On Buffon Machines and Numbers

Philippe Flajolet, Maryse Pelletier, Michèle Soria

On Buffon Machines and Numbers

Philippe Flajolet* Maryse Pelletier† Michèle Soria†

philippe.flajolet@inria.fr, maryse.pelletier@lip6.fr, michele.soria@lip6.fr

Abstract

The well-know needle experiment of Buffon can be regarded as an analog (i.e., continuous) device that stochastically "computes" the number $2/\pi \doteq 0.63661$, which is the experiment's probability of success. Generalizing the experiment and simplifying the computational framework, we consider probability distributions, which can be produced *perfectly*, from a *discrete source* of unbiased coin flips. We describe and analyse a few simple *Buffon machines* that generate geometric, Poisson, and logarithmic-series distributions. We provide human-accessible Buffon machines, which require a dozen coin flips or less, on average, and produce experiments whose probabilities of success are expressible in terms of numbers such as π, $\exp(-1)$, $\log 2$, $\sqrt{3}$, $\cos(\frac{1}{4})$, $\zeta(5)$. Generally, we develop a collection of constructions based on *simple probabilistic mechanisms* that enable one to design Buffon experiments involving compositions of exponentials and logarithms, polylogarithms, direct and inverse trigonometric functions, algebraic and hypergeometric functions, as well as functions defined by integrals, such as the Gaussian error function.

Introduction

Buffon's experiment (published in 1777) is as follows [1, 4]. *Take a plane marked with parallel lines at unit distance from one another; throw a needle at random; finally, declare the experiment a success if the needle intersects one of the lines.* Basic calculus implies that the probability of success is $2/\pi$.

One can regard Buffon's experiment as a simple analog device that takes as input *real* uniform $[0,1]$–random variables (giving the position of the centre and the angle of the needle) and outputs a discrete $\{0,1\}$–random variable, with 1 for success and 0 for failure. The process then involves *exact* arithmetic operations over real numbers. In the same vein, the classical problem of *simulating* random variables can be described as the construction of analog devices (algorithms) operating over the *reals* and equipped with a *real* source of randomness, which are both simple and computationally efficient. The encyclopedic treatise of Devroye [6] provides many examples relative to the simulation of distributions, such as Gaussian, exponential, Cauchy, stable, and so on.

Law		*supp.*	*distribution*	*gen.*
Bernoulli	$\mathrm{Ber}(p)$	$\{0,1\}$	$\mathbb{P}(X=1)=p$	$\Gamma\mathrm{B}(p)$
geometric	$\mathrm{Geo}(\lambda)$	$\mathbb{Z}_{\geq 0}$	$\mathbb{P}(X=r)=\lambda^r(1-\lambda)$	$\Gamma\mathrm{G}(\lambda)$
Poisson,	$\mathrm{Poi}(\lambda)$	$\mathbb{Z}_{\geq 0}$	$\mathbb{P}(X=r)=e^{-\lambda}\frac{\lambda^r}{r!}$	$\Gamma\mathrm{P}(\lambda)$
logarithmic,	$\mathrm{Log}(\lambda)$	$\mathbb{Z}_{\geq 0}$	$\mathbb{P}(X=r)=\frac{1}{L}\frac{\lambda^r}{r}$	$\Gamma\mathrm{L}(\lambda)$

Figure 1: Main discrete probability laws: support, expression of the probabilities, and naming convention for generators ($L := \log(1-\lambda)^{-1}$).

Buffon machines. Our objective is the *perfect simulation of discrete random variables* (i.e., variables supported by $\mathbb{Z}$ or one of its subsets; see Fig. 1). In this context, it is natural to start with a discrete source of randomness that produces *uniform random bits* (rather than uniform $[0,1]$ real numbers), since we are interested in finite computation (rather than infinite-precision real numbers); cf Fig. 2.

Definition 1. *A* Buffon machine *is a deterministic device belonging to a computationally universal class (Turing machines, equivalently, register machines), equipped with an external* source of independent uniform random bits *and input–output queues capable of storing integers (usually, $\{0,1\}$-bits), which is assumed to halt*[1] with probability 1.

The fundamental question is then the following. *How does one generate,* exactly *and* efficiently*, discrete distributions using only a discrete source of random bits and finitary computations?* A pioneering work in this direction is that of Knuth and Yao [20] who discuss the power of various restricted devices (e.g., finite-state machines). Knuth's treatment [18, §3.4] and the articles [11, 36] provide additional results along these lines.

*Algorithms, INRIA, F-78153 Le Chesnay, France
†LIP6, UPMC, 4, Place Jussieu, 75005 Paris, France

[1] Machines that *always* halt can only produce Bernoulli distributions whose parameter is a dyadic rational $s/2^t$; see [20] and §1.

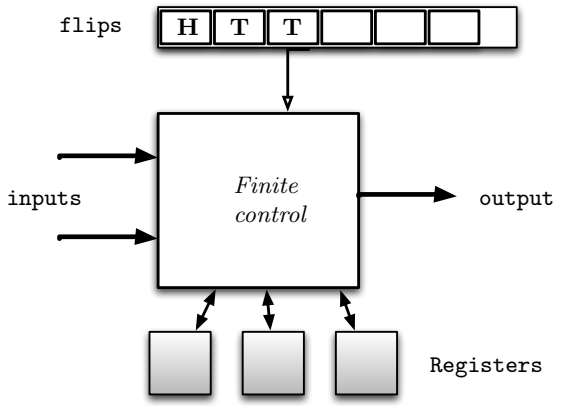


Figure 2: A Buffon machine with two inputs, one output, and three registers.

Our original motivation for studying discrete Buffon machines came from *Boltzmann samplers* for combinatorial structures, following the approach of Duchon, Flajolet, Louchard, and Schaeffer [7] and Flajolet, Fusy, and Pivoteau [9]. The current implementations relie on real number computations, and they require generating distributions such as geometric, Poisson, or logarithmic, with various ranges of parameters—since the objects ultimately produced are discrete, it is natural to try and produce them by purely discrete means.

Numbers. Here is an intriguing range of related issues. Let $\mathcal{M}$ be an input-free Buffon machine that outputs a random variable X, whose value lies in $\{0,1\}$. It can be seen from the definition that such a machine $\mathcal{M}$, when called repeatedly, produces an independent sequence of *Bernoulli random variables*. We say that $\mathcal{M}$ is a *Buffon machine* or *Buffon computer* for the number $p := \mathbb{P}(X = 1)$. We seek *simple mechanisms—Buffon machines—that produce, from uniform* $\{0,1\}$*-bits, Bernoulli variables whose probabilities of success are numbers such as*

$$(0.1)\quad 1/\sqrt{2},\quad e^{-1},\quad \log 2,\quad \frac{1}{\pi},\quad \pi - 3,\quad \frac{1}{e-1}\,.$$

This problem can be seen as a vast generalization of Buffon's needle problem, adapted to the discrete world.

Complexity and simplicity. We will impose the loosely defined constraint that the Buffon machines we consider be *short* and conceptually *simple*, to the extent of being easily implemented by a human. Thus, emulating infinite-precision computation with multiprecision interval arithmetics or appealing to functions of high complexity as primitives is disallowed[2]. Our Buffon programs only make use of simple integer counters, string registers (§2) and stacks (§3), as well as "bags" (§4). The reader may try her hand at determining Buffonness in this sense of some of the numbers listed in (0.1). We shall, for instance, produce a Buffon computer for the constant $\mathrm{Li}_3(1/2)$ of Eq. (2.11) (which involves $\log 2$, π, and $\zeta(3)$), one that is human-compatible, that consumes on average less than 6 coin flips, and that requires at most 20 coin flips in 95% of the simulations. We shall also devise *nine different ways of simulating Bernoulli distributions whose probability involves* π*, some requiring as little as five coin flips on average.* Furthermore, the constructions of this paper can all be encapsulated into a universal interpreter, briefly discussed in Section 5, which has less than 60 lines of MAPLE code and produces all the constants of Eq. (0.1), as well as many more.

In this extended abstract, we focus the discussion on *algorithmic design*. Analytic estimates can be approached by means of (probability, counting) generating functions in the style of methods of analytic combinatorics [12]; see the typical discussion in Subsection 2.1. The main results are Theorem 2.2 (Poisson and logarithmic generators), Theorem 2.3 (realizability of exps, logs, and trigs), Theorem 4.1 (general integrator), and Theorem 4.2 (inverse trig functions).

1 Framework and examples

Our approach consists in setting up a system based on the *composition* of simple probabilistic experiments, corresponding to simple computing devices. (For this reason, the Buffon machines of Definition 1 are allowed input/output registers.) The *unbiased random-bit generator*[3], with which Buffon machines are equipped will be named "**flip**". The *cost measure* of a computation (simulation) is taken to be the number of flips. (For the restricted devices we shall consider, the overall simulation cost is essentially proportional to this measure.)

