 | 4.09 | 11.57 | 2.39 |
| | 0.9 | 4.38 | 14.76 | 2.79 |
| | 1 | 3.60 | 13.36 | 2.59 |
| | 2 | 3.39 | 12.77 | 2.79 |
| | 5 | 2.99 | 11.37 | 2.39 |
| | 10 | 2.99 | 10.97 | 2.59 |
| | 50 | 3.19 | 12.17 | 2.59 |
| | 100 | 2.99 | 11.17 | 2.39 |
| 0.5 | 0.5 | 4.19 | 3.39 | 2.39 |
| | 0.9 | 4.39 | 3.99 | 2.59 |
| | 1 | 4.79 | 3.06 | 2.59 |
| | 2 | 3.79 | 4.39 | 2.60 |
| | 5 | 3.18 | 4.59 | 2.79 |
| | 10 | 3.19 | 4.79 | 2.59 |
| | 50 | 3.20 | 4.79 | 2.58 |
| | 100 | 3.19 | 4.29 | 2.59 |
| 0.9 | 0.9 | 4.18 | 3.39 | 2.19 |
| | 1 | 3.59 | 2.79 | 2.20 |
| | 2 | 3.60 | 3.59 | 2.39 |
| | 5 | 3.39 | 3.39 | 2.40 |
| | 10 | 3.38 | 3.10 | 2.79 |
| | 50 | 2.99 | 3.59 | 2.40 |
| | 100 | 3.19 | 2.99 | 2.59 |

## 5. Conclusions

There are currently many high-speed methods capable of simulating beta random variables, but more are still being developed in order to find the fastest, simplest, and most accurate algorithm.

The main objective of this work was to analyse in detail the statistical/mathematical and computational aspects of the generation of the beta random variable and to perform an empirical analysis covering the different generation scenarios. After studying the generation of beta random variables and their implementation in R, it has been concluded that the most efficient way to obtain samples of this type is to use algorithms that are already present in R or that only require a simple transformation. Thus, it is established that the function RBETA of R is the most appropriate option for any type of parameter, especially if any of them are less than one, ahead of the NINIGL algorithm [31], which stands out especially when both parameters are greater than one. In case the reader's objective is to use an algorithm without resorting to R's own algorithms, we recommend implementing Schmeiser and Babu's B4PE or B2PE algorithms [21]. If we only want to simulate beta random variables of parameters greater than one, then the so-called "switching algorithm" (SW1 and SW2) by Atkinson and Whittaker [47] is recommended, as well as if any of the parameters are less than one. Otherwise, the BA algorithm by Cheng [49] is recommended if we want one that is valid for any type of parameter.

However, this work only includes the analysis in R, and through the results presented by the authors of certain algorithms, it was observed that they do not behave in the same way in other languages.

**Author Contributions:** Conceptualization, E.A.L. and C.G.; methodology, E.A.L. and C.G.; software, E.A.L. and C.G.; validation, E.A.L. and C.G.; formal analysis, E.A.L. and C.G.; investigation, E.A.L. and C.G.; resources, E.A.L. and C.G.; data curation, E.A.L. and C.G.; writing—original draft preparation, E.A.L. and C.G.; writing—review and editing, E.A.L. and C.G.; visualization, E.A.L. and C.G.; supervision, E.A.L. and C.G. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** The data presented in this study are available on request from the corresponding author.

