Simulating a coin with irrational bias using rational arithmetic

Luis Mendo
Universidad Politécnica de Madrid
luis.mendo@upm.es

November 8, 2024

Abstract

An algorithm is presented that, taking independent Bernoulli random variables with parameter 1/2 as inputs and using only rational arithmetic, simulates a Bernoulli random variable with possibly irrational parameter \( \tau \). It requires a series representation of \( \tau \) with positive, rational terms, and a rational bound on its truncation error that converges to 0. The number of required inputs has an exponentially bounded tail, and its mean is at most 3. The number of arithmetic operations has a tail that can be bounded in terms of the sequence of truncation error bounds.

The algorithm is applied to two specific values of \( \tau \), including Euler’s constant, for which obtaining a simple simulation algorithm was an open problem.

Keywords: Simulation, Random number generation, Bernoulli random variables, Rational arithmetic, Series.

MSC2010: 65C10, 65C50.

1 Introduction

Consider the problem of generating a Bernoulli random variable \( Y \) with known parameter \( \tau \in (0, 1) \). The only source of randomness is a sequence of independent, identically distributed Bernoulli variables \( X_k \) with parameter 1/2, and only rational-number computations can be performed. A sequential algorithm will be used, whereby the number of consumed inputs is random, and is governed by a certain stopping rule.

The enounced problem has applications in random number generation and simulation. In the following, independent Bernoulli random variables with parameter \( p \) will be referred to as \( p \)-coins”, or “unbiased coins” if \( p = 1/2 \). The restriction to use unbiased coins as inputs is a natural one, as these constitute the most “basic” form of randomness. Unbiased coins can easily be obtained
from $p$-coins, even if $p$ is unknown, using the well known von Neumann’s procedure [18] or its refinement by Peres [16]. Requiring only rational arithmetic operations has obvious advantages in terms of simplicity and precision. Clearly, rational arithmetic can be reduced to integer arithmetic by storing each rational number $x = n/d$ as $n$ and $d$ separately, and implementing operations on $x = n/d$ and $x' = n'/d'$ by means of integer arithmetic with $n, d, n$ and $d'$.

The stated problem is easy when $\tau$ is a rational number $n/d$. Let $b$ be the number of digits in the binary expression of $d$. Then it suffices to sample $b$ unbiased coins, interpret the result as an integer $t \in \{0, 1, \ldots, 2^b - 1\}$, repeat until $t \leq d - 1$, and then output 1 if $t \leq n - 1$ or 0 otherwise. The general problem when $\tau$ is not necessarily rational is more interesting.

An algorithm for producing a $\tau$-coin from unbiased coins is termed a Buffon machine in [7]. This is related to the Bernoulli factory problem [10], which consists in generating an $f(p)$-coin from $p$-coins, when the function $f$ is known and $p$ is unknown. This problem has been extensively studied, and it is known how characteristics of algorithms that can solve it are restricted by properties of the function [10, 14]. Efficient algorithms are known for several classes of functions [12, 9, 13].

This paper describes a general algorithm that solves the above problem when there exists a representation of $\tau$ as a series with rational terms. The algorithm is described in §2, and its basic properties are addressed. The complexity of the algorithm is analysed in §3. Application to specific values, including Euler’s constant $\gamma$ and $\pi/4$, is discussed in §4. Conclusions and future work are presented in §5. Proofs to all results are given in §6.

The following notation and definitions are used. A random variable $V$ is referred to as a shifted geometric variable if $V - 1$ is geometric. Given two non-negative functions $f$ and $g$, $f(x)$ is said to be $O(g(x))$ if there exist $K$ and $x_0$ such that $f(x) \leq Kg(x)$ for all $x \geq x_0$. The natural and binary logarithms of $x$ are written as $\ln x$ and $\log_2 x$ respectively. The number of digits in the binary expression of a positive integer $t$ is denoted as $B(t)$. For $n \in \mathbb{N}$, the notation $n!!$ represents the double factorial, that is, $n(n - 2) \cdots 3 \cdot 1$ for $n$ odd and $n(n - 2) \cdots 4 \cdot 2$ for $n$ even.

### 2 Algorithm description and basic properties

The proposed algorithm is inspired by [12, algorithm 2], which uses a continuous uniform random variable $U$ on $(0, 1)$, a sequence of monotonically increasing lower bounds $\lambda_k$ that converge to $\tau$, and a sequence of monotonically decreasing upper bounds $\mu_k$ that converge to $\tau$. The algorithm in the cited reference generates $U$ and then computes $\lambda_k, \mu_k$ for successive $k$ until $U \leq \lambda_k$ or $U > \mu_k$ (this occurs for finite $k$ with probability 1). The output $Y$ is then 1 if $U \leq \lambda_k$ and 0 if $U > \mu_k$ for the last $k$.

Generating a continuous uniform random variable in a computer simulation poses numerical precision problems, and is not compatible with using ra-
tional arithmetic. The reason is that computer simulations typically represent non-integer numerical values using floating-point data types, which implies that the number of significant digits that can be used is limited to a fixed value. Instead, the algorithm to be presented does not generate \( U \) explicitly, but only the information about it which is necessary at each iteration \( k \) to determine if \( U \) is below \( \lambda_k \), above \( \mu_k \), or between the two bounds. This avoids the loss of accuracy that would be incurred when trying to represent the exact value of \( U \).

Let \( \tau \in (0, 1) \) be represented as a convergent series with positive, rational terms \( a_j \):

\[
\tau = \sum_{j=1}^{\infty} a_j. \quad (1)
\]

Series expressions of this form are available for the majority of commonly used constants. In addition, a bound \( \varepsilon(N) \) for the truncation error is assumed to be known, and to be computable with operations involving only rational numbers:

\[
\tau - \sum_{j=1}^{N} a_j \leq \varepsilon(N), \quad (2)
\]

where \( \varepsilon(N) \) is monotonically non-increasing with \( \lim_{N \to \infty} \varepsilon(N) = 0 \). The function \( \varepsilon \) will be referred to as the *error function*.