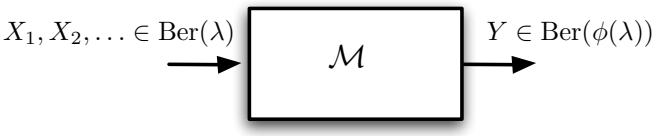
DEFINITION 2. *The function* $\lambda \mapsto \phi(\lambda)$*, defined for* $\lambda \in (0,1)$ *and with values* $\phi(\lambda) \in (0,1)$*, is said to be* weakly realizable *if there is a machine* $\mathcal{M}$*, which,*

[2] The informal requirement of *"simplicity"* can be captured by the formal notion of *program size*. All the programs we develop necessitate at most a few dozen register-machine instructions, see §5 and the Appendix, as opposed to programs based on arbitrary-precision arithmetics, which must be rather huge; cf [31]. If program size is unbounded, the problem becomes trivial, since any Turing-computable number α can be emulated by a Buffon machine with a number of coin flips that is $O(1)$ on average, e.g., by computing the sequence of digits of α on demand; see [20] and Eq. (1.5) below.

[3] This convention entails no loss of generality. Indeed, as first observed by von Neumann, suppose we only have a biased coin, where $\mathbb{P}(1) = p$, $\mathbb{P}(0) = 1 - p$, with $p \in (0,1)$. Then, one should toss the coin twice: if `01` is observed, then output 0; if `10` is observed, then output 1; otherwise, repeat with new coin tosses. See, e.g., [20, 28] for more on this topic.

when provided on its input with a perfect generator of Bernoulli variables of (unknown) parameter λ, outputs, with probability 1*, a Bernoulli random variable of parameter $\phi(\lambda)$. The function ϕ is said to be* realizable*, resp.,* strongly realizable*, if the (random) number C of coin flips has finite expectation, resp., exponential tails*[4]*.*

We shall also say that ϕ has a [weak, strong] *simulation* if it is realizable by a machine [in the weak, strong sense]. Schematically:



The devices are normally implemented as *programs*. Using $\Gamma\mathrm{B}(\lambda)$ as a generic notation for a Bernoulli generator of parameter λ, a Buffon machine that realizes the function ϕ is then equivalent to a program that can call (possibly several times) a $\Gamma\mathrm{B}(\lambda)$, as an external routine, and then outputs a random Bernoulli variable of parameters $\phi(\lambda)$. It corresponds to a construction of type $\Gamma\mathrm{B}(\lambda) \longrightarrow \Gamma\mathrm{B}(\phi(\lambda))$.

The definition is extended to machines with m inputs, in which case a function $\phi(\lambda_1, \ldots, \lambda_m)$ of m arguments is *realized*; see below for several examples. A machine with no input register then computes a function $\phi()$ of no argument, that is, a constant p, and it does so based solely on its source of unbiased coin flips: this is what we called in the previous section a Buffon machine (or computer) for p.

The fact that Buffon machines are allowed input registers makes it possible to *compose* them. For instance, if $\mathcal{M}$ and $\mathcal{N}$ realize the unary functions ϕ and ψ, connecting the output of $\mathcal{M}$ to the input of $\mathcal{N}$ realizes the composition $\psi \circ \phi$. It is one of our goals to devise Buffon computers for special values of the success probability p by composition of simpler functions, eventually only applied to the basic flip primitive.

Note that there are obstacles to what can be done: Keane and O'Brien [16], for instance, showed that the "success doubling" function $\min(2\lambda, 1)$ cannot be realized and discontinuous functions are impossible to simulate—this, for essentially measure-theoretic reasons. On the positive side, Nacu and Peres [27] show that every (computable) Lipschitz function is realizable and every (computable) real-analytic function has a strong simulation, but their constructions require *computationally unrestricted* devices; namely, sequences of approximation functions of increasing complexity. Instead, our purpose here is to show how sophisticated *perfect* simulation algorithms can be systematically and efficiently synthetized by *composition of* simple *probabilistic processes*, this without the need of hard-to-compute sequences of approximations.

To set the stage of the present study, we shall briefly review in sequence: (*i*) decision trees and polynomial functions; (*ii*) Markov chains (finite graphs) and rational functions.

Decision trees and polynomials. Given three machines P, Q, R with outputs in $\{0, 1\}$ and with $\mathbb{P}_P(1) = p$, $\mathbb{P}_Q(1) = q$, $\mathbb{P}_R(1) = r$, we can easily build machines, corresponding to loopless programs, whose probability of success is $p \cdot q$, $1 - p$, or any composition thereof: see Eq. (1.2) [top of next page] for the most important boolean primitives. We can then simulate a Bernoulli of parameter any dyadic rational $s/2^t$, starting from the unbiased flip procedure, performing t draws, and declaring a success in s designated cases. Also, by calling a Bernoulli variate of (unknown) parameter λ a fixed number m of times, then observing the sequence of outcomes and its number k of 1s, we can realize any polynomial function $a_{m,k}\lambda^k(1-\lambda)^{m-k}$, with $a_{m,k}$ any integer satisfying $0 \le a_{m,k} \le \binom{m}{k}$.

Finite graphs (Markov chains) and rational functions. We now consider programs that allow iteration and correspond to a finite control graph—these are equivalent to Markov chains, possibly parameterized by "boxes" representing calls to external Bernoulli generators. In this way, we can produce a Bernoulli generator of any rational parameter $\lambda \in \mathbb{Q}$, by combining a simpler dyadic generator with iteration. (For instance, to get a Bernoulli generator of parameter $\frac{1}{3}$, denoted by $\Gamma\mathrm{B}(\frac{1}{3})$, flip an unbiased coin twice; return success if `11` is observed, failure in case of `01` or `10`, and repeat the experiment in case of `00`.) Clearly, only rational numbers and functions can be realized by finite graphs.

A highly useful construction in this range of methods is the *even-parity* construction:

$$\text{(1.3)} \qquad \text{[even-parity]} \quad \begin{array}{l} \textbf{do}\ \{\ \textbf{if}\ P() = 0\ \textbf{then}\ \text{return}(1); \\ \qquad\qquad \textbf{if}\ P() = 0\ \textbf{then}\ \text{return}(0)\ \}. \end{array}$$

This realizes the function $p \mapsto 1/(1+p)$. Indeed, the probability of $k+1$ calls to $P()$ is $(1-p)p^k$, which, when summed over $k = 0, 2, 4, \ldots$, gives the probability of success as $1/(1+p)$; thus, this function is realizable. Combining this function with complementation ($x \mapsto 1 - x$) leads, for instance, to a way of compiling any rational probability a/b into a generator $\Gamma\mathrm{B}\left(\frac{a}{b}\right)$ whose size is proportional to the sum of digits of the continued

[4] $C \equiv C(\lambda)$ has exponential tails if there are constants K and $\rho < 1$ such that $\mathbb{P}(C > m) \le K\rho^m$.

(1.2)

Name	*realization*	*function*
Conjunction $(P \wedge Q)$	**if** $P() = 1$ **then** return$(Q())$ **else** return(0)	$p \wedge q = p \cdot q$
Disjunction $(P \vee Q)$	**if** $P() = 0$ **then** return$(Q())$ **else** return(1)	$p \vee q = p + q - pq$
Complementation $(\neg P)$	**if** $P() = 0$ **then** return(1) **else** return(0)	$1 - p$
Squaring	$(P \wedge P)$	p^2
Conditional $(R \to P \,\vert\, Q)$	**if** $R() = 1$ **then** return$(P())$ **else** return$(Q())$	$rp + (1-r)q$.
Mean	**if** flip() **then** return$(P())$ **else** return$(Q())$	$\frac{1}{2}(p+q)$.

fraction representation of a/b. (See also Eq. (1.5) below for an alternative based on binary representations and [20] for more on this topic.)

Here are furthermore two important characterizations based on works of Nacu–Peres [27] (see also Wästlund [35]) and Mossel–Peres [26]:

THEOREM 1.1. ([26, 27, 35]) *(i) Any polynomial $f(x)$ with rational coefficients that maps $(0,1)$ into $(0,1)$ is strongly realizable by a finite graph. (ii) Any rational function $f(x)$ with rational coefficients that maps $(0,1)$ into $(0,1)$ is strongly realizable by a finite graph.*

(Part (i) is based on a theorem of Pólya, relative to nonnegative polynomials; Part (ii) depends on an ingenious "block simulation" principle. The corresponding constructions however require unbounded precision arithmetics.)

We remark that we can also produce a *geometric variate* from a Bernoulli of the same parameter: just repeat till failure. This gives rise to the program

$$\text{(1.4)}\quad \text{[Geometric]}\quad \Gamma\mathrm{G}(\lambda) := \{\, K := 0;\ \mathbf{do}\ \{\, \mathbf{if}\ \Gamma\mathrm{B}(\lambda) = 0 \ \mathbf{then}\ \text{return}(K);\ K := K+1;\,\}\,\}.$$

(The even-parity construction implicitly makes use of this.) The special $\Gamma\mathrm{G}(\frac{1}{2})$ then simply iterates on the basis of a flip.