**Acknowledgments:** We thank the Editorial Office for their support in the dissemination of this work. We also thank the reviewers for their recommendations to improve the final version of this paper.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Thomas, D.B.; Luk, W.; Leong, P.H.W.; Villasenor, J.D. Gaussian Random Number Generators. *ACM Computing Surv.* **2007**, *39*, 11–es. [CrossRef]
2. Malik, J.S.; Hemani, A. Gaussian Random Number Generation: A Survey on Hardware Architectures. *ACM Comput. Surv.* **2016**, *49*, 1–37. [CrossRef]
3. Maatouk, H.; Bay, X. A new rejection sampling method for truncated multivariate Gaussian random ariables restricted to convex sets. In *Mathematics in Monte Carlo and Quasi-Monte Carlo Methods*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 521–530.
4. Morán-Vásquez, R.; Zarrazola, E.; Nagar, D.K. Some Theoretical and Computational Aspects of the Truncated Multivariate Skew-Normal/Independent Distributions. *Mathematics* **2023**, *11*, 3579. [CrossRef]
5. Almaraz Luengo, E. Gamma Pseudo Random Number Generators. *ACM Comput. Surv.* **2023**, *55*, 1–33. [CrossRef]
6. Rubinstein, R.Y.; Kroese, D.P. *Simulation and the Monte Carlo Method*, 3rd ed.; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2016.
7. Altiok, T.; Melamed, B. *Simulation Modeling and Analysis with ARENA*; Elsevier Science & Technology: New York, NY, USA, 2007.
8. Zhu, S.; Deng, X.; Zhang, W.; Zhu, C. Construction of a New 2D Hyperchaotic Map with Application in Efficient Pseudo-Random Number Generator Design and Color Image Encryption. *Mathematics* **2023**, *11*, 3171. [CrossRef]
9. Bagdasar, O.; Chen, M.; Drăgan, V.; Ivanov, I.G.; Popa, I.L. On Horadam Sequences with Dense Orbits and Pseudo-Random Number Generators. *Mathematics* **2023**, *11*, 1244. [CrossRef]
10. Mahalingam, H.; Rethinam, S.; Janakiraman, S.; Rengarajan, A. Non-Identical Inverter Rings as an Entropy Source: NIST-90B-Verified TRNG Architecture on FPGAs for IoT Device Integrity. *Mathematics* **2023**, *11*, 1049. [CrossRef]
11. Ridley, D.; Ngnepieba, P. Antithetic Power Transformation in Monte Carlo Simulation: Correcting Hidden Errors in the Response Variable. *Mathematics* **2023**, *11*, 2097. [CrossRef]
12. Almaraz Luengo, E. A brief and understandable guide to pseudo-random number generators and specific models for security. *Stat. Surv.* **2022**, *16*, 137–181. [CrossRef]