An alternating series

\[
\tau = \sum_{j=1}^{\infty} (-1)^{j+1} b_j \quad (3)
\]

with terms that decrease monotonically in absolute value can be rewritten in the form (1) with \( a_j = b_{2j-1} - b_{2j} \). If \( \lim_{j \to \infty} b_j = 0 \), Leibniz’s rule [4, theorem 10.14] implies that the series converges, and \( \tau - \sum_{j=1}^{N} a_j \leq b_{2N+1} \). Thus a simple characterization of the truncation error is possible in this case, namely \( \varepsilon(N) = b_{2N+1} \). Series with negative terms or with alternating signs opposite from those in (3) are reduced to the preceding cases by considering \( 1 - \tau \) instead of \( \tau \); and then it suffices to replace the algorithm output \( Y \) by \( Y' = 1 - Y \) to achieve \( \Pr[Y' = 1] = \tau \).

The monotonicity requirement for the error function does not impose any restriction, because any error function can be modified to fulfil this condition, simply replacing \( \varepsilon(n) \) by its cumulative minimum \( \varepsilon'(n) = \min \{ \varepsilon(1), \ldots, \varepsilon(n) \} \). This can be done because, since the series has positive terms, the error bound \( \varepsilon(i) \) is valid not only for \( \sum_{j=1}^{i} a_j \), but also for any \( \sum_{j=1}^{n} a_j \) with \( n > i \). As will be seen, the algorithm to be presented uses the error bound for the sum with \( n \) terms after it has already used the error bound for the sum with \( n - 1 \) terms.

Thus the cumulative minimum \( \varepsilon'(n) \) can be efficiently obtained as

\[
\varepsilon'(n) = \begin{cases} 
\varepsilon(n) & \text{if } n = 1 \\
\min \{ \varepsilon'(n-1), \varepsilon(n) \} & \text{if } n \geq 2.
\end{cases} \quad (4)
\]

The proposed algorithm consists of a random number \( M \) of iterations. At the beginning of iteration \( k \), the continuous uniform variable \( U \) is known to be
in an interval \((\lambda_{k-1}, \mu_{k-1})\) resulting from the previous iteration. The iteration shrinks this interval to a new interval \((\lambda_k, \mu_k) \subset (\lambda_{k-1}, \mu_{k-1})\). These intervals are quantized, with finer resolution as the algorithm progresses. Specifically, the endpoints of the interval \((\lambda_k, \mu_k)\) are multiples of \(2^{-(k+1)}\), and the length of the interval is \(2^{-k}\). Thus each quantized interval is half as wide as the one from the preceding iteration. The shrinking and quantizing conditions leave only three possible choices for the interval \((\lambda_k, \mu_k)\) given \((\lambda_{k-1}, \mu_{k-1})\):

\[
\lambda_k = \lambda_{k-1} + s_k \cdot 2^{-(k+1)}, \quad \mu_k = \lambda_k + 2^{-k}, \quad s_k \in \{0, 1, 2\}.
\]  

Thus \((\lambda_k, \mu_k)\) is the lower half, the middle half or the upper half of \((\lambda_{k-1}, \mu_{k-1})\) for \(s_k = 0, 1\) or 2 respectively (see Figure 1 below).

The choice of \(s_k\) is dictated by intermediate, unquantized bounds \(\tilde{\lambda}_k\) and \(\tilde{\mu}_k\) computed from the series representation of \(\tau\):

\[
\tilde{\lambda}_k = \sum_{j=1}^{N_k} a_j, \quad \tilde{\mu}_k = \tilde{\lambda}_k + \varepsilon(N_k),
\]

which define an unquantized interval \((\tilde{\lambda}_k, \tilde{\mu}_k)\) that also shrinks at each iteration. More specifically, knowing \(N_{k-1}\), \(\tilde{\lambda}_{k-1}\) and \(\tilde{\mu}_{k-1}\) from the previous iteration, the new bounds \(\tilde{\lambda}_k\) and \(\tilde{\mu}_k\) are obtained adding series terms up to a certain index \(N_k \geq N_{k-1}\):

\[
\tilde{\lambda}_k = \tilde{\lambda}_{k-1} + \sum_{j=N_{k-1}+1}^{N_k} a_j,
\]

and computing the corresponding truncation error \(\varepsilon(N_k)\) to be used in (7). The number of terms \(N_k\) at iteration \(k\) is chosen as the smallest value such that \((\tilde{\lambda}_k, \tilde{\mu}_k) \cap (\lambda_{k-1}, \mu_{k-1})\) is contained in one of the three possible quantized intervals \((\lambda_k, \mu_k)\) defined by (5), which also determines the choice of \(s_k\). The reason for this is that if \(\tau\) is in \((\tilde{\lambda}_k, \tilde{\mu}_k)\) and this interval is contained in one of the three quantized intervals, \(\tau\) is assured to be contained in that quantized interval. Besides, if part of the interval \((\tilde{\lambda}_k, \tilde{\mu}_k)\) exceeds the boundaries of \((\lambda_{k-1}, \mu_{k-1})\) that part can be disregarded (only the intersection matters), because \(\tau\) is known not to be outside \((\lambda_{k-1}, \mu_{k-1})\). Note that both sequences \(N_k\) and \(s_k\) are deterministic. Figure 1 illustrates the steps involved in moving from \((\lambda_{k-1}, \mu_{k-1})\) to \((\lambda_k, \mu_k)\).