In case we have access to the complete binary expansion of p (note that this is *not* generally permitted in our framework), a Bernoulli generator is obtained by

$$\text{(1.5)}\quad \text{[Bernoulli/binary]}\quad \{\, \mathbf{let}\ Z := 1 + \Gamma\mathrm{G}(\tfrac{1}{2});\ \text{return}(\text{bit}(Z,p))\,\}.$$

In words: *in order to draw a Bernoulli variable of parameter p whose binary representation is available, return the bit of p whose random index is given by a shifted geometric variable of parameter $1/2$.* (Proof: emulate a comparison between a uniformly random $V \in [0,1]$ with $p \in [0,1]$; see [20, p. 365] for this trick.) The cost is by design a geometric of rate $1/2$. In particular, Eq. (1.5) automatically gives us a $\Gamma\mathrm{B}(p)$, for any rational $p \in \mathbb{Q}$, by means of a simple Markov chain, based on the eventual periodicity of the representation of p. (The construction will prove useful when we discuss "bags" in §4.)

2 The von Neumann schema

The idea of generating certain probability distributions by way of their Taylor expansion seems to go back to von Neumann. An early application discussed by Knuth and Yao [20], followed by Flajolet and Saheb [11], is the *exact* simulation of an exponential variate by means of random $[0,1]$–uniform variates, this by a "continuation" process that altogether *avoids multiprecision operations*. We build upon von Neumann's idea and introduce in Subsection 2.1 a general schema for random generation—the *von Neumann schema*. We then explain in Subsection 2.2 how this schema may be adapted to realize classical transcendental functions, such as $e^{-\lambda}$, $\cos(\lambda)$, only knowing a generator $\Gamma\mathrm{B}(\lambda)$.

2.1 Von Neumann generators of discrete distributions.

First a few notations from [12]. Start from a class $\mathcal{P}$ of permutations, with $\mathcal{P}_n$ the subset of permutations of size n and P_n the cardinality of $\mathcal{P}_n$. The (counting) *exponential generating function*, or EGF, is

$$P(z) := \sum_{n \geq 0} P_n \frac{z^n}{n!}.$$

For instance, the classes $\mathcal{Q}, \mathcal{R}, \mathcal{S}$ of, respectively, *all* permutations, *sorted* permutations, and *cyclic* permutations have EGFs given by $Q(z) = (1-z)^{-1}$, $R(z) = e^z$, $S(z) = \log(1-z)^{-1}$, since $Q_n = n!$, $R_n = 1$, $S_n = (n-1)!$, for $n \geq 1$. We observe that the class $\mathcal{S}$ of cyclic permutations is isomorphic to the class of permutations such that the maximum occurs in the first position: $U_1 > U_2, \ldots, U_N$. (It suffices to "break" a cycle at its maximum element.) We shall also denote by $\mathcal{S}$ the latter class, which is easier to handle algorithmically.

Let $\mathbf{U} = (U_1, \ldots, U_n)$ be a vector of real numbers. By replacing each U_j by its rank in $\mathbf{U}$, we obtain a permutation $\sigma = (\sigma_1, \ldots, \sigma_n)$, which is called the (order) *type* of $\mathbf{U}$ and is written $\text{type}(\mathbf{U})$. For instance: $\text{type}(1.41, 0.57, 3.14, 2.71) = (2,1,4,3)$. The von Neumann schema, relative to a class $\mathcal{P}$ of permutations is described in Fig. 3 and denoted by $\Gamma\mathrm{VN}[\mathcal{P}](\lambda)$. Observe that it only needs a geometric generator $\Gamma\mathrm{G}(\lambda)$, hence, eventually, only a Bernoulli generator $\Gamma\mathrm{B}(\lambda)$, whose parameter λ is *not* assumed to be known.

$\Gamma\mathrm{VN}[\mathcal{P}](\lambda) := \{$ **do** $\{$
$N := \Gamma\mathrm{G}(\lambda)$;
let $\mathbf{U} := (U_1, \ldots, U_N)$, vector of $[0,1]$–uniform
{ *bits of the* U_j *are produced on call-by-need basis*}
let $\tau :=$ trie($\mathbf{U}$); **let** $\sigma :=$ type($\mathbf{U}$);
if $\sigma \in \mathcal{P}_N$ **then** return(N) $\}\}$.

Figure 3: The von Neumann schema $\Gamma\mathrm{VN}[\mathcal{P}](\lambda)$, in its basic version, relative to a class of permutations $\mathcal{P}$ and a parameter λ (equivalently given by its Bernoulli generator $\Gamma\mathrm{B}(\lambda)$).

First, by construction, a value N is, at each stage of the iteration, chosen with probability $(1-\lambda)\lambda^N$. The procedure consists in a sequence of failed trials (when type($\mathbf{U}$) is not in $\mathcal{P}$), followed by eventual success. An iteration (trial) that succeeds then returns the value $N = n$ with probability

$$\frac{(1-\lambda)P_n\lambda^n/n!}{(1-\lambda)\sum_n P_n\lambda^n/n!} = \frac{1}{P(\lambda)}\frac{P_n\lambda^n}{n!}. \tag{2.6}$$

For $\mathcal{P}$ one of the three classes $\mathcal{Q}, \mathcal{R}, \mathcal{S}$ described above this gives us three interesting distributions:

(2.7)

all ($\mathcal{Q}$)	sorted ($\mathcal{R}$)	cyclic ($\mathcal{S}$)
$(1-\lambda)\lambda^n$	$e^{-\lambda}\frac{\lambda^n}{n!}$	$\frac{1}{L}\frac{\lambda^n}{n}$
geometric	Poisson	logarithmic.

The case of all permutations ($\mathcal{Q}$) is trivial, since no order-type restriction is involved, so that the initial value of $N \in \mathrm{Geo}(\lambda)$ is returned. It is notable that, in the other two cases ($\mathcal{R}, \mathcal{S}$), *one produces the Poisson and logarithmic distributions, by means of permutations obeying simple restrictions.*

Next, implementation details should be discussed. Once N has been drawn, we can imagine producing the U_j in sequence, by generating at each stage only the *minimal* number of bits needed to distinguish any U_j from the other ones. This corresponds to the construction of a *digital tree*, also known as *"trie"* [19, 23, 33] and is summarized by the command "**let** $\tau :=$ trie(U)" in the schema of Fig. 3. As the digital tree τ is constructed, the U_j are gradually sorted, so that the order type σ can be determined—this involves no additional flips, just bookkeeping. The test $\sigma \in \mathcal{P}_N$ is of this nature and it requires no flip at all.

The general properties of the von Neumann schema are summarized as follows.

THEOREM 2.1. (*i*) *Given an arbitrary class $\mathcal{P}$ of permutations and a parameter $\lambda \in (0,1)$, the von Neumann schema $\Gamma\,VN[\mathcal{P}](\lambda)$ produces* exactly *a discrete random variable with probability distribution*

$$\mathbb{P}(N = n) = \frac{1}{P(\lambda)}\frac{P_n\lambda^n}{n!}.$$

(*ii*) *The number K of iterations has expectation $1/s$, where $s = (1-\lambda)P(\lambda)$, and its distribution is $1 + Geo(1-s)$.*

(*iii*) *The number C of flips consumed by the algorithm (not counting the ones in $\Gamma G(\lambda)$) is a random variable with probability generating function*

$$\mathbb{E}(q^C) = \frac{H^+(\lambda, q)}{1 - H^-(\lambda, q)}, \tag{2.8}$$

where H^+, H^- are computable from the family of polynomials in (2.10) below by

$$\begin{aligned} H^+(z,q) &= (1-z)\sum_{n=0}^{\infty}\frac{P_n}{n!}h_n(q)z^n \\ H^-(z,q) &= (1-z)\sum_{n=0}^{\infty}\left(1 - \frac{P_n}{n!}\right)h_n(q)z^n. \end{aligned}$$

The distribution of C has exponential tails.

Proof. [Sketch] Items (*i*) and (*ii*) result from the discussion above. Regarding Item (*iii*), a crucial observation is that the digital tree created at each step of the iteration is only a function of the underlying *set* of values. But there is complete independence between this *set* of values and their *order type*. This justifies (2.8), where H^+, H^- are the probability generating functions associated with success and failure of one trial, respectively.

We next need to discuss the fine structure of costs. The cost of each iteration, as measured by the number of coin flips, is exactly that of generating the tree τ of random size N. The number of coin flips to build τ coincides with the *path length* of τ, written $\omega(\tau)$, which satisfies the inductive definition

$$\omega(\tau) = |\tau| + \omega(\tau_0) + \omega(\tau_1), \qquad |\tau| \geq 2, \tag{2.9}$$

where $\tau = \langle\tau_0, \tau_1\rangle$ and $|\tau|$ is the size of τ, that is, the number of elements contained in τ.