13. Rukhin, A.L.; Soto, J.; Nechvatal, J.R.; Smid, M.E.; Barker, E.; Leigh, S.; Levenson, M.; Vangel, M.; Banks, D.; Heckert, A.; et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*; Technical Report; NIST: Gaithersburg, MD, USA, 2010.
14. Marsaglia, G. The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness. 1995. Available online: https://web.archive.org/web/20160220101002/http://stat.fsu.edu/pub/diehard/ (accessed on 1 January 2022).
15. Brown, R.G.; Eddelbuettel, D.; Bauer, D. Dieharder: A Random Number Test Suite (Version 3.31.1). 2014. Available online: https://webhome.phy.duke.edu/~rgb/General/dieharder.php (accessed on 1 January 2022).
16. Walker, J. ENT: A Pseudorandom Number Sequence Test Program. 2008. Available online: https://www.fourmilab.ch/random/ (accessed on 1 January 2022).
17. Evans, D.L.; Bond, P.; Bement, A. *FIPS Pub 140-2: Security Requirements for Cryptographic Modules;* Federal Information Processing Standards Publication; NIST: Gaithersburg, MD, USA, 2002; Volume 12.
18. NIST. *FIPS 140-2. Security Requirements for Cryptographic Modules;* Technical Report; NIST: Gaithersburg, MD, USA, 2001.
19. NIST. *FIPS 140-3. Security Requirements for Cryptographic Modules;* Technical Report; NIST: Gaithersburg, MD, USA, 2019.
20. L'ecuyer, P.; Simard, R. TestU01: A C library for empirical testing of random number generators. *ACM Trans. Math. Softw. (TOMS)* **2007**, *33*, 1–40. [CrossRef]
21. Schmeiser, B.W.; Babu, A.J.G. Beta Variate Generation via Exponential Majorizing Functions. *Oper. Res.* **1980**, *28*, 917–926. [CrossRef]
22. Stern, J.M.; de Bragança Pereira, C.A. Special characterizations of standard discrete models. *Revstat-Stat. J.* **2008**, *6*, 199–230.
23. Kuhl, M.E.; Ivy, J.S.; Lada, E.K.; Steiger, N.M.; Wagner, M.A.; Wilson, J.R. Univariate input models for stochastic simulation. *J. Simul.* **2010**, *4*, 81–97. [CrossRef]
24. Thangjai, W.; Niwitpong, S.A.; Niwitpong, S.; Smithpreecha, N. Confidence Interval Estimation for the Ratio of the Percentiles of Two Delta-Lognormal Distributions with Application to Rainfall Data. *Symmetry* **2023**, *15*, 794. [CrossRef]
25. Pfeifer, D.; Ragulina, O. Adaptive Bernstein Copulas and Risk Management. *Mathematics* **2020**, *8*, 2221. [CrossRef]
26. Malcolm, D.G.; Roseboom, J.H.; Clark, C.E.; Fazar, W. Application of a Technique for Research and Development Program Evaluation. *Oper. Res.* **1959**, *7*, 646–669. [CrossRef]
27. Kelley, J.E. Critical-Path Planning and Scheduling: Mathematical Basis. *Oper. Res.* **1961**, *9*, 296–320. [CrossRef]
28. Bętkowski, M.; Pownuk, A. Calculating Risk of Cost Using Monte Carlo Simulations with Fuzzy Parameters in Civil Engineering. In Proceedings of the NSF Workshop on Reliable Engineering Computing, Savannah, Georgia, 15–17 September 2004; pp. 179–192.
29. Price, A.L.; Patterson, N.J.; Plenge, R.M.; Weinblatt, M.E.; Shadick, N.A.; Reich, D. Principal components analysis corrects for stratification in genome-wide association studies. *Nat. Genet.* **2006**, *38*, 904–909. [CrossRef]
30. Kirichenko, L.; Radivilova, T.; Bulakh, V. Machine Learning in Classification Time Series with Fractal Properties. *Data* **2019**, *4*, 5. [CrossRef]
31. Derflinger, G.; Hörmann, W.; Leydold, J. Random Variate Generation by Numerical Inversion When Only the Density is Known. *ACM Trans. Model. Comput. Simul.* **2010**, *20*, 1–25. [CrossRef]
32. Devroye, L. *Non-Uniform Random Variate Generation*; Springer: New York, NY, USA, 1986.
33. Hörmann, W.; Leydold, J. Continuous Random Variate Generation by Fast Numerical Inversion. *ACM Trans. Model. Comput. Simul.* **2003**, *13*, 347–362. [CrossRef]
34. Marsaglia, G.; Tsang, W.W. The ziggurat method for generating random variables. *J. Stat. Softw.* **2000**, *5*, 1–7. [CrossRef]
35. Givens, G.H.; Hoeting, J.A. *Computational Statistics*, 2nd ed.; John Wiley & Sons: Hoboken, NJ, USA, 2012.
36. Baumgarten, C. Random variate generation by fast numerical inversion in the varying parameter case. *Res. Stat.* **2023**, *1*, 2279060. [CrossRef]
37. Law, A.M. *Simulation Modeling and Analysis*, 5th ed.; McGraw-Hill Education: New York, NY, USA, 2015.
38. Von Neumann, J. Various Techniques Used in Connection with Random Digits. In *Monte Carlo Method*; Householder, A.S., Forsythe, G.E., Germond, H.H., Eds.; US Government Printing Office: Washington, DC, USA, 1951; Volume 12, National Bureau of Standards Applied Mathematics Series, Chapter 13, pp. 36–38.
39. Hörmann, W.; Leydold, J.; Derflinger, G. *Automatic Nonuniform Random Variate Generation*, 1st ed.; Springer: Berlin/Heidelberg, Germany, 2004.
40. Marsaglia, G. The squeeze method for generating gamma variates. *Comput. Math. Appl.* **1977**, *3*, 321–325. [CrossRef]
41. Jöhnk, M. Erzeugung von betaverteilten und gammaverteilten Zufallszahlen. *Metrika* **1964**, *8*, 5–15. [CrossRef]
42. Forsythe, G.E. Von Neumann's Comparison Method for Random Sampling from the Normal and Other Distributions. *Math. Comput.* **1972**, *26*, 817–826.
43. Ahrens, J.H.; Dieter, U. Extensions of Forsythe's method for random sampling from the normal distribution. *Math. Comput.* **1973**, *27*, 927–937.
44. Brent, R.P. Algorithm 488: A Gaussian pseudo random number generator. *Commun. ACM* **1974**, *17*, 704–706. [CrossRef]
45. Atkinson, A.C.; Pearce, M.C. The computer generation of Beta, Gamma and Normal Random Variables. *J. R. Stat. Soc. Ser. A* **1976**, *139*, 431–461. [CrossRef]
46. Ahrens, J.H.; Dieter, U. Computer methods for sampling from gamma, beta, poisson and binomial distributions. *Computing* **1974**, *12*, 223–246. [CrossRef]