According to the above, the rules for selecting \(N_k\) and \(s_k\) at iteration \(k\) are:

\[
\text{if } \tilde{\mu}_k \leq \tilde{\lambda}_{k-1} + 2^{-k} \Rightarrow s_k = 0; \\
\text{elseif } \tilde{\lambda}_k > \tilde{\lambda}_{k-1} + 2^{-k} \Rightarrow s_k = 2; \\
\text{elseif } \tilde{\lambda}_k > \tilde{\lambda}_{k-1} + \frac{1}{2} \cdot 2^{-k} \text{ and } \tilde{\mu}_k \leq \tilde{\lambda}_{k-1} + \frac{3}{2} \cdot 2^{-k} \Rightarrow s_k = 1; \\
\text{else a larger } N_k \text{ is needed.}
\]

(9)
$$s_k = 1$$

$$s_{k-1} = 2$$

$$\lambda_{k-1}$$

$$\mu_{k-1}$$

$$\lambda_k$$

$$\mu_k$$

$$2^{-(k+1)}$$

$$2^{-k}$$

$$\lambda_{k-1}$$

$$\mu_{k-1}$$

$$\lambda_k$$

$$\mu_k$$

Iteration $k-1$

Iteration $k$

(a) Case $(\tilde{\lambda}_k, \tilde{\mu}_k) \subset (\lambda_{k-1}, \mu_{k-1})$

(b) Case $(\tilde{\lambda}_k, \tilde{\mu}_k) \not\subset (\lambda_{k-1}, \mu_{k-1})$

Figure 1: Steps in moving from iteration $k-1$ to $k$
Thus, starting from the partial sum $\tilde{\lambda}_{k-1}$ from the previous iteration, which contains $N_{k-1}$ terms, new terms are added one by one, computing tentative values of $\tilde{\lambda}_k$ and $\tilde{\mu}_k$ for each new partial sum, checking conditions (9), and stopping as soon as one of the three conditions holds.

Once the new quantized interval $(\lambda_k, \mu_k]$ has been obtained in iteration $k$, a random input $X_k$ is used to decide if $U$ is in $(\lambda_k, \mu_k]$. Specifically, the algorithm uses initial values $\tilde{\lambda}_0 = 0$, $\tilde{\mu}_0 = 1$ and $(\tilde{\lambda}_0, \tilde{\mu}_0) = (0, 1]$. At this point, $U$ is only known to be in this interval. The algorithm proceeds with the first $N$ values of $\tilde{\lambda}_k$ and $\tilde{\mu}_k$ for which the random input $X \tau$ reduces the interval in which $X \tau$ is known to be to a new interval with half the size, that are constructed by the algorithm based on the partial sums and error bounds of the series.

Denoting the total number of required inputs by $M$, depending on whether one last input is needed to generate the output or not.

As is apparent from the foregoing description, although the continuous variable $U$ is helpful for explaining the process, its exact value is not actually needed, and is never generated. $U$ is only known to be in intervals of decreasing size, that are constructed by the algorithm based on the partial sums and error bounds of the series.

The described procedure can be compared with that given in [7, section 1] to simulate a rational constant $\tau \in (0, 1)$: using the binary representation of $\tau$, which in the rational case is completely known (it is either finite or repeating), output the $i$-th binary digit where $i$ is given by a shifted geometric random variable with parameter $1/2$. To extend this approach for irrational $\tau$ its binary
representation, which is not fully known, would have to be computed up to the \(i\)-th digit. That is similar to what the algorithm presented here does: it computes increasingly accurate approximations of \(\tau\) until a sufficient level of accuracy, given by the shifted geometric random variable \(M\), is achieved.

Another related approach is the “interval algorithm” in [8], which in general transforms a discrete, finite-support distribution into another, both distributions being known. Particularized to an unbiased coin as input and a biased coin with parameter \(\tau\) as output, the referred algorithm iteratively replaces an interval, initialised as \([0, 1]\), by either its lower or upper half, as indicated by a random input, until the interval no longer contains \(\tau\). The main differences with the algorithm described in this paper are that [8] uses disjoint subintervals and assumes \(\tau\) to be known exactly. Here, in contrast, overlapping subintervals are used, and \(\tau\) needs not be known exactly (which allows using rational arithmetic).

Based on the above the algorithm can be precisely stated; see Algorithm 1. Its inputs are:

1. A series with positive terms \(a_j\), as given by (1), that converges to the target value \(\tau\).
2. A function that computes a bound \(\varepsilon(N)\) for the error of approximating the series with the first \(N\) terms, as given by (2), with \(\lim_{N \to \infty} \varepsilon(N) = 0\). If necessary, its cumulative minimum should be taken in order to make the function monotonically non-increasing.
3. A sequence of independent Bernoulli random variables \(X_k\) with parameter 1/2.

The output is a Bernoulli random variable \(Y\) with parameter \(\tau\).

Algorithm 1 is seen to consist of three parts: initialisation of variables (first two lines), iterations (main loop: “repeat”), and output generation (final “if” block). The inner loop (“while”) updates the partial sum of the series and its error bound. A difference from the description in the preceding paragraphs is that sequences of variables such as \(N_k, \tilde{\lambda}_k, \lambda_k, s_k\), which were indexed by the iteration number \(k\), are here expressed more compactly by single variables \(N, \tilde{\lambda}, \lambda, \) and \(s\) that are updated at each iteration. Similarly, the variable \(\varepsilon\) stores the value of the latest error bound, \(\varepsilon(N_k)\); and \(k\) is the current iteration index (which defines resolution).

**Theorem 1** (Basic properties). Algorithm 1 satisfies the following:

1. The algorithm terminates with probability 1.
2. The output \(Y\) is a Bernoulli random variable with parameter \(\tau\).
3. If each term \(a_j\) in the series (1) and the error bound \(\varepsilon(N)\) defined by (2) can be computed with a finite number of operations involving rational numbers, the algorithm can be implemented using rational arithmetic.