Path length is a much studied parameter, starting with the work of Knuth in the mid 1960s relative to the analysis of radix-exchange sort [19, pp. 128–134]; see also the books of Mahmoud [23] and Szpankowski [33] as well as Vallée *et al.*'s analyses under dynamical source models [5, 34]. It is known from these works that the expectation of path length, for n uniform binary sequences, is *finite*, with exact value

$$\mathbb{E}_n[\omega] = n\sum_{k=0}^{\infty}\left[1 - \left(1 - \frac{1}{2^k}\right)^{n-1}\right],$$

and asymptotic form (given by a Mellin transform analysis [10, 23, 33]): $\mathbb{E}_n[\omega] = n\log_2 n + O(n)$. A consequence of this last fact is that $\mathbb{E}(C)$ is finite, i.e., the generator $\Gamma\mathrm{VN}[\mathcal{P}](\lambda)$ has the basic *simulation (realizability) property.*

The distribution of path length is known to be asymptotically Gaussian, as $n \to \infty$, after Jacquet and Régnier [15]

and Mahmoud *et al.* [25]; see also [24, §11.2]. For our purposes, it suffices to note that the bivariate EGF

$$H(z,q) := \sum_{n=0}^{\infty} \mathbb{E}_n[q^\omega] \frac{z^n}{n!}$$

satisfies the nonlinear functional equation $H(z,q) = H\left(\frac{zq}{2}, q\right)^2 + z(1-q)$, with $H(0,q) = 1$. This equation fully determines H, since it is equivalent to a recurrence on coefficients, $h_n(q) := n![z^n]H(z,q)$, for $n \geq 2$:

$$h_n(q) = \frac{1}{1-q^n 2^{1-n}} \sum_{k=1}^{n-1} \frac{1}{2^n} \binom{n}{k} h_k(q) h_{n-k}(q). \tag{2.10}$$

The computability of H^+, H^- then results. In addition, large deviation estimates can be deduced from (2.10), which serve to establish exponential tails for C, thereby ensuring a *strong simulation property* in the sense of Definition 2.

A notable consequence of Theorem 2.1 is the possibility of generating a Poisson or logarithmic variate by a simple device: as we saw in the discussion preceding the statement of the Theorem, only *one branch* of the trie needs to be maintained, in the case of the classes $\mathcal{R}$ and $\mathcal{S}$ of (2.7).

THEOREM 2.2. *The Poisson and logarithmic distributions of parameter $\lambda \in (0,1)$ have a strong simulation by a Buffon machine, $\Gamma VN[\mathcal{R}](\lambda)$ and $\Gamma VN[\mathcal{S}](\lambda)$, respectively, which only uses a single string register.*

Since the sum of two Poisson variates is Poisson (with rate the sum of the rates), the strong simulation result extends to any $X \in \mathrm{Poi}(\lambda)$, for any $\lambda \in \mathbb{R}_{\geq 0}$. *This answers an open question of Knuth and Yao* in [20, p. 426]. We may also stress here that the distributions of costs are easily computable: with the symbolic manipulation system MAPLE, the cost of generating a Poisson(1/2) variate is found to have probability generating function (Item (*iii*) of Theorem 2.1)

$$\frac{3}{4} + \frac{7}{128}q^2 + \frac{119}{4096}q^4 + \frac{19}{1024}q^5 + \frac{2023}{131072}q^6 + \frac{179}{16384}q^7 + \cdots.$$

Interestingly enough, the analysis of the logarithmic generators involves ideas similar to those relative to a classical leader election protocol [8, 30].

2.2 Buffon computers: logarithms, exponentials, and trig functions. We can also take any of the previous constructions and specialize it by declaring a success whenever a special value $N = a$ is returned, for some predetermined a (usually $a = 0, 1$), declaring a failure, otherwise. For instance, the Poisson generator with $a = 0$ gives us in this way a Bernoulli generator with parameter $\lambda' = \exp(-\lambda)$. Since the von Neumann machine only requires a Bernoulli generator $\Gamma\mathrm{B}(\lambda)$, we thus have a purely discrete construction $\Gamma\mathrm{B}(\lambda) \longrightarrow \Gamma\mathrm{B}\left(e^{-\lambda}\right)$. Similarly, the logarithmic generator restricted to $a = 1$ provides a construction $\Gamma\mathrm{B}(\lambda) \longrightarrow \Gamma\mathrm{B}\left(\frac{\lambda}{\log(1-\lambda)^{-1}}\right)$. Naturally, these constructions can be enriched by the basic ones of Section 1, in particular, complementation.

Another possibility is to make use of the number K of iterations, which is a shifted geometric of rate $s = (1-\lambda)P(\lambda)$; see Theorem 2.1, Item (*ii*). If we declare a success when $K = b$, for some predetermined b, we then obtain yet another brand of generators. The Poisson generator used in this way with $b = 1$ gives us $\Gamma\mathrm{B}(\lambda) \longrightarrow \Gamma\mathrm{B}\left((1-\lambda)e^{\lambda}\right), \Gamma\mathrm{B}\left(\lambda e^{1-\lambda}\right)$, where the latter involves an additional complementation.

Trigonometric functions can also be brought into the game. A sequence $\mathbf{U} = (U_1, \ldots, U_n)$ is said to be *alternating* if $U_1 < U_2 > U_3 < U_4 > \cdots$. It is well known that the EGFs of the classes $\mathcal{A}^+$ of even-sized and $\mathcal{A}^-$ of odd-sized permutations are respectively $A^+(z) = \sec(z) = 1/\cos(z)$, and $A^-(z) = \tan(z) = \sin(z)/\cos(z)$. (This result, due to Désiré André around 1880, is in most books on combinatorial enumeration, e.g., [12, 13, 32].) Note that the property of being alternating can once more be tested sequentially: only a partial expansion of the current value of U_j needs to be stored at any given instant. By making use of the properties $\mathcal{A}^+, \mathcal{A}^-$, with, respectively $N = 0, 1$, we then obtain new trigonometric constructions.

In summary:

THEOREM 2.3. *The following functions admit a strong simulation:*

$$e^{-x}, \quad e^{x-1}, \quad (1-x)e^x, \quad xe^{1-x},$$
$$\frac{x}{\log(1-x)^{-1}}, \quad \frac{1-x}{\log(1/x)}, \quad (1-x)\log\frac{1}{1-x}, \quad x\log\frac{1}{x},$$
$$\cos(x), \quad \frac{1-x}{\cos(x)}, \quad \frac{x}{\tan(x)}, \quad (1-x)\tan(x).$$

2.3 Polylogarithmic constants. The probability that a vector $\mathbf{U}$ is such that $U_1 > U_2, \ldots, U_n$ (the first element is largest) equals $1/n$, a property that underlies the logarithmic series generator. By sequentially drawing r several such vectors and requiring success on *all* r trials, we deduce constructions for families involving the polylogarithmic functions, $\mathrm{Li}_r(z) := \sum_{n\geq 1} z^n/n^r$, with $r \in \mathbb{Z}_{\geq 1}$. Of course, $\mathrm{Li}_1(1/2) = \log 2$. The few special evaluations known for polylogarithmic values (see the books by Berndt on Ramanujan [3, Ch. 9] and by Lewin [21, 22]) include $\mathrm{Li}_2(1/2) = \frac{\pi^2}{12} - \frac{1}{2}\log^2 2$ and

$$\mathrm{Li}_3(\frac{1}{2}) = \frac{\log^3 2}{6} - \frac{\pi^2 \log 2}{12} + \frac{7\zeta(3)}{8}, \ \zeta(s) := \sum_{n\geq 1} \frac{1}{n^s}. \tag{2.11}$$

By rational convex combinations, we obtain Buffon computers for $\frac{\pi^2}{24}$ and $\frac{7}{32}\zeta(3)$. Similarly, the celebrated BBP (Bailey–Borwein–Plouffe) formulae [2] can be implemented as Buffon machines.

3 Square roots, algebraic, and hypergeometric functions

We now examine a new brand of generators based on properties of *ballot sequences*, which open the way to new constructions, including an important square-root mechanism. The probability that, in $2n$ tosses of a fair coin, there are as many heads as tails is $\varpi_n = \frac{1}{2^{2n}}\binom{2n}{n}$. The property is easily testable with a single integer counter R subject only to the basic operation $R := R \pm 1$ and to the basic test $R \stackrel{?}{=} 0$. From this, one can build a square-root computer and, by repeating the test, certain hypergeometric constants can be obtained.