47. Atkinson, A.C.; Whittaker, J. A Switching Algorithm for the Generation of Beta Random Variables with at Least One Parameter Less than 1. *J. R. Stat. Soc. Ser. A* **1976**, *139*, 462–467. [CrossRef]
48. Atkinson, A.C.; Whittaker, J. Algorithm AS 134: The Generation of Beta Random Variables with one Parameter Greater than and One Parameter Less than 1. *J. R. Stat. Soc. Ser. A* **1979**, *20*, 90–93. [CrossRef]
49. Cheng, R.C.H. Generating Beta Variates with Nonintegral Shape Parameters. *Commun. ACM* **1978**, *21*, 317–322. [CrossRef]
50. Schmeiser, B.W.; Shalaby, M.A. Acceptance Rejection Methods for Beta Variate Generation. *J. Am. Stat. Assoc.* **1980**, *75*, 673–678. [CrossRef]
51. Schmeiser, B.W.; Babu, A.J.G. Errata. *Oper. Res.* **1983**, *31*, 802.
52. Schmeiser, B.W. Generation of Variates from Distribution Tails. *Oper. Res.* **1980**, *28*, 1012–1017. [CrossRef]
53. Fox, B.L. Generation of Random Samples from the Beta F Distributions. *Technometrics* **1963**, *5*, 269–270. [CrossRef]
54. Floyd, R.W.; Rivest, R.L. Algorithm 489: The algorithm SELECT-for finding the ith smallest of n elements [M1]. *Commun. ACM* **1975**, *18*, 173. [CrossRef]
55. Singleton, R.C. Algorithm 347: An Efficient Algorithm for Sorting with Minimal Storage [M1]. *Commun. ACM* **1969**, *12*, 185–186. [CrossRef]
56. Sakasegawa, H. Stratified rejection and squeeze method for generating beta random numbers. *Ann. Inst. Stat. Math. Part B* **1983**, *35*, 291–302. [CrossRef]
57. Zechner, H.; Stadlober, E. Generating beta variates via patchwork rejection. *Computing* **1993**, *50*, 1–18. [CrossRef]
58. Kennedy, D.P. A note on stochastic search methods for global optimization. *Adv. Appl. Probab.* **1988**, *20*, 476–478. [CrossRef]
59. Hung, Y.C.; Balakrishnan, N.; Lin, Y.T. Evaluation of Beta Generation Algorithms. *Commun. Stat.-Simul. Comput.* **2009**, *38*, 750–770. [CrossRef]
60. Barabesi, L.; Pratelli, L. Universal methods for generating random variables with a given characteristic function. *J. Stat. Comput. Simul.* **2014**, *85*, 1679–169. [CrossRef]
61. Devroye, L. On the computer generation of random variables with a given characteristic function. *Comput. Math. Appl.* **1981**, *7*, 547–552. [CrossRef]
62. Smarandache, F. *Introduction to Neutrosophic Statistics*; Sitech & Education Publishing: New York, NY, USA, 2014. [CrossRef]
63. Jdid, M.; Nabeeh, N.A. Generating Random Variables that follow the Beta Distribution Using the Neutrosophic Acceptance-Rejection Method. *Neutrosophic Sets Syst.* **2023**, *58*, 139–147.
64. R Core Team. *R: A Language and Environment for Statistical Computing*; R Foundation for Statistical Computing: Vienna, Austria, 2021.