7
Algorithm 1 Simulation of a constant using unbiased coins

\[ N \leftarrow 0, \tilde{\lambda} \leftarrow 0, \epsilon \leftarrow 1 \]
\[ \lambda \leftarrow 0, s \leftarrow 0, k \leftarrow 0 \]

**repeat**
\[ k \leftarrow k + 1 \]
\[ \lambda \leftarrow \lambda + s \cdot 2^{-k} \]

**while not** \( \left( \tilde{\lambda} + \epsilon \leq \lambda + 2^{-k} \text{ or } \tilde{\lambda} > \lambda + 2^{-k} \right) \)
\[ \left( \tilde{\lambda} > \lambda + (1/2) \cdot 2^{-k} \text{ and } \tilde{\lambda} + \epsilon \leq \lambda + (3/2) \cdot 2^{-k} \right) \]
\[ \text{do} \]
\[ N \leftarrow N + 1 \]
\[ \tilde{\lambda} \leftarrow \tilde{\lambda} + a_N \]
\[ \epsilon \leftarrow \epsilon(N) \]
**end while**

**if** \( \tilde{\lambda} + \epsilon \leq \lambda + 2^{-k} \)**
\[ s \leftarrow 0 \]
**else if** \( \tilde{\lambda} > \lambda + 2^{-k} \)
\[ s \leftarrow 2 \]
**else**
\[ s \leftarrow 1 \]
**end if**

**until** \( X_k = 0 \)

**if** \( s = 0 \)**
\[ Y \leftarrow 0 \]
**else if** \( s = 2 \)
\[ Y \leftarrow 1 \]
**else**
\[ Y \leftarrow X_{k+1} \]
**end if**
3 Complexity analysis

The complexity of a sequential algorithm is primarily determined by the number of required inputs, because consuming a new input is typically considered more costly than the arithmetical operations needed to process it. However, it is also important to analyse the number of required arithmetical operations, to ensure that it is not unrealistically large. In this regard, assessing its order of magnitude is usually enough.

Algorithm 1 consumes one input for each iteration, and possibly one additional input to generate the output. As for arithmetical operations, the algorithm uses two groups thereof, from the point of view of a complexity analysis:

1. Operations that are carried out to obtain each new term of the series, and to update the partial sum and error bound. The total number of these operations is roughly proportional to the total number of terms in the partial sum when the algorithm ends, $N_M$.

2. Operations that are needed to update the variables used by the algorithm. The total number of these is proportional to the number of iterations, $M$.

From the preceding it is clear that a good characterization of algorithm complexity can be obtained by analysing the number of iterations $M$, the number of inputs $L$, and the number of series terms when the algorithm terminates, $N_M$. The first two are characterized very easily, because the distribution of $M$ is known.

An algorithm that uses $L$ inputs is fast, as defined by Nacu and Peres [14], if $L$ is exponentially bounded, that is, if there exist $C > 0$, $\rho < 1$ such that for all $l \in \mathbb{N}$

$$\Pr[L > l] \leq C\rho^l.$$  \hfill (11)

**Theorem 2** (Number of inputs). In Algorithm 1, the number of required inputs satisfies

$$\Pr[L > l] \leq 2^{-l+1}$$  \hfill (12)

for $l \in \mathbb{N}$, and thus the algorithm is fast in the sense of Nacu-Peres. In addition,

$$2 \leq \mathbb{E}[L] \leq 3.$$  \hfill (13)

Thus, according to this theorem, the number of inputs used by the algorithm has an exponentially bounded tail and its average value is very small. Indeed, [11, theorem 6] establishes that any algorithm that outputs a Bernoulli random variable with parameter $\tau$ from inputs with parameter $1/2$ must use at least 2 inputs on average, except when $\tau$ is a dyadic number. Therefore the average number of inputs consumed by Algorithm 1 is close to the optimum. Moreover, even if the number of required inputs in a given realization of the algorithm can potentially be much larger than its average value, the probability that this happens is very small thanks to the exponential-bound property.
The distribution of \( N_M \) is more difficult to characterize, because it depends on the error function \( \varepsilon \): the more slowly this function decreases, the more likely it is for \( N_M \) to take larger values. However, as has been discussed, \( N_M \) only affects the number of required arithmetical operations and thus it suffices to know its order or magnitude. The following result establishes a bound on \( \Pr[N_M > n] \), and gives a sufficient condition for \( \mathbb{E}[N_M] \) to be finite.

**Theorem 3 (Number of series terms).** The number of series terms used by Algorithm 1 satisfies

\[
\Pr[N_M > n] < 4\varepsilon(n)
\]

for \( n \in \mathbb{N} \), where \( \varepsilon \) is the error function. In addition, if \( \varepsilon(n) = O(1/n^r) \) for some \( r > 1 \), \( \mathbb{E}[N_M] \) is finite.

This theorem describes how the truncation error bound influences the number of required arithmetical operations: namely, \( \Pr[N_M > n] = O(\varepsilon(n)) \). Furthermore, \( \varepsilon(n) \) asymptotically decreasing as the inverse of a power with exponent greater than 1 is sufficient to ensure that \( \mathbb{E}[N_M] \) is finite. This requirement on \( \varepsilon \) is not very stringent. The two detailed examples to be presented in §4 will satisfy this condition (and \( \mathbb{E}[N_M] \) will be seen to be not only finite but very small).

A conceivable modification of Algorithm 1 would be to only allow cases \( s_k = 0 \) and \( s_k = 2 \) in each iteration (like [8] does). With this approach the third condition in (9) is eliminated, and series terms are added until the interval \( (\bar{\lambda}_k, \bar{\mu}_k) \cap (\lambda_{k-1}, \mu_{k-1}] \) is contained either in the lower half or in the upper half of \( (\lambda_{k-1}, \mu_{k-1}] \). This way the final iteration never requires an additional input to generate the output. The problem with this method is that, depending on the value of \( \tau \), shrinking \( (\bar{\lambda}_k, \bar{\mu}_k] \) until it fits into one of the two halves of the previous quantized interval may require an arbitrarily large number of series terms. Furthermore, if \( \tau \) is a dyadic number there is a non-zero probability that the algorithm does not terminate. Algorithm 1 increases the average number of inputs from the optimum 2 to at most 3, but in return it terminates with probability 1 and \( \Pr[N_M > n] \) is bounded. (Note that the procedure in [8] always terminates, but assumes perfect knowledge of \( \tau \)).