3.1 Square-roots Let N be a random variable with distribution Geo(λ). Assume we flip $2N$ coins and return a success, if the score of heads and tails is balanced. The probability of success is

$$S(\lambda) := \sum_{n=0}^{\infty} (1-\lambda)\lambda^n \varpi_n = \sqrt{1-\lambda}.$$

(The final simplification is due to the binomial expansion of $(1-x)^{-1/2}$.) This simple property gives rise to the *square-root construction* due to Wästlund [35] and Mossel–Peres [26]:

$\Gamma\mathrm{B}\left(\sqrt{1-\lambda}\right) :=$ { **let** $N := \Gamma\mathrm{G}(\lambda)$;
draw $X_1, \ldots, X_{2N}$ with $\mathbb{P}(X_j = +1) = \mathbb{P}(X_j = -1) = \frac{1}{2}$;
set $\Delta := \sum_{j=0}^{2N} X_j$;
if $\Delta = 0$ **then** return(1) **else** return(0) }.

The mean number of coin flips used is then simply obtained by differentiation of generating functions.

THEOREM 3.1. ([26, 35]) *The square-root construction yields a Bernoulli generator of parameter $\sqrt{1-\lambda}$, given a $\Gamma B(\lambda)$. The mean number of coin flips required, not counting the ones involved in the calls to $\Gamma B(\lambda)$, is $\frac{2\lambda}{1-\lambda}$. The function $\sqrt{1-\lambda}$ is strongly realizable.*

By complementation of the original Bernoulli generator, we also have a construction $\Gamma\mathrm{B}(\lambda) \longrightarrow \Gamma\mathrm{B}(1-\lambda) \longrightarrow \Gamma\mathrm{B}\left(\sqrt{\lambda}\right)$, albeit one that is irregular at 0.

Note 1. *Computability with a pushdown automaton.* It can be seen that the number N in the square-root generator never needs to be stored explicitly: an equivalent form is

$\Gamma\mathrm{B}\left(\sqrt{1-\lambda}\right) :=$ { **do** { $\Delta := 0$;
if $\Gamma\mathrm{B}(\lambda) = 0$ **then break**;
if flip=1 **then** $\Delta := \Delta + 1$ **else** $\Delta := \Delta - 1$;
if flip=1 **then** $\Delta := \Delta + 1$ **else** $\Delta := \Delta - 1$ }
if $\Delta = 0$ **then** return(1) **else** return(0) }.

In this way, only a stack of unary symbols needs to be maintained: the stack keeps track of the absolute value $|\Delta|$ stored in unary, the finite control can keep track of the sign of Δ. We thus have *realizability of the square-root construction by means of a pushdown (stack) automaton.*

This suggests a number of variants of the square-root construction, which are also computable by a pushdown automaton. For instance, assume that, upon the condition "flip=1", one does $\Delta := \Delta + 2$ (and still does $\Delta := \Delta - 1$ otherwise). The sequences of H (heads) and T (tails) that lead to eventual success (i.e., the value 1 is returned) correspond to lattice paths that are bridges with vertical steps in $\{+2, -1\}$; see [12, §VII.8.1]. The corresponding counting generating function is then $S(z) = \sum_{n\geq 0}\binom{3n}{n} z^{3n}$, and the probability of success is $(1-\lambda)S\left(\frac{\lambda}{2}\right)$. As it is well known (via Lagrange inversion), the function $S(z)$ is a variant of an algebraic function of degree 3; namely, $S(z) = (1 - 3zY(z)^2)^{-1}$, where $Y(z) = z + zY(z)^3$, and $Y(z) = z + z^4 + 3z^7 + \cdots$ is a generating function of ternary trees. One can synthetize in the same way the family of algebraic functions

$$S(z) \equiv S^{[t]}(z) = \sum_{n\geq 0} \binom{tn}{n} z^{tn},$$

by updating Δ with $\Delta := \Delta + (t-1)$.

As a consequence of Theorems 2.3 and 3.1, the function $\sin(\lambda)$ is strongly realizable, since $\sin(\lambda) = \sqrt{1-\cos(\lambda)^2}$.

3.2 Algebraic functions and stochastic grammars. It is well known that unambiguous context-free grammars are associated with generating functions that are algebraic: see [12] for background (the Chomsky–Schützenberger Theorem).

DEFINITION 3. *A* binary stochastic grammar *(a "bistoch") is a context-free grammar whose terminal alphabet is binary, conventionally $\{H, T\}$, where each production is of the form*

$$\mathcal{X} \longrightarrow H\mathfrak{m} + T\mathfrak{n}, \tag{3.12}$$

with $\mathfrak{m}, \mathfrak{n}$ that are monomials in the non-terminal symbols. It is assumed that each non-terminal is the left hand side of at most one production.

Let G, with axiom $\mathcal{S}$, be a bistoch. We let $\mathbf{L}[G; \mathcal{S}]$ be the language associated with $\mathcal{S}$. By the classical theory of formal languages and automata, this language can be recognized by a pushdown (stack) automata. The constraint that there is a single production for each non-terminal on the left means that the automaton corresponding to a bistoch is *deterministic*. (It is then a simple matter to come up with a recursive procedure that parses a word in $\{H, T\}^\star$ according to a non-terminal symbols S.) In order to avoid trivialities, we assume that all non-terminals are "useful", meaning

{ **let** $N := \Gamma\mathrm{G}(\lambda)$;
draw $w := X_1 X_2 \cdots X_N$ with $\mathbb{P}(X_j = H) = \mathbb{P}(X_j = T) = \frac{1}{2}$;
if $w \in \mathbf{L}(G;\mathcal{S})$ **then** return(1) **else** return(0) }.

Figure 4: The algebraic construction associated to the pushdown automaton arising from a bistoch grammar.

that each produces at least one word of $\{H, T\}^\star$. For instance, the one-terminal grammar $\mathcal{Y} = H\mathcal{Y}\mathcal{Y}\mathcal{Y} + T$ generates all Łukasiewicz codes of ternary trees [12, §I.5.3] and is closely related to the construction of Note 1.

Next, we introduce the *ordinary generating function* (or OGF) of G and S,

$$S(z) := \sum_{w \in \mathbf{L}[G;\mathcal{S}]} z^{|w|} = \sum_{n \geq 0} S_n z^n,$$

with S_n the number of words of length n in $\mathbf{L}[G;\mathcal{S}]$. The deterministic character of a bistoch grammar implies that the OGFs are bound by a system of equations (one for each nonterminal): from (3.12), we have $X(z) = z\widehat{\mathfrak{m}} + z\widehat{\mathfrak{n}}$, where $\widehat{\mathfrak{m}}, \widehat{\mathfrak{n}}$ are monomials in the OGFs corresponding to the non-terminals of $\mathfrak{m}, \mathfrak{n}$; see [12, §I.5.4]. For instance, in the ternary tree case: $Y = z + zY^3$.

Thus, any OGF y arising from a bistoch is a component of a system of polynomial equations, hence, an *algebraic function*. By elimination, the system reduces to a single equation $P(z, y) = 0$. We obtain, with a simple proof, a result of Mossel-Peres [26, Th. 1.2]:

THEOREM 3.2. ([26]) *To each bistoch grammar G and non-terminal $\mathcal{S}$, there corresponds a construction (Fig. 4), which can be implemented by a deterministic pushdown automaton and calls to a $\Gamma B(\lambda)$ and is of type $\Gamma B(\lambda) \longrightarrow \Gamma B\left(S\left(\frac{\lambda}{2}\right)\right)$, where $S(z)$ is the algebraic function canonically associated with the grammar G and non-terminal S.*

Note 2. *Stochastic grammars and positive algebraic functions.* First, we observe that another way to describe the process is by means of a stochastic grammar with production rules $\mathcal{X} \longrightarrow \frac{1}{2}\mathfrak{m} + \frac{1}{2}\mathfrak{n}$, where each possibility is weighted by its probability (1/2). Then fixing $N = n$ amounts to conditioning on the size n of the resulting object. This bears a superficial resemblance to branching processes, upon conditioning on the size of the total progeny, itself assumed to be finite. (The branching process may well be supercritical, as in the ternary tree case.)

The algebraic generating functions that may arise from such grammars and positive systems of equations have been widely studied. Regarding coefficients and singularities, we refer to the discussion of the Drmota–Lalley–Woods Theorem in [12, pp. 482–493]. Regarding the values of the generating functions, we mention the studies by Kiefer, Luttenberger, and Esparza [17] and by Pivoteau, Salvy, and Soria [29]. The former is motivated by the probabilistic verification of recursive Markov processes, the latter by the efficient implementation of Boltzmann samplers.

It is not (yet) known whether a function as simple as $(1-\lambda)^{-1/3}$ is realizable by a stochastic context-free grammar or, equivalently, a deterministic pushdown automaton. (We conjecture a negative answer, as it seems that only square-root and iterated square-root singularities are possible.)

