ON BUFFON MACHINES AND NUMBERS

PHILIPPE FLAJOLET, MARYSE PELLETIER, AND MICHÈLE SORIA

Abstract. Buffon's needle experiment is well-known: take a plane on which parallel lines at unit distance one from the next have been marked, throw a needle of unit length at random, and, finally, declare the experiment a success if the needle intersects one of the lines. Basic calculus implies that the probability of success is $\frac{2}{\pi} \doteq 0.63661$, and the experiment can be regarded as an analog (i.e., continuous) device that stochastically "computes" $\frac{2}{\pi}$. Generalizing the experiment and simplifying the computational framework, we ask ourselves which probability distributions can be produced *perfectly*, from a *discrete source* of unbiased coin flips. We describe and analyse a few simple *Buffon machines* that can generate geometric, Poisson, and logarithmic-series distributions (these are in particular required to transform continuous Boltzmann samplers of classical combinatorial structures into purely discrete random generators). Say that a *number* is *Buffon* if it is the probability of success of a probabilistic experiment based on discrete coin flippings. We provide human-accessible Buffon machines, which require a dozen coin flips or less, on average, and produce experiments whose probabilities are expressible in terms of numbers such as π, $\exp(-1)$, $\log 2$, $\sqrt{3}$, $\cos(\frac{1}{4})$, $\zeta(5)$. More generally, we develop a collection of constructions based on *simple probabilistic mechanisms* that enable one to create 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.

Keywords: *random generation, perfect simulation, computability theory, probabilistic algorithms*
MSC classification: 60-04, 60A99, 65C05, 68Q25, 68W20.

Date: June 30, 2009. A preliminary announcement of results contained in this extended abstract has been made at the *Analysis of Algorithms* Conference AofA'09, Fréjus, June 14–19, 2009.

P.F.: Algorithms, INRIA-Rocquencout, F-78153 Le Chesnay (France)
M.P. & M. S.: LIP6, Université Paris 6, 104 av. Prés. Kennedy, 75016 Paris .

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

FIGURE 1. The main probability laws: support, expression of the distribution, probability generating function (PGF), and naming convention for generators.

The Buffon experiment described in the abstract is modelled by two variates, U, V, uniformly distributed over the unit interval $[0,1]$ (representing the initial position of the centre of the needle modulo 1 and its orientation measured as a fraction of 180 degrees). The condition for intersection is that the closest distance $\min(U, 1-U)$ to a parallel line be at most half the length $\frac{1}{2}\sin(\pi V)$ of the perpendicular projection of the needle; that is, the predicate $\min(U, 1-U) \leq \frac{1}{2}\sin(\pi V)$. Simple calculations then justify the $\frac{2}{\pi}$ probability evaluation.

One can regard Buffon's experiment as a simple analog device that takes as input uniform $[0,1]$–random variables and outputs a discrete $\{0,1\}$–random variable, with 1 for success and 0 for failure. What are permitted are *exact* operations over real numbers; here, computing a minimum, multiplying by $1/2$, computing $x \mapsto \sin(\pi x)$, and determining inequalities between reals. Without attempting a formal definition, we shall name *real Buffon machine*, any abstract device that has internal registers capable of storing infinite-precision real numbers, that is equipped with a fixed set of base functions (e.g., $\pm$, max, and sin), and is capable of testing signs of numbers. The classical problem of *simulating* random variables can then be described as the contruction of real Buffon machines that are both simple and computationally efficient. We refer to the encyclopedic treatise of Devroye [4] for many examples related to the simulation of distributions of various types, such as Gaussian, exponential, Cauchy, and stable.

Discrete Buffon machines. Our objective here is the *perfect simulation of discrete random variables*, i.e., variables supported by $\mathbb{Z}$ or one of its subsets: see Fig. 1 for those we shall be considering. It is then natural to restrict the source of randomness to produce only *uniform random bits* provided by a procedure **flip**$\equiv$ $\mathrm{random}_{\{0,1\}}$. Since we are interested in finite computation (rather than infinite-precision real numbers), we introduce *discrete Buffon machines*, or *discrete generators*, whose registers can contain binary sequences of finite length ("strings"). A fundamental question is then the following.