## 4 Application

Two specific cases will be considered in detail: Euler’s constant \( \gamma \) (§4.1), and \( \pi/4 \) (§4.2). These are in themselves interesting; especially the former, as simulating \( \gamma \) without real-number arithmetic is one of the open problems mentioned in [7]. In addition, they illustrate the algorithm’s performance in two different situations: a series of positive terms with error decaying as a power law, and an alternating series with exponential error decay. A few additional examples are then briefly discussed (§4.3).
4.1 Simulation of $\gamma$

Euler’s constant is defined as $\gamma = \lim_{n \to \infty} \left( -\ln n + \sum_{i=1}^{n} 1/i \right) = 0.5772156\ldots$

Many series are known that converge to $\gamma$; see for example [3, 17, 6]. The following one [3, 17] is of interest for application of Algorithm 1:

$$\gamma = \frac{1}{2} + \sum_{j=1}^{\infty} \frac{B(j)}{2j(2j+1)(2j+2)}$$

(15)

where $B(n)$ is the number of binary digits of the positive integer $n$, as defined in §1. This can be computed using only integer operations, namely successive values $b = 1, 2, \ldots$ are tried, and the output is the first $b$ such that $2^b > n$.

The series (15) can be rewritten in the form (1) with

$$a_j = \begin{cases} 
1/2 & \text{if } j = 1 \\
\frac{B(j-1)}{2j(2j-1)(2j-2)} & \text{if } j \geq 2.
\end{cases}$$

(16)

A rational bound can easily be obtained for the truncation error of this series.

**Proposition 1.** For $N \geq 2$, the series defined by (16) satisfies $\gamma - \sum_{j=1}^{N} a_j < \varepsilon(N)$ with

$$\varepsilon(N) = \frac{2 + B(N-1) + 1/(N-1)}{16(N-1)^2}. \tag{17}$$

This error bound clearly converges to 0, but it is not monotonic, due to the term $B(N-1)$ in the numerator. Namely, $B(2t) = B(2t-1) + 1$ for any positive integer $t$, which can cause (17) to increase when $N$ changes from $2^t$ to $2^t + 1$. Indeed, it can be seen that $\varepsilon(N+1) > \varepsilon(N)$ for $N = 2^4, 2^5, 2^6, \ldots$ and $\varepsilon(N+1) < \varepsilon(N)$ otherwise. Nonetheless, as discussed in §2, monotonicity can be achieved by redefining the error function as

$$\varepsilon(N) = \begin{cases} 
1/2 & \text{if } N = 1 \\
\min \left\{ \varepsilon(N-1), \frac{2 + B(N-1) + 1/(N-1)}{16(N-1)^2} \right\} & \text{if } N \geq 2.
\end{cases} \tag{18}$$

In addition, this function is easily shown to be $O(1/n^r)$ for any $r < 2$.

Thus the series defined by (16) and its truncation error bound (18) satisfy all the requirements in §2 and §3 (series with positive rational terms that can be computed easily; error bound that is monotonically non-increasing, tends to 0, and is $O(1/n^r)$ for some $r > 1$). Therefore Algorithm 1 can be applied to (16) and (18), and the results in Theorems 1–3 hold.

Table 1 shows the sample mean of $Y$, $L$ and $N_M$ obtained from running the algorithm $10^8$ times. The sample mean of $Y$ differs from $\gamma$ by 0.000027. This difference is comparable to the standard deviation of the average of $10^8$ Bernoulli variables with parameter $\gamma$, which is $(\gamma(1-\gamma)/10^8)^{1/2} = 0.000049.$
The average number of inputs is only slightly greater than 2, and therefore very close to the optimum.

The average number of series terms is also seen to be very small. In view of (16) and (18), updating the partial sum of the series with a new term and computing the corresponding error bound requires around 20 arithmetical operations (the exact number of operations depends on implementation details such as whether intermediate results are stored; note also that the first terms, which are needed with higher probability, require fewer operations). Updating the algorithm variables in each iteration adds a small amount of computational burden. Therefore a simulation requires an average number of arithmetical operations of the order of several tens.

Figures 2 and 3 depict the sample estimates of \( \Pr[L > l] \) and \( \Pr[N_M > n] \), together with their respective bounds \( 2^{-l+1} \) and \( 4\epsilon(n) \), for \( l, n \in \mathbb{N} \). The actual value (filled circle) is always below or at the same height as the corresponding bound (empty circle). \( \Pr[L > l] \) equals its bound whenever \( s_l = 1 \), or half its bound otherwise. Observe how \( \Pr[N_M > n] \) consists of runs of equal values, caused by the fact that \( N_M \) cannot take any value between \( N_k \) and \( N_{k+1} \). Also, the bound of \( \Pr[N_M > n] \) has short runs of equal values near all powers of 2 except for the smallest ones, corresponding to an increase in the original bound (17).

Some optimization is possible if the algorithm is to be repeatedly applied
4.2 Simulation of $\pi/4$

Rational multiples of $\pi$ are a classical example for the simulation of Bernoulli random variables. Consider Euler’s Machin-like formula [15],

$$\frac{\pi}{4} = \arctan \frac{1}{2} + \arctan \frac{1}{3},$$

(19)

The Taylor expansion of the arctan function [2, section 4.4],

$$\arctan x = \sum_{j=1}^{\infty} \frac{(-1)^{j+1}}{2j-1} x^{2j-1},$$

(20)
is alternating with terms that monotonically decrease in absolute value. Therefore, as discussed in §2, it can be expressed as a series of positive terms,

$$\arctan x = \sum_{j=1}^{\infty} \left( \frac{x^{4j-3}}{4j-3} \frac{x^{4j-1}}{4j-1} \right),$$

(21)

and the sum of the first $N$ terms has an error bounded by $x^{4N+1}/(4N + 1)$.