3.3 Ramanujan, hypergeometrics, and a Buffon computer for $1/\pi$.

A subtitle might be: *What to do if you want to perform Buffon's experiment but don't have needles, just coins?* The identity

$$\frac{1}{\pi} = \sum_{n=0}^{\infty} \binom{2n}{n}^3 \frac{6n+1}{2^{8n+4}},$$

due to Ramanujan (see [14] for related material), lends itself somewhat miraculously to evaluation by a simple Buffon computer. The following simple experiment (the probabilistic procedure *Rama*) succeeds (returns the value 1) with probability exactly $1/\pi$. It thus constitutes a discrete analogue of Buffon's original, one with only three counters (T, a copy of T, and Δ).

procedure Rama(); {*returns 1 with probability* $1/\pi$}
S1. let $T := X_1 + X_2$, where $X_1, X_2 \in \mathrm{Geom}(\frac{1}{4})$;
S2. with probability $\frac{5}{9}$ do $T := T + 1$;
S3. for $j = 1, 2, 3$ do
S4. draw a sequence of $2T$ coin flips;
if $(\Delta \equiv \#\,\text{Heads} - \#\,\text{Tails}) \neq 0$ then return(0);
S5. return(1).

4 A Buffon integrator

Our purpose here is to develop, given a construction of type $\Gamma\mathrm{B}(\lambda) \longrightarrow \Gamma\mathrm{B}(\phi(\lambda))$, a generator for the function

$$\Phi(\lambda) = \frac{1}{\lambda} \int_0^\lambda \phi(w)\, dw. \tag{4.13}$$

An immediate consequence will be a generator for $\lambda\Phi(\lambda)$; that is, an *"integrator"*.

To start with, we discuss a purely discrete implementation of $\Gamma\mathrm{B}(\lambda) \longrightarrow \Gamma\mathrm{B}(U\lambda)$, with $U \in [0, 1]$, uniformly, where multiple invocations of $\Gamma\mathrm{B}(\lambda)$ must involve the same value of U. Conceptually, it suffices to draw U as an infinite sequence of flips, then make use of this U to get a $\Gamma\mathrm{B}(U)$ and then appeal to the conjunction (product) construction to get a $\Gamma\mathrm{B}(\lambda U)$ as $\Gamma\mathrm{B}(U) \cdot \Gamma\mathrm{B}(\lambda)$. To implement this idea, it suffices to resort to *lazy evaluation*. One may think of U as a potentially infinite vector $(\upsilon_1, \upsilon_2, \ldots)$, where υ_j represents the jth bit of U. Only a finite section of the vector is used at any given time and the υ_j are initially undefined (or unspecified). Remember that a $\Gamma\mathrm{B}(U)$ is simply obtained by fetching the bit of U that is of order J, where $J \in 1 + \mathrm{Geo}(\frac{1}{2})$; cf Eq. (1.5). In our relaxed lazy context, whenever such a bit υ_j is fetched, we first examine

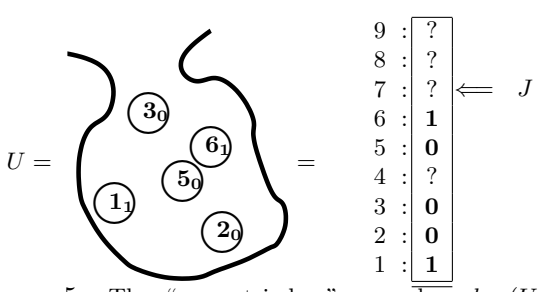


Figure 5: The "geometric-bag" procedure $bag(U)$: two graphic representations of a state (the pairs index–values and a partly filled register).

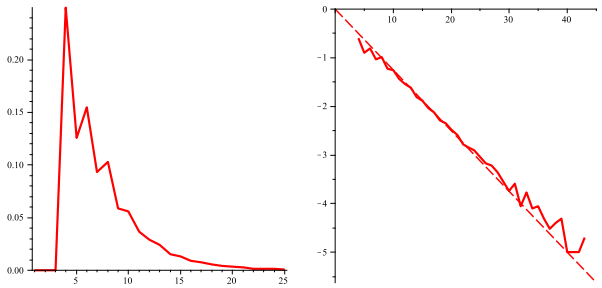


Figure 6: The distribution of costs of the Machin machine (4.15). *Left*: histogram. *Right*: decimal logarithms of the probabilities, compared to $\log_{10}(10^{-k/8})$ (dashed line).

whether it has already been assigned a $\{0,1\}$–value; if so, we return this value; if not, we first perform a flip, assign it to v_j, and return the corresponding value: see Fig. 5. (The implementation is obvious: one can maintain an association list of the already "known" indices and values, and update it, as the need arises; or keep a boolean vector of not yet assigned values; or encode a yet unassigned position by a special symbol, such as '?' or '-1'. See the Appendix for a simple implementation.)

Assume that $\phi(\lambda)$ is realized by a Buffon machine that calls a Bernoulli generator $\Gamma\mathrm{B}(\lambda)$. If we replace $\Gamma\mathrm{B}(\lambda)$ by $\Gamma\mathrm{B}(\lambda U)$, as described in the previous paragraph, we obtain a Bernoulli generator whose parameter is $\phi(\lambda U)$, where U is uniform over $[0,1]$. This is equivalent to a Bernoulli generator whose parameter is $\int_0^1 \phi(\lambda u)\,du = \Phi(\lambda)$, with $\Phi(\lambda)$ as in (4.13).

THEOREM 4.1. *Let $\phi(\lambda)$ be realizable by a Buffon machine $\mathcal{M}$. Then the function $\Phi(\lambda) = \frac{1}{\lambda}\int_0^\lambda \phi(w)\,dw$ is realizable by addition of a geometric bag to $\mathcal{M}$. In particular, if $\phi(\lambda)$ is realizable, then its integral taken starting from 0 is also realizable.*

This result paves the way to a large number of derived constructions. For instance, starting from the even-parity construction of §1, we obtain $\Phi_0(\lambda) := \frac{1}{\lambda}\int_0^\lambda \frac{1}{1+w}\,dw = \frac{1}{\lambda}\log(1+\lambda)$, hence, by product, a construction for $\log(1+\lambda)$. When we now combine the parity construction with "squaring", where a $\Gamma\mathrm{B}(p)$ is replaced by the product $\Gamma\mathrm{B}(p)\cdot\Gamma\mathrm{B}(p)$, we obtain $\Phi_1(\lambda) := \frac{1}{\lambda}\int_0^\lambda \frac{dw}{1+w^2} = \frac{1}{\lambda}\arctan(\lambda)$, hence also $\arctan(\lambda)$. When use is made of the exponential (Poisson) construction $\lambda \mapsto e^{-\lambda}$, one obtains (by squaring and after multiplications) a construction for $\Phi_2(\lambda) := \int_0^\lambda e^{-w^2/2}\,dw$, so that the error function ("erf") is also realizable. Finally, the square-root construction combined with parity and integration provides $\Phi_3(\lambda) := \int_0^\lambda \frac{\sqrt{1-w^2}}{1+w}\,dw = -1+\sqrt{1-\lambda^2}+\arcsin(\lambda)$, out of which we can construct $\frac{1}{2}\arcsin(\lambda)$. In summary:

THEOREM 4.2. *The following functions are strongly realizable* $(0 \le x < 1)$*:*

$$\log(1+x), \quad \arctan(x), \quad \frac{1}{2}\arcsin(x), \quad \int_0^x e^{-w^2/2}\,dw.$$

The first two only require one bag; the third requires a bag and a stack; the fourth can be implemented with a string register and bag.

Buffon machines for π. The fact that $\Phi_1(1) = \arctan(1) = \frac{\pi}{4}$. yields a Buffon computer for $\pi/4$. There are further simplifications due to the fact that $\Gamma\mathrm{B}(1)$ is trivial: this computer then only makes use of the U vector. Given its extreme simplicity, we can even list the complete code of this Madhava–Gregory–Leibniz (MGL) generator for $\pi/4$:

```
MGL:=proc()    do
      if bag(U)=0 then return(1) fi;
      if bag(U)=0 then return(1) fi;
      if bag(U)=0 then return(0) fi;
      if bag(U)=0 then return(0) fi; od; end.
```

The Buffon computer based on arctan(1) works fine for small simulations. For instance, based on 10,000 experiments, we infer the approximate value $\pi/4 \approx \mathbf{0.78}76$, whereas $\pi/4 \doteq \mathbf{0.78}539$, with a mean number of flips per experiment about 27. However, values of U very close to 1 are occasionally generated (the more so, as the number of simulation increases). Accordingly, *the expected number of flips is infinite*, a fact to be attributed to slow convergence in the Madhava–Gregory–Leibniz series, $\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$.