> *How to generate discrete distributions* exactly *and* efficiently*, using only a discrete source of random bits and finitary computations?*

A pioneering work in this direction is that of Knuth and Yao [17] who discuss the power of various restricted devices (e.g., finite-state machines). Knuth's treatment [15, §3.4] and the articles [9, 31] present additional results along these lines.

Our original motivation for studying discrete Buffon machines came from *Boltzmann samplers* for combinatorial structures, following the approach of Duchon, Flajolet, Louchard, and Schaeffer [5] and Flajolet, Fusy, and Pivoteau [7]. The current implementations relie on real number computations, and the need arises to generate 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.

Buffon numbers. Here is an intriguing range of related questions. Let $\mathcal{M}$ be a discrete Buffon machine that generates a Bernoulli random variable X, whose value lies in $\{0,1\}$. We say that $\mathcal{M}$ is a *Buffon computer* for the number $p := \mathbb{P}(X=1)$. A number is said to be a *Buffon number* if there is a Buffon machine that computes it stochastically. We will be explicitly restricting attention to devices that have as simple a structure as seems possible. The question then arises as to which real numbers are Buffon numbers in that sense. In other words, we seek *whether there* *exist simple finite mechanisms that transform uniform* $\{0,1\}$*-bits into Bernoulli variables whose probabilities of success are numbers such as*

$$1/\sqrt{2}, \quad e^{-1}, \quad \log 2, \quad \frac{1}{\pi}, \quad \pi-3, \quad \frac{1}{e-1}, \quad \cdots . \tag{1}$$

This problem can be seen as a vast generalization of Buffon's needle problem.

Complexity and simplicity. For our discrete Buffon generators and computers, we add the requirement of operating *with a constant expected number of coin flippings.* We will also impose for ourselves the losely defined constraint that the programs are *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. Our programs then 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 Buffon-ness in this sense of some of the numbers listed in (1). As briefly discussed in Section 5, our universal interpreter has less than 60 lines of Maple code, being built on primitives of between one and a dozen lines. We shall for instance later produce a Buffon computer for the constant (see (11))

$$C = \frac{7}{8}\zeta(3) - \frac{\pi^2}{12}\log 2 + \frac{1}{6}(\log 2)^3 = 0.53721\,13193\cdots, \qquad \zeta(s) := \sum_{n=1}^{\infty}\frac{1}{n^s}, \tag{2}$$

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.

Warning. Due to space limitations 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 [10]; see, the typical discussion in Subsection 2.1. The main results of this note are Theorem 2 (Poisson and logarithmic generators), Theorem 3 (realizability of exps, logs, and trigs), Theorem 6 (general integrator), and Theorem 7 (inverse trig functions).

1. Framework and examples

Our purpose is to set up a system based on the composition of simple probabilistic experiments, corresponding to simple computing devices. The framework of our Buffon machines allows for a finite control, with conditionals and tests, as well as unbounded iterations (do–loops). The basic data structures used here are: (*i*) integer registers (subject to addition of ± 1 and test to 0); (*ii*) stacks, in the usual sense of automata theory; (*iii*) string registers, which are stacks with some additional operations permitted (e.g., clear from current position to the top); (*iv*) bags, which are partly specified string registers with local modification of a symbol being allowed. The Buffon machines we consider are supposed to be equipped with an *unbiased random-bit generator*[1], named "**flip**". The *cost measure* is taken to be the number of coin flips. Since we only consider restricted devices, the overall simulation cost is, in most cases, proportional to this measure.

Definition 1. *Given a class* $\mathcal{M}$ *of Buffon machines, we say that the function* $\lambda \mapsto \phi(\lambda)$*, defined for* $\lambda \in (0,1)$ *and with values* $\phi(\lambda) \in (0,1)$ *is* realizable *(or* has a simulation*) if there is a machine in* $\mathcal{M}$*, such that, when equipped with a perfect generator* $\Gamma\mathrm{B}(\lambda)$ *of a Bernoulli variable of parameter* $\lambda = \mathbb{P}(1)$*, it outputs a Bernoulli random variable of parameter* $\phi(\lambda)$*. The function* $\phi(\lambda)$ *is said to be* strongly realizable *(or* have a strong simulation*) if the number of coin flips* C *admits exponential tails: for any* $\epsilon > 0$*, there are constants* K *and* $\rho < 1$ *so that, for any* $\lambda \in [\epsilon, 1-\epsilon]$*, one has* $\mathbb{P}(C > m) \le K\rho^{-m}$*.*