Using this into (19) yields the representation $\pi/4 = \sum_{j=1}^{\infty} a_j$ with

$$a_j = \frac{2^{-4j+3} + 3^{-4j+3} - 2^{-4j+1} + 3^{-4j+1}}{4j-3} - \frac{2^{-4j+3} + 3^{-4j+3} - 2^{-4j+1} + 3^{-4j+1}}{4j-1},$$

(22)

where $a_j$ is positive and rational; and $\pi/4 - \sum_{j=1}^{N} a_j < \varepsilon(N)$ for $N \geq 1$ with

$$\varepsilon(N) = \frac{2^{-4N-1} + 3^{-4N-1}}{4N + 1},$$

(23)

which is rational and monotonically decreasing with $\lim_{N \to \infty} \varepsilon(N) = 0$.

Based on the above, Algorithm 1 can be applied to (22) and (23). The results for $10^8$ simulations are presented in Table 2 and in Figures 4 and 5. The average number of inputs is again slightly larger than the optimum 2.

The convergence of the series is much faster in this case than in §4.1 (exponential instead or inverse power law), which translates into smaller values of $N_M$. In fact, the maximum value observed in the $10^8$ simulations is $N_M = 6$, which only occurs in 3 cases (thus the sample estimate of $\Pr[N_M > 5]$ is not reliable, and is not plotted in Figure 5). In view of (22) and (23), the number of arithmetical operations needed for each new term of the series and for the corresponding error bound is small, and a simulation requires an average number of operations of the order of a few tens.

### 4.3 Other examples

The following briefly discusses how the proposed algorithm can be applied to a few other specific values of $\tau$.

- $\tau = 1/\sqrt{2}$: using the Taylor expansion of $1/\sqrt{x}$ about $x = 1$, the value $1/\sqrt{2}$ can be expressed as in (3) with

$$b_j = \begin{cases} 
1 & \text{if } j = 1 \\
(2j-3)!! & \text{if } j \geq 2.
\end{cases}$$

(24)
Figure 4: Sample estimate of $\Pr[L > l]$ for simulation of $\pi/4$

Figure 5: Sample estimate of $\Pr[N_M > n]$ for simulation of $\pi/4$
Making use of the technique described in §2, this can be rewritten as a series with positive, rational terms, and a rational bound for the truncation error is easily obtained. Therefore Algorithm 1 is applicable.

- $\tau = 1/e$: similarly, the Taylor expansion of $e^{-x}$ about $x = 0$ yields an alternating series for $1/e$ of the form (3) with

$$b_j = \frac{1}{(j-1)!},$$  \hspace{1cm} (25)

to which an analogous procedure can be applied.

- $\tau = 1/\pi$: in this case the series for $2/\pi$ given by [5, equation (10.1)] can be employed. Adapting it for $\tau = 1/\pi$, this corresponds to (3) with

$$b_j = \begin{cases} 
1/2 & \text{if } j = 1 \\
\frac{4}{2} (\frac{(2j-3)!!}{(2j-2)!!})^3 & \text{if } j \geq 2,
\end{cases} \hspace{1cm} (26)
$$

which is used similarly.

- $\tau = 1/(\sqrt{2}\pi)$: a well-known series found by Ramanujan [5, equation (2.1)] can be put in the form (1) with $\tau = 1/(\sqrt{2}\pi)$ and

$$a_j = 19602 \frac{(4j-4)!}{4^j (j-4)(j-1)!^4} \frac{26390j - 25287}{99^j},$$  \hspace{1cm} (27)

where $a_j$ is rational. The first fraction in (27) is less than 1, and thus the truncation error can be bounded as

$$\sum_{j=N+1}^{\infty} a_j < \varepsilon(N) = 19602 \sum_{j=N+1}^{\infty} \frac{26390j - 25287}{99^j}. \hspace{1cm} (28)$$

An explicit, rational expression for $\varepsilon(N)$ is obtained from (28) making use of the identities

$$\sum_{j=1}^{\infty} q^{j-1} = \frac{1}{1-q}, \hspace{1cm} (29)$$

$$\sum_{j=1}^{\infty} jq^{j-1} = \frac{1}{(1-q)^2}, \hspace{1cm} (30)$$

and then Algorithm 1 can be applied.

### 5 Conclusions and future work

An algorithm has been proposed that generates a Bernoulli random variable of arbitrary parameter $\tau$, using a sequence of independent Bernoulli variables of
parameter $1/2$ as input. The algorithm requires a positive series representation of $\tau$, and a bound for the truncation error that converges to 0. If the series terms and the error bound are rational, the algorithm can be implemented using only rational arithmetic.

Standard simulation methods, as implemented in computer software, typically define the parameter $\tau$ as a floating-point value. This inevitably incurs a loss of precision. More specifically, it is not possible to represent irrational values (or even certain rational values) exactly using floating-point variables. The method presented in this paper avoids that problem, and generates a random variable with the exact parameter $\tau$.

The algorithm consumes a random number of inputs $L$ with $2 \leq E[L] \leq 3$. Thus $E[L]$ is close to the optimum achievable by any algorithm, which is 2. In addition, the algorithm is fast in the sense of Nacu-Peres (that is, $\Pr[L > l]$ has an exponential bound). The number of series terms that need to be computed, $N_M$, is also random, and $\Pr[N_M > n]$ decreases with $n$ at least as fast as the truncation error bound. $E[N_M]$ is finite if $\Pr[N_M > n]$ is $O(1/n^r)$, $r > 1$.

The algorithm has been applied to the simulation of $\gamma$ and of $\pi/4$. The former solves an open question in [7], and establishes that $\gamma$ can be simulated using rational arithmetic only. In both cases the average numbers of consumed inputs and of required operations are very small.

As future work, an interesting extension would be to allow the inputs to be biased coins, with an arbitrary and possibly unknown parameter $p$. A straightforward approach for this problem is to transform the input $p$-coins into $1/2$-coins and then apply the algorithm presented here. With $p$ unknown, the average number of $1/2$-coins that can be obtained per $p$-coin is known to be at most $-\log_2 p - \log_2(1 - p)$, and a rate arbitrarily close to this can be achieved using Peres’ iterative version of von Neumann’s procedure [16]. Consequently, the average number of $p$-coin inputs needed to generate a $\tau$-coin using this approach can be roughly approximated by $E[L]/(-\log_2 p - \log_2(1 - p))$, with $E[L]$ as resulting from Algorithm 1 (and bounded by Theorem 2). Perhaps a more efficient method can be found.