The next idea is to consider formulae of a kind made well-known by Machin, who appealed to arctangent addition formulae in order to reach the record computation of 100 digits of π in 1706. For our purposes, a formula without negative signs is needed, the simplest of which,

$$\frac{\pi}{4} = \arctan\left(\frac{1}{2}\right) + \arctan\left(\frac{1}{3}\right), \tag{4.14}$$

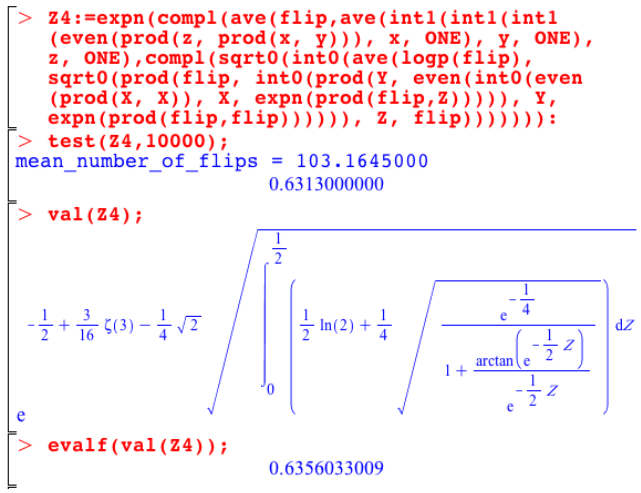


Figure 7: Screen copy of a MAPLE session fragment showing: (*i*) the symbolic description of a generator; (*ii*) a simulation of 10^4 executions having a proportion of successes equal to **0.63**130, with a mean number of flips close to 103; (*iii*) the actual symbolic value of the probability of success and its numerical evaluation **0.63**560$\cdots$.

being especially suitable for a short program is easily compiled *in silico* under the form

$$\frac{\pi}{4} = \frac{1}{2}\left[2\arctan\left(\frac{1}{2}\right) + \frac{2}{3}\cdot 3\arctan\left(\frac{1}{3}\right)\right]. \tag{4.15}$$

(This last form only uses the realizable functions $\lambda^{-1}\arctan(\lambda)$, $2\lambda/3$ and the binary operation $\frac{1}{2}[p+q]$.)

With 10^6 simulations, we obtained an estimate $\pi/4 \approx$ **0.785**98, to be compared to the exact value $\pi/4 =$ **0.785**39$\cdots$; that is, an error of about $6\cdot 10^{-4}$, well within normal statistical fluctuations. The empirically measured average number of flips per generation of this Machin-like $\Gamma\mathrm{B}(\pi/4)$ turned out to be about 6.45 coin flips. Fig. 6 furthermore displays the empirical distribution of the number of coin flips, based on another campaign of 10^5 simulations. The distribution of costs appears to have exponential tails matching fairly well the approximate formula $\mathbb{P}(C=k)\approx 10^{-k/8}$. The complete code for a version of this generator, which produces $\frac{\pi}{8}$, is given in the Appendix.

Yet an alternative construction is based on the arcsine and Φ_3. Many variations are possible, related to multiple or iterated integrals (use several bags).

5 Experiments

We have built a complete prototype implementation under the MAPLE symbolic manipulation system, in order to test and validate the ideas expounded above; see Fig. 7. A generator, such as $Z4$ of Fig. 7, is specified as a composition of basic constructions, such as $f \mapsto \exp(-f)$ [`expn`], $f \mapsto \sqrt{f}$ [`sqrt0`], $f \mapsto \int f$ [`int1`], and so on. An interpreter using as source of randomness the built-in function `random` then takes this description and produces a $\{0,1\}$ result; this interpreter, which is comprised of barely 60 lines, contains from one to about a dozen instructions for each construction. In accordance with the conditions of our contract, only simple register manipulations are ever used, so that a transcription in C or Java would be of a roughly comparable size.

We see here that even a complicated constant such as the "value" of the probability associated with $Z4$,

$$e^{-\frac{1}{2}+\frac{3}{16}\zeta(3)-\frac{1}{4}\sqrt{2}}\sqrt{\int_0^{\frac{1}{2}}\frac{1}{2}\ln 2+\frac{1}{4}\sqrt{e^{-\frac{1}{4}}\left(1+\frac{\arctan\left(e^{-Z/2}\right)}{e^{-Z/2}}\right)^{-1}}\,dZ},$$

is effectively simulated with an error of $4\,10^{-3}$, which is once more consistent with normal statistical fluctuations. For such a complicated constant, the observed mean number of flips is a little above 100. Note that the quantity $\zeta(3)$ is produced (as well as retrieved automatically by MAPLE's symbolic engine!) from Beuker's triple integral: $\frac{7}{8}\zeta(3) = \int_0^1\int_0^1\int_0^1 \frac{1}{1+xyz}\,dx\,dy\,dz$. (On batches of 10^5 experiments, that quantity alone only consumed an average of 6.5 coin flips, whereas the analogous $\frac{31}{32}\zeta(5)$ required barely 6 coin flips on average.)

Note that the implementation is, by nature, freely extendible. Thus, given integration and our basic primitives (e.g., $\mathrm{even(f)} \equiv \frac{1}{1+f}$), we readily program an arc-tangent as a one-liner,

$$\arctan(f) = f\cdot\left[\frac{1}{f}\int_0^f \frac{dx}{1+x^2}\right], \tag{5.16}$$

and similarly for sine, arc-sine, erf, etc, with the symbolic engine automatically providing the symbolic and numerical values, as a validation.

Here is finally a table recapitulating nine ways of building Buffon machines for π-related constants, with, for each of the methods, the value, and empirical average of the number of coin flips, as observed over 10^4 simulations:

$\mathrm{Li}_2(\frac{1}{2})$	Rama	arcsin $[1;\frac{1}{\sqrt{2}};\frac{1}{2}]$			arctan $[\frac{1}{2}+\frac{1}{3};1]$		$\zeta(4)$	$\zeta(2)$
$\frac{\pi^2}{24}$	$\frac{1}{\pi}$	$\frac{\pi}{4}$	$\frac{\pi}{4}$	$\frac{\pi}{12}$	$\frac{\pi}{4}$	$\frac{\pi}{4}$	$\frac{7\pi^4}{720}$	$\frac{\pi^2}{12}$
7.9	10.8	76.5 (∞)	16.2	4.9	6.5	26.7 (∞)	6.2	7.2.

(The tag "∞" means that the expected cost of the simulation is infinite—a weak realization.)

6 Conclusion

As we pointed out in the introduction, every computable number can be simulated by a machine, but one that,

in general, will violate our charter of simplicity (as measured, typically, by program size). Numbers accessible to our framework seem not to include Euler's constant $\gamma \doteq 0.57721$, and we must leave it as an open problem to come up with a "natural" experiment, whose probability of success is γ. Perhaps the difficulty of the problem lies in the absence of a simple "positive" expression that could be compiled into a correspondingly simple Buffon generator. By contrast, exotic numbers, such as $\pi^{-1/\pi}$ or $e^{-\sin(1/\sqrt{7})}$ are easily simulated...

On another note, we have not considered the generation of *continuous* random variables X, specified by a distribution function $F(x) = \mathbb{P}(X \leq x)$. Von Neumann's original algorithm for an exponential variate belongs to this paradigm. In this case, the bits of X are obtained by a short computation of $O(1)$ initial bits, continued by the production of an infinite flow of *uniform* random bits. This theme is thoroughly explored by Knuth and Yao in [20]. It would be of obvious interest to be able to hybridize the von Neumann-Knuth-Yao generators of continuous distributions with our Buffon computers for discrete distributions. Interestingly, the fact that the Gaussian error function, albeit restricted to the interval $(0, 1)$, is realizable by Buffon machines suggests the possibility of a totally discrete generator for a (standard) *normal* variate.

The present work is excellently summarized by Keane and O'Brien's vivid expression of *"Bernoulli factory"* [16]. It was initially approached with a purely theoretical goal. It then came as a surprise that *a priori* stringent theoretical requirements—those of perfect generation, discreteness of the random source, and simplicity of the mechanisms—could lead to computationally efficient algorithms. We have indeed seen many cases, from Bernoulli to logarithmic and Poisson generators, where short programs and the execution of just a few dozen instructions suffice!

Acknowledgements. This work was supported by the French ANR Projects Gamma, Boole, and Magnum. The authors are grateful to the referees of SODA-2011 for their perceptive and encouraging comments.

References

[1] Badger, L. Lazzarini's lucky approximation of π. *Mathematics Magazine 67*, 2 (1994), 83–91.

[2] Bailey, D., Borwein, P., and Plouffe, S. On the rapid computation of various polylogarithmic constants. *Mathematics of Computation 66*, 218 (1997), 903–913.

[3] Berndt, B. C. *Ramanujan's Notebooks, Part I.* Springer Verlag, 1985.