If one allows totally unrestricted machines, *every* computable real number of $(0,1)$ can be simulated (e.g., make use of continued fraction convergents or dyadic approximations). Similarly, Nacu and Peres [24] show that essentially every (computable) Lipschitz function is realizable and every (computable) real-analytic function has a strong simulation, with such unrestricted devices. Rather, our purpose here is to show how sophisticated *perfect* simulation algorithms can

[1]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, start again another round of coin tossings.

be systematically and efficiently synthetized by composition of simple probabilistic processes, this without a need for any multiprecision arithmetic routines. To set the stage of the present study, we shall briefly discuss 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 below for the most important boolean primitives:

(3)

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$(Q())$	$\frac{1}{2}(p + q).$

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 parameter x a fixed number m of times, then observing the sequence of outcomes and its number k of 1s, we can realize any polynomial function $a_{m,k} x^k (1 - x)^{m-k}$, with $a_{m,k}$ any integer satisfying $0 \le a_{m,k} \le \binom{m}{k}$.

Finite graphs (Markov chains) and rational functions. We consider now programs that allow iteration and correspond to a finite control graph—these are equivalent to Markov chains, possibly parametrized 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 $\Gamma\text{B}(\frac{1}{3})$, flip an unbiased coin twice; return success if `11` is observed, failure in case of `01` or `10`, and redo the experiment in case of `00`.) Clearly, only rational functions can be realized by finite graphs. A highly useful function is the *even-parity*:

$$(4) \qquad \text{[even-parity]} \qquad \textbf{do}\ \{\ \textbf{if}\ P() = 0\ \textbf{then}\ \text{return}(1);\ \textbf{if}\ P() = 0\ \textbf{then}\ \text{return}(0)\ \}.$$

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. Here are furthermore two important characterizations based on works of Nacu–Peres [24] (see also Wästlund [30]) and Mossel–Peres [23], after adaptation to our framework.

Theorem 1 ([23, 24, 30])**.** (*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.*

(Both parts relie on the possibility of generating arbitrary rational mixtures of distributions. Part (*i*) is further based on a theorem of Pólya, relative to nonnegative polynomials; Part (*ii*) depends on an ingenious "block simulation" principle.)

We remark at this stage 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

$$(5) \qquad \text{[Geometric]} \qquad \Gamma\text{G}(\lambda) := \{\, K := 0;\ \textbf{do}\ \{\, \textbf{if}\ \Gamma\text{B}(\lambda) = 0\ \textbf{then}\ \text{return}(K);\ K := K + 1; \}\, \}.$$

(The even-parity construction implicitly makes use of this.) The particular $\Gamma\text{G}(\frac{1}{2})$ then simply iterates on the basis of a flip. Also, if we have access in some way to the complete binary expansion of p, we have a Bernoulli generator as

$$(6) \qquad \text{[Bernoulli/binary]} \qquad \Gamma\text{B}(p) := \{\, \textbf{let}\ Z := 1 + \Gamma\text{G}(\tfrac{1}{2});\ \text{return}(\text{bit}(Z, p))\, \}.$$

In words: *in order to draw a Bernoulli variable of parameter p, return the bit of p whose random index is given by a geometric variable of parameter* $1/2$. (Proof: emulate a comparison between a uniformly random $U \in [0, 1]$ with $p \in [0, 1]$.) This property gives us a $\Gamma\text{B}(p)$ for $p \in \mathbb{Q}$ by means

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

FIGURE 2. The von Neumann schema $\Gamma\mathrm{VN}[\mathcal{P}](\lambda)$, in its continuous version, relative to a class of permutations $\mathcal{P}$ and a parameter λ (equivalently given by its Bernoulli generator $\Gamma\mathrm{B}(\lambda)$).

of a simple Markov chain, based on the eventual periodicity of the representation of p. The cost is plainly a geometric of rate $1/2$. (The construction will prove valuable when we discuss "bags".)