6 Proofs

6.1 Proof of Theorem 1

1. Since the number of iterations $M$ is a shifted geometric random variable, it is finite with probability 1.

2. The algorithm outputs $Y = 1$ if $U < \tau$ and $Y = 0$ if $U > \tau$, where $U$ is uniform on the interval $(0, 1]$. Therefore $\Pr[Y = 1] = \Pr[U < \tau] = \tau$ and $\Pr[Y = 0] = \Pr[U > \tau] = 1 - \tau$ (the event $U = \tau$ has probability 0).

3. Under the stated assumptions, all variables used by the algorithm are obtained as a finite sequence of additions, multiplications or divisions of rational numbers, or powers of rational numbers with integer exponents.
6.2 Proof of Theorem 2

The number of inputs, $L$, satisfies

$$M \leq L \leq M + 1.$$  \hfill (31)

For $l \geq 1$, using (10) and (31),

$$\Pr[L > l] \leq \Pr[M > l - 1] = \Pr[M \geq l] = 2^{-l+1}. \hfill (32)$$

As a consequence, (11) holds with $C = 2$, $\rho = 1/2$.

6.3 Proof of Theorem 3

The sequence formed by the numbers of series terms used in each iteration, $N_k$, is deterministic, and is dictated by the values of the terms $a_j$ and of the error function $\varepsilon(n)$. This sequence is monotonically non-decreasing. The total number of terms used by the algorithm is that corresponding to the last iteration, that is, $N_M$. This is a random variable, because $M$ is.

In general, the sequence $N_k$ consists of runs of equal values, where each run has length 1 or greater. Let $u(i)$ denote the initial index of the $i$-th run. Note that $N_{u(1)} = N_1$. Thus $N_{u(1)}, N_{u(2)}, \ldots$ is the subsequence of unique values of sequence $N_k$, with

$$N_{u(i-1)} = N_{u(i)} - 1 < N_{u(i)}.$$  \hfill (33)

The following observation is key to the proof. If the interval $[\tilde{\lambda}_k, \tilde{\mu}_k]$ were chosen such that $\tilde{\mu}_k - \tilde{\lambda}_k \leq 2^{-(k+1)}$, this would guarantee that at least one of the conditions (9) would hold (see Figure 1, and observe that $2^{-(k+1)}$ is $1/4$ of the length of the previous quantized interval $(\lambda_{k-1}, \mu_{k-1})$).

Let the sequence $N_k$ and the error function $\varepsilon$ be extended by defining $N_0 = 0$, $\varepsilon(0) = 1$. Consider $k = u(i)$ for $i \in \mathbb{N}$ arbitrary; that is, the index $k$ starts a run of equal values of the sequence $N_k$. This means that, at iteration $k$, the previous $N_{k-1}$ was not enough to fulfill any of conditions (9), and new series terms up to index $N_k$ had to be added. In particular, $N_k - 1$ terms were not sufficient to fulfill (9). Taking into account the observation in the preceding paragraph, it follows that $\varepsilon(N_k - 1)$ is necessarily greater than $2^{-(k+1)}$; that is,

$$\varepsilon(N_{u(i)} - 1) > 2^{-(u(i)+1)}. \hfill (34)$$

This holds even if $i = 1$ and $N_1 = 1$, thanks to the definitions of $N_0$ and $\varepsilon(0)$.

Since $u(i)$ is the starting index of its run, $\Pr[N_M \geq N_{u(i)}]$ can be bounded from (10) and (34) as

$$\Pr[N_M \geq N_{u(i)}] = \Pr[M \geq u(i)] = 2^{-u(i)+1} < 4\varepsilon(N_{u(i)} - 1). \hfill (35)$$
The variable $N_M$ can only take the values $N_{u(1)}, N_{u(2)}, \ldots$, which form an increasing sequence. Therefore, $N_M$ cannot take any value between $N_{u(i-1)}$ and $N_{u(i)}$. Thus for $n \in \{N_{u(i-1)}, N_{u(i-1)} + 1, \ldots, N_{u(i)} - 1\}$ the event $N_M > n$ is equivalent to $N_M \geq N_{u(i)}$. Using the fact that $\varepsilon$ is non-increasing, (35) implies that, for the referred values of $n$,

$$\Pr[N_M > n] < 4\varepsilon(N_{u(i)} - 1) \leq 4\varepsilon(n).$$  \hfill (36)

As this is valid for $i \geq 1$, it follows that (36) holds for any $n \geq 1$.

If $\varepsilon(n)$ is $O(1/n^r)$ there exist $K$ and $n_0$ such that $\varepsilon(n) \leq K/n^r$ for all $n \geq n_0$. Therefore the mean of $N_M$ can be expressed as

$$E[N_M] = \sum_{n=1}^{\infty} \Pr[N_M \geq n] = \sum_{n=0}^{n_0-1} \Pr[N_M > n] + \sum_{n=n_0}^{\infty} \Pr[N_M > n]$$

$$< \sum_{n=0}^{n_0-1} \Pr[N_M > n] + 4 \sum_{n=n_0}^{\infty} \varepsilon(n)$$

$$\leq \sum_{n=0}^{n_0-1} \Pr[N_M > n] + 4K \sum_{n=n_0}^{\infty} \frac{1}{n^r}$$  \hfill (37)

For $r > 1$ the last term in (37) is a convergent series [1, corollary 2.4.7], and therefore $E[N_M]$ is finite. \hfill \small\textbf{\rule{0.5em}{0.5em}}