[4] Buffon, Comte de. Essai d'arithmétique morale. In *Histoire Naturelle, générale et particulière. Servant de suite à l'Histoire Naturelle de l'Homme. Supplément, Tome Quatrième.* Imprimerie Royale, Paris, 1749–1789. Text available at `gallica.bnf.fr`. (Buffon's needle experiment is described on pp. 95–105.).

[5] Clément, J., Flajolet, P., and Vallée, B. Dynamical sources in information theory: A general analysis of trie structures. *Algorithmica 29*, 1/2 (2001), 307–369.

[6] Devroye, L. *Non-Uniform Random Variate Generation.* Springer Verlag, 1986.

[7] Duchon, P., Flajolet, P., Louchard, G., and Schaeffer, G. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing 13*, 4–5 (2004), 577–625. Special issue on Analysis of Algorithms.

[8] Fill, J. A., Mahmoud, H. M., and Szpankowski, W. On the distribution for the duration of a randomized leader election algorithm. *The Annals of Applied Probability 6*, 4 (1996), 1260–1283.

[9] Flajolet, P., Fusy, É., and Pivoteau, C. Boltzmann sampling of unlabelled structures. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithmics and Combinatorics* (2007), D. A. *et al.*, Ed., SIAM Press, pp. 201–211. Proceedings of the New Orleans Conference.

[10] Flajolet, P., Gourdon, X., and Dumas, P. Mellin transforms and asymptotics: Harmonic sums. *Theoretical Computer Science 144*, 1–2 (June 1995), 3–58.

[11] Flajolet, P., and Saheb, N. The complexity of generating an exponentially distributed variate. *Journal of Algorithms 7* (1986), 463–488.

[12] Flajolet, P., and Sedgewick, R. *Analytic Combinatorics.* Cambridge University Press, 2009. 824 pages. Also available electronically from the authors' home pages.

[13] Goulden, I. P., and Jackson, D. M. *Combinatorial Enumeration.* John Wiley, New York, 1983.

[14] Guillera, J. A new method to obtain series for $1/\pi$ and $1/\pi^2$. *Experimental Mathematics 15*, 1 (2006), 83–89.

[15] Jacquet, P., and Régnier, M. Trie partitioning process: Limiting distributions. In *CAAP'86* (1986), P. Franchi-Zanetacchi, Ed., vol. 214 of *Lecture Notes in Computer Science*, pp. 196–210. Proceedings of the 11th Colloquium on Trees in Algebra and Programming, Nice France, March 1986.

[16] Keane, M. S., and O'Brien, G. L. A Bernoulli factory. *ACM Transactions on Modeling and Computer Simulation (TOMACS) 4*, 2 (1994), 213–219.

[17] Kiefer, S., Luttenberger, M., and Esparza, J. On the convergence of Newton's method for monotone systems of polynomial equations. In *Symposium on Theory of Computing (STOC'07)* (2007), ACM Press, pp. 217–226.

[18] Knuth, D. E. *The Art of Computer Programming*, 3rd ed., vol. 2: Seminumerical Algorithms. Addison-Wesley, 1998.

[19] Knuth, D. E. *The Art of Computer Programming*, 2nd ed., vol. 3: Sorting and Searching. Addison-Wesley, 1998.

[20] Knuth, D. E., and Yao, A. C. The complexity of nonuniform random number generation. In *Algorithms and complexity (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1976)*. Academic Press, New York, 1976, pp. 357–428.

[21] Lewin, L. *Polylogarithms and Associated Functions*. North Holland Publishing Company, 1981.

[22] Lewin, L., Ed. *Structural Properties of Polylogarithms*. American Mathematical Society, 1991.

[23] Mahmoud, H. M. *Evolution of Random Search Trees*. John Wiley, 1992.

[24] Mahmoud, H. M. *Sorting, A Distribution Theory*. Wiley-Interscience, New York, 2000.

[25] Mahmoud, H. M., Flajolet, P., Jacquet, P., and Régnier, M. Analytic variations on bucket selection and sorting. *Acta Informatica 36*, 9-10 (2000), 735–760.

[26] Mossel, E., and Peres, Y. New coins from old: Computing with unknown bias. *Combinatorica 25*, 6 (2005), 707–724.

[27] Nacu, Ş., and Peres, Y. Fast simulation of new coins from old. *The Annals of Applied Probability 15*, 1A (2005), 93–115.

[28] Peres, Y. Iterating Von Neumann's procedure for extracting random bits. *Annals of Statistics 20*, 1 (1992), 590–597.

[29] Pivoteau, C., Salvy, B., and Soria, M. Boltzmann oracle for combinatorial systems. *Discrete Mathematics & Theoretical Computer Science Proceedings* (2008). *Mathematics and Computer Science Conference*. In press, 14 pages.

[30] Prodinger, H. How to select a loser. *Discrete Mathematics 120* (1993), 149–159.

[31] Schönhage, A., Grotefeld, A., and Vetter, E. *Fast Algorithms, A Multitape Turing Machine Implementation*. Bibliographisches Institut, Mannheim, 1994.

[32] Stanley, R. P. *Enumerative Combinatorics*, vol. II. Cambridge University Press, 1999.

[33] Szpankowski, W. *Average-Case Analysis of Algorithms on Sequences*. John Wiley, 2001.

[34] Vallée, B. Dynamical sources in information theory: Fundamental intervals and word prefixes. *Algorithmica 29*, 1/2 (2001), 262–306.

[35] Wästlund, J. Functions arising by coin flipping. Technical Report, K.T.H, Stockholm, 1999.

[36] Yao, A. C. Contex-free grammars and random number generation. In *Combinatorial Algorithms on Words* (1985), A. Apostolico and Z. Galil, Eds., vol. 12 of *NATO Advance Science Institute Series.* Series F: Computer and Systems Sciences, Springer Verlag, pp. 357–361.

Appendix: A complete Buffon machine for $\frac{\pi}{8}$

Here is the complete pseudo-code (in fact an executable Maple code), cf procedure `Pi8` below, for a $\pi/8$ experiment, based on Eq. (4.14) and not using any high-level primitive. It exemplifies many constructions seen in the text. The translation to various low level languages, such as `C`, should be immediate, possibly up to inlining of code or macro expansions, in case procedures cannot be passed as arguments of other procedures. The expected number of coin flips per experiment is about 4.92.

The *flip* procedure (returns a pseudo-random bit):

```
flip:=proc() if rand()/10.^12<1/2
   then return(1) else return(0) fi; end:
```

A $\Gamma G(\frac{1}{2})$ returns a geometric of parameter $\frac{1}{2}$; cf Eq. (1.4):

```
1.  geo_half:=proc () local j;  j := 0;
2.    while flip() = 0 do j:=j+1 od; return j end:
```

A $\Gamma B(\frac{1}{3})$ returns a Bernoulli of parameter $\frac{1}{3}$; cf Eq. (1.5):

```
3.  bern_third:=proc() local a,b;
4.    do a:=flip(); b:=flip();
5.      if not ((a=1) and (b=1)) then break fi; od;
6.    if (a=0) and (b=0) then return(1)
7.       else return(0) fi; end:
```

Bags. Initialization and result of a comparison with a random $U \in [0,1]$; cf Fig. 5 and §4:

```
    INFINITY:=50:
8.  init_bagU:=proc() local j; global U;
9.    for j from 1 to INFINITY do U[j]:=-1 od; end:
10. bagU:=proc() local k;global U; k:=1+geo_half();
11.  if U[k]=-1 then U[k]:=flip() fi; return(U[k]);
                                                 end:
```

(To obtain a *perfect* generator, dynamically increase `INFINITY`, if ever needed—the probability is $< 10^{-15}$.)

The `EVENP` construction takes $f \in \Gamma B(\lambda)$ and produces a $\Gamma B(\frac{1}{1+\lambda})$; cf Eq. (1.3):

```
12. EVENP:=proc(f) do if f()=0 then return(1); fi;
13.        if f()=0 then return(0) fi; od; end:
```

The main `atan` construction, based on bags implementing integration, takes an $f \in \Gamma B(\lambda)$ and produces a $\Gamma B(\arctan(\lambda))$; the auxiliary procedure $g()$ builds a $\Gamma B(\lambda^2 U^2)$, with $U \in [0,1]$ random; cf §4 and Eq. (5.16):

```
14. ATAN:=proc(f) local g;
15.  if f()=0 then return(0) fi;
16.  init_bagU();
17.  g:=proc() if bagU()=1 then if f()=1 then
18.     if bagU()=1 then return(f()) fi; fi; fi;
19.     return(0); end;
20.  EVENP(g); end:
```

The `Pi8` procedure is a $\Gamma B(\frac{\pi}{8})$ based on Machin's arc tangent formula (the arithmetic mean of $\arctan(1/2)$ and $\arctan(1/3)$ is taken); cf Eq. (4.14):

```
21. Pi8:=proc() if flip()=0 then ATAN(flip)
22.             else ATAN(bern_third) fi end:.
```