2. **The von Neumann schema**

The idea of generating certain probability distributions by means of their Taylor expansion seems to go back to von Neumann. An early application discussed by Knuth and Yao [17], followed by Flajolet and Saheb [9], 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.* In this section, 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, following [10]. 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 $\mathcal{P}_n$. Introduce the (counting) *exponential generating function*, or EGF,

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

For instance, the classes $\mathcal{Q}, \mathcal{R}, \mathcal{S}$ of, respectively, all permutations, sorted permutations, and cyclic permutations have EGFs given by $Q(z) = \frac{1}{1-z}$, $R(z) = e^z$, $S(z) = \log \frac{1}{1-z}$, since $Q_n = n!$, $R_n = 1$, $S_n = (n-1)!$, for $n \geq 1$. Observe that the class $\mathcal{S}$ of cyclic permutation is isomorphic to the class of permutations such that the maximum occurs in the first position: $U_1 > U_2, \dots, U_N$. (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 $\mathbf{U} = (U_1, \dots, U_n)$ be a vector of real numbers. By replacing each U_j by its rank in $\mathbf{U}$, we obtain a permutation $\sigma = (\sigma_1, \dots, \sigma_n)$, which is called the (order) *type* of $\mathbf{U}$. For instance, $U = (1.41, 0.57, 3.14, 2.71)$ has order type $\sigma = (2, 1, 4, 3)$. We shall write type($\mathbf{U}$) for the type of the vector $\mathbf{U}$. The von Neumann schema, relative to a class $\mathcal{P}$ of permutation is described in Figure 2. Observe that it only needs a geometric generator $\Gamma\mathrm{G}(\lambda)$, hence, eventually, only a Bernoulli generator $\Gamma\mathrm{B}(\lambda)$. (The latter can result either from a basic generator that has the parameter λ presented in binary on an auxiliary input tape or be itself constructed by composition rules such as those of Subsection 1.)

First, by construction, a value N is, at each stage of the iteration, chosen with probability $(1-\lambda)\lambda^N$. An iteration that succeeds then returns the value $N = n$ with probability

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

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

(7)

permutations ($\mathcal{P}$):	all ($\mathcal{Q}$)	sorted ($\mathcal{R}$)	cyclic ($\mathcal{S}$)
distribution:	$(1-\lambda)\lambda^n$ geometric	$e^{-\lambda}\dfrac{\lambda^n}{n!}$ Poisson	$\dfrac{1}{L}\dfrac{\lambda^n}{n}$, $L := \log(1-\lambda)^{-1}$ logarithmic.

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

Next, implementation details should be discussed. At the most general level, 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* [16, 28] and is summarized by the command "**set** $\tau := \mathrm{trie}(U)$" in the schema of Figure 2. 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. Furthermore, in the case of the distributions of (7), one never needs to store the entire digital tree: it suffices to keep a single string register that preserves the value of current interest. For instance, for the logarithmic series distribution only the value of U_1 ever needs to be preserved; for the Poisson distribution, only the value of a "current" U_k has to be maintained at any given time.

Finally, we need to discuss 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

$$\omega(\tau) = |\tau| + \omega(\tau_0) + \omega(\tau_1), \qquad |\tau| \geq 2, \tag{8}$$

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 [16, pp. 128–134]; see also the books of Mahmoud [22] and Szpankowski [28] as well as Vallée *et al.*'s analyses under dynamical source models [3, 29]. It is known that the expectation of path length, for n uniform infinite 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 [8, 22, 28]): $\mathbb{E}_n[\omega] = n \log_2 n + O(n)$.