### 6.4 Proof of Proposition 1

Noting that $B(t) = \lfloor \log_2(2t) \rfloor$, the truncation error can be bounded as

$$\gamma - \sum_{j=1}^{N} a_j = \sum_{j=N+1}^{\infty} \frac{B(j-1)}{2j(2j-1)(2j-2)} = \sum_{j=N}^{\infty} \frac{B(j)}{2j(2j+1)(2j+2)}$$

$$< \sum_{j=N}^{\infty} \frac{\log_2(2j)}{8j^3}. \hfill (38)$$

Each term $\log_2(2j)/8j^3$ in (38) satisfies

$$\frac{\log_2(2j)}{8j^3} = \int_{j-1}^{j} \frac{\log_2(2x)}{8x^3} dx < \int_{j-1}^{j} \frac{\log_2(2x+1)}{8x^3} dx,$$  \hfill (39)

and therefore

$$\gamma - \sum_{j=1}^{N} a_j < \int_{N-1}^{\infty} \frac{\log_2(2x+1)}{8x^3} dx < \int_{N-1}^{\infty} \frac{\log_2\left(\frac{2x}{2(N-1)}\right)}{8x^3} dx$$

$$= \int_{N-1}^{\infty} \frac{\log_2 x + 1 + \log_2\left(1 + \frac{1}{2(N-1)}\right)}{8x^3} dx. \hfill (40)$$
Using the fact that
\[
\log_2 \left( 1 + \frac{1}{2(N-1)} \right) < \frac{1}{2(N-1) \ln 2},
\]
(41)
inequality (40) becomes
\[
\gamma - \sum_{j=1}^{N} a_j < \frac{1}{8 \ln 2} \int_{N-1}^{\infty} \frac{\ln x}{x^3} \, dx + \left( \frac{1}{8} + \frac{1}{16(N-1) \ln 2} \right) \int_{N-1}^{\infty} \frac{dx}{x^3}.
\]
(42)
Evaluating the two integrals gives
\[
\gamma - \sum_{j=1}^{N} a_j < \frac{1}{32(N-1)^2} \left( \frac{2 \ln(N-1) + 1 + 1/(N-1)}{\ln 2} + 2 \right)
= \frac{1}{16(N-1)^2} \left( \log_2(N-1) + 1 + \frac{1 + 1/(N-1)}{2 \ln 2} \right).
\]
(43)
Since \(2 \ln 2 > 1\),
\[
\gamma - \sum_{j=1}^{N} a_j < \frac{\lfloor \log_2(N-1) \rfloor + 3 + 1/(N-1)}{16(N-1)^2} = \frac{B(N-1) + 2 + 1/(N-1)}{16(N-1)^2}.
\]
(44)

Acknowledgment

Thanks to Peter Occil for bringing to the author’s attention the problem of simulating Euler’s constant without using floating-point arithmetic, which led to the algorithm and results presented in this paper; also for pointing out reference [11], and for some corrections to an early version of the manuscript.

References

[1] Stephen Abbott. Understanding Analysis. Springer, 2001.

[2] Milton Abramowitz and Irene A. Stegun, editors. Handbook of Mathematical Functions. Dover, tenth edition, 1972.

[3] A. W. Addison. A series representation for Euler’s constant. The American Mathematical Monthly, 74(7):823–824, 1967.

[4] Tom M. Apostol. Calculus, volume 1. John Wiley and Sons, second edition, 1967.
[5] Nayandeep Deka Baruah, Bruce C. Berndt, and Heng Huat Chan. Ramanujan’s series for $1/\pi$: A survey. *The American Mathematical Monthly*, 116(7):567–587, 2009.

[6] Iaroslav V. Blagouchine. Expansions of generalized Euler’s constants into the series of polynomials in $\pi^{-2}$ and into the formal enveloping series with rational coefficients only. *Journal of Number Theory*, 158:365–396, January 2016.

[7] Philippe Flajolet, Maryse Pelletier, and Michèle Soria. On Buffon machines and numbers. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 172–183, 2011.

[8] Te Sun Han and Mamoru Hoshi. Interval algorithm for random number generation. *IEEE Transactions on Information Theory*, 43(2):599–611, 1997.

[9] Mark Huber. Nearly optimal Bernoulli factories for linear functions. *Combinatorics, Probability and Computing*, 25(4):577–591, July 2016.

[10] M.S. Keane and George L. O’Brien. A Bernoulli factory. *ACM Transactions on Modeling and Computer Simulation*, 4(2):213–219, April 1994.

[11] Dexter Kozen. Optimal coin flipping. In F. van Breugel, E. Kashefi, C. Palamidessi, and J. Rutten, editors, *Horizons of the Mind. A Tribute to Prakash Panangaden*, pages 407–426. Springer, 2014.

[12] Krzysztof Łatuszyński, Ioannis Kosmidis, Omiros Papaspiliopoulos, and Gareth O. Roberts. Simulating events of unknown probabilities via reverse time martingales. *Random Structures and Algorithms*, 38(4):441–452, July 2011.

[13] Luis Mendo. An asymptotically optimal Bernoulli factory for certain functions that can be expressed as power series. *Stochastic Processes and their Applications*, 129(11):4366–4384, 2019.

[14] Şerban Nacu and Yuval Peres. Fast simulation of new coins from old. *Annals of Applied Probability*, 15(1A):93–115, 2005.

[15] Yutaka Nishiyama. Machin’s formula and Pi. *International Journal of Pure and Applied Mathematics*, 82(3):421–430, 2013.

[16] Yuval Peres. Iterating von Neumann’s procedure for extracting random bits. *Annals of Statistics*, 20(1):590–597, March 1992.

[17] Jonathan Sondow. New Vacca-type rational series for Euler’s constant $\gamma$ and its “alternating” analog $\ln(4/\pi)$. In D. Chudnovsky and G. Chudnovsky, editors, *Additive Number Theory. Festschrift in Honor of the Sixtieth Birthday of Melvyn B. Nathanson*, pages 331–340. Springer, 2010.
[18] John von Neumann. Various techniques used in connection with random digits. In A. S. Householder, G. E. Forsythe, and H. H. Germond, editors, Monte Carlo Method, volume 12 of National Bureau of Standards Applied Mathematics Series, chapter 13, pages 36–38. US Government Printing Office, 1951.