The distribution of path length is known to be asymptotically Gaussian, as $n \to \infty$, after Jacquet and Régnier [13] and Mahmoud *et al.* [21]; see also [20, §11.2] for an exposition. For our purposes, it suffices to observe 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; $h_n(q) := n![z^n]H(z,q)$, namely, for $n \geq 2$,

$$h_n(q) = \frac{1}{1 - q^n 2^{1-n}} \sum_{k=1}^{n-1} \frac{1}{2^n} \binom{n}{k} h_k(q) h_{n-k}(q). \tag{9}$$

We can now summarize the foregoing discussion and state:

Proposition 1. (*i*) *Given a class $\mathcal{P}$ of permutations and a parameter $\lambda \in (0,1)$, the von Neumann schema $\Gamma\mathrm{VN}[\mathcal{P}](\lambda)$ 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 $1 + \mathrm{Geo}(s)$.*

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

$$\mathbb{E}(q^C) = \frac{H^+(\lambda, q)}{1 - H^-(\lambda, q)}, \tag{10}$$

where H^+, H^- are determined by (9)*:*

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

The distribution of C has exponential tails.

Proof. 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 serves to justify in particular (10). □

A simple consequence of Proposition 1 and (7) is the possibility of generating a Poisson or logarithmic variate by a simple device: only *one branch* of the trie needs to be maintained.

Theorem 2. *The Poisson and logarithmic distributions of parameter $\lambda \in (0,1)$ have a strong simulation by a Buffon machine that 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 [17, p. 426]. We may also stress here that the distributions of costs are easily computable: with a symbolic manipulation system, such as Maple, the cost of generating a Poisson(1/2) variate is found to have probability generating function (Item (*iii*) of Proposition 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 [6, 26].

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 Proposition 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., [10, 11, 27].) Note that the property of being alternating can 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 3. *The following functions admit a strong simulation:*

$$\begin{array}{l} e^{-x},\ e^{x-1},\ (1-x)e^x,\ xe^{1-x}, \\ \dfrac{x}{\log(1-x)^{-1}},\ \dfrac{1-x}{\log(1/x)},\ (1-x)\log\dfrac{1}{1-x},\ x\log(1/x), \\ \dfrac{1}{\cos(x)},\ x\cot(x),\ (1-x)\cos(x), (1-x)\tan(x). \end{array}$$

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,

$$\mathrm{Li}_r(z) := \sum_{n=1}^{\infty} \frac{z^n}{n^r},$$

with $r \in \mathbb{Z}_{\geq 1}$. Of course, $\mathrm{Li}_1(1/2) = \log 2$. There are a few special evaluations known for polylogarithmic values, when $r \geq 2$ (see the books by Berndt on Ramanujan [2, Ch. 9] and by Lewin [18, 19]). Of special interest for us are the evaluations

$$\mathrm{Li}_2(1/2) = \frac{\pi^2}{12} - \frac{1}{2}\log^2 2, \qquad \mathrm{Li}_3(1/2) = \frac{1}{6}\log^3 2 - \frac{\pi^2}{12}\log 2 + \frac{7}{8}\zeta(3). \tag{11}$$

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 [1] 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 $R \overset{?}{=} 0$. 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 $\mathrm{Geo}(\lambda)$. Assume we flip $2N$ coins and return a success (1), 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*[2]:

```
(12)   $\Gamma\mathrm{B}\left(\sqrt{1-\lambda}\right)$ :=  { let $N := \Gamma\mathrm{G}(\lambda)$;
                 draw $X_1, \ldots, X_{2N}$ with $X_j \in \{-1, +1\}$ and $\mathbb{P}(+1) = \mathbb{P}(-1) = \frac{1}{2}$;
                 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 4. *The square-root construction of Equation* (12) *provides an exact Bernoulli generator of parameter $\sqrt{1-\lambda}$, given a $\Gamma\mathrm{B}(\lambda)$. The mean number of coin flips required, not counting the ones involved in the calls to $\Gamma\mathrm{B}(\lambda)$, is $\frac{2\lambda}{1-\lambda}$. The function $\sqrt{1-x}$ 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. If we expand the code of the $\Gamma\mathrm{G}(\lambda)$ generator that produces N, we obtain the equivalent sequence of instructions

```
$\Gamma\mathrm{B}\left(\sqrt{1-\lambda}\right)$ := { 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.*

[2] Similar generators have been first developed by Wästlund [30] and by Mossel–Peres [23].

```
{ let $N := \Gamma\mathrm{G}(\lambda)$;
draw $w := X_1X_2\cdots X_N$ with $X_j \in \{H,T\}$ and $\mathbb{P}(H)=\mathbb{P}(T)=\frac{1}{2}$;
if $w \in \mathbf{L}(G;\mathcal{S})$ then return(1) else return(0) }.
```

FIGURE 3. The algebraic construction associated to the pushdown automaton arising from a bistoch grammar.

Note 1 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 [10, §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, $S(z) = \frac{1}{1-3zY(z)^2}$, 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 $\Delta := \Delta + (t-1)$.

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 [10] for background (the Chomsky–Schützenberger Theorem).

Definition 2. *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*

$$\mathcal{X} \longrightarrow H\mathfrak{m} + T\mathfrak{n}, \tag{13}$$

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 that each of them produces at least one word of $\{H,T\}^\star$. For instance, the one-terminal grammar $\mathcal{Y} = H\mathcal{YYY} + T$ generates all Łukasiewicz codes of ternary trees [10, §I.5.3] and is closely related to the previously seen construction.

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

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

where S_n is the number of words of length n in $\mathbf{L}[G;\mathcal{S}]$. The deterministic character of a bistoch grammar implies that the couting OGFs are bound by a system of equations (one for each non-terminal) deduced directly from (13), $X(z) = z\widehat{\mathfrak{m}} + z\widehat{\mathfrak{n}}$, where $\widehat{\mathfrak{m}}, \widehat{\mathfrak{n}}$ are monomials in the OGFs corresponding to the non-terminals of $\mathfrak{m}, \mathfrak{n}$; see [10, §I.5.4]. For instance we have, 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 can be reduced to a single polynomial equation $P(z,y) = 0$. We obtain, with a simple proof, a result of Mossel-Peres [23, Th. 1.2]:

Theorem 5 ([23])**.** *To each bistoch grammar G and non-terminal $\mathcal{S}$, there corresponds a construction (Figure 3), which can be implemented by a deterministic pushdown automaton and calls* *to a* $\Gamma\mathrm{B}(\lambda)$ *and is of type* $\Gamma\mathrm{B}(\lambda) \longrightarrow \Gamma\mathrm{B}\left(S\left(\frac{\lambda}{2}\right)\right)$, *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 of the resulting object (n). 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 their coefficients and singularities, we refer to the discussion of the Drmota–Lalley–Woods Theorem in [10, pp. 482–493] and references therein. Regarding the values of the generating functions, we mention the studies by Kiefer, Luttenberger, and Esparza [14] and by Pivoteau, Salvy, and Soria [25]. The former is motivated by the probabilistic verification of recursive Markov processes, the latter by the efficient implementation of Boltzmann samplers.

It is not known however at this moment 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 for a wide class of positive polynomial systems.)

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 [12] 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 the value 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 flippings;
         if ($\Delta \equiv$ # Heads $-$ # Tails) $\neq 0$ then return(0);
S5.  return(1).
```

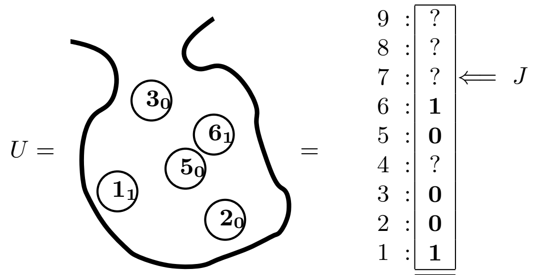
Observe that setting up the generation of a variable that is geometric with parameter $\frac{1}{4}$ necessitates 2 coin flips and then 2 coin flips per incrementation in the worst case (in fact $1\frac{1}{2}$ coin flip on average); drawing a Bernoulli variable with probability $\frac{5}{9}$ requires about 4 coin flips on average—this crude analysis takes care of steps **S1** and **S2**. The mean value of S is $\frac{11}{9} \doteq 1.222$, so that typical values for $2S$ are in the set $\{0, 2, 4, 6, 8\}$ (about 95% of the time). Finally, the for-loop of Step **S3.** is aborted as soon as the number of heads (or tails) differs from S, in which case the procedure returns 0, which means "failure". It is only when the battery of the three tests in **S4** is successful that the procedure terminates returns the value 1, meaning "success" (in **S5**). On average, 9.8 coin flips are consumed by this experiment. (Similar hypergeometric constants can be produced by such devices.)

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

$$\Phi(\lambda) = \frac{1}{\lambda}\int_0^\lambda \phi(w)\,dw. \tag{14}$$

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



```
Ghalf:=proc() local K;
# a geometric RV of param. 1/2
      K:=-1; do K:=K+1; if flip()=0
                then return(K) fi; od;

bag:=proc(U)  local J;
           J:=1+Ghalf();
           if type(U[J],name)
              then U[J]:=flip(); fi;
           return(U[J]); end;
```

FIGURE 4. The "geometric-bag" procedure *bag*(U): two graphic representations of a state (the pairs index–values and a partly filled register) and the complete MAPLE code.

To start with, we discuss a purely discrete implementation of $\Gamma B(\lambda) \longrightarrow \Gamma B(U\lambda)$, with $U \in [0,1]$, uniformly, where multiple invocations of $\Gamma B(\lambda)$ 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 $\Gamma B(U)$ and then appeal to the conjunction (product) construction to get a $\Gamma B(\lambda U)$ as $\Gamma B(U) \cdot \Gamma B(\lambda)$. 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 $\Gamma B(U)$ is simply obtained by fetching the bit of U that is of order J, where $J \in 1 + \mathrm{Geo}(\frac{1}{2})$. In our relaxed lazy context, whenever such a bit υ_j is fetched, we first examine 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 υ_j, and return the corresponding value: see Figure 4. In a language like MAPLE, where it is possible to test whether a variable has been assigned, the implementation is obvious; otherwise, one should maintain an association list of the already "known" indices and values, and update it, as the need arises.

Assume that $\phi(\lambda)$ is realized by a Buffon machine that calls a Bernoulli generator $\Gamma B(\lambda)$. If we replace $\Gamma B(\lambda)$ by $\Gamma B(\lambda U)$, 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 (14). In summary:

Theorem 6. *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. If in addition $\phi(\lambda)$ is analytic at 0, then its integral is strongly realizable.*

This result paves the way to a large number of derived constructions. For instance, starting from the parity construction, 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 $\Gamma B(p)$ is replaced by the product $\Gamma B(p) \cdot \Gamma B(p)$, 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 multiplication by λ) 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 7. *The following functions are strongly realizable* $(0 < x < 1)$*:*

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

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 computers for π. Of special interest, in connection with Φ_1, is the fact that $\Phi_1(1) = \arctan(1) = \dfrac{\pi}{4}$. This gives us a Buffon computer for $\pi/4$. There are even further simplifications

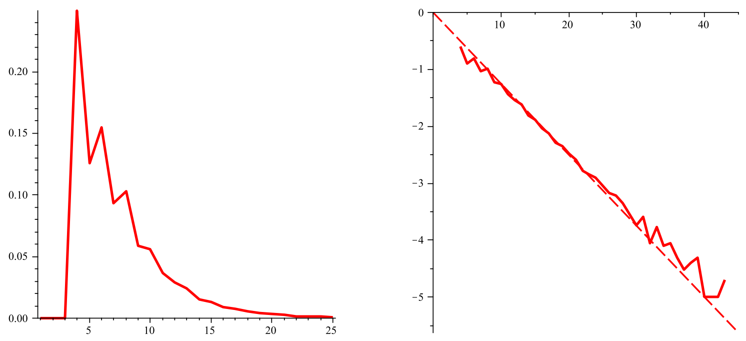


FIGURE 5. The distribution of costs of the Machin machine (15). *Left*: histogram. *Right*: decimal logarithms of the probabilities, compared to $\log_{10}(10^{-k/8})$ (dashed line).

due to the fact that $\Gamma B(1)$ 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 $\pi/4 \doteq \mathbf{0.78}539$. In that particular run, the mean number of flips per experiment was about 27. However, values of U very close to 1 are occasionally generated (the more so, as the number of simulation increases), and in fact *the expected number of flips is infinite*. This unwanted feature is somehow a reflection of the fact that, behind the identity $\pi/4 = \arctan(1)$, there is lurking the slowly convergent series representation (Madhava–Gregory–Leibniz): $\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$.

The next idea is to consider formulae of a kind made well-known by Machin, who made use of arc-tangent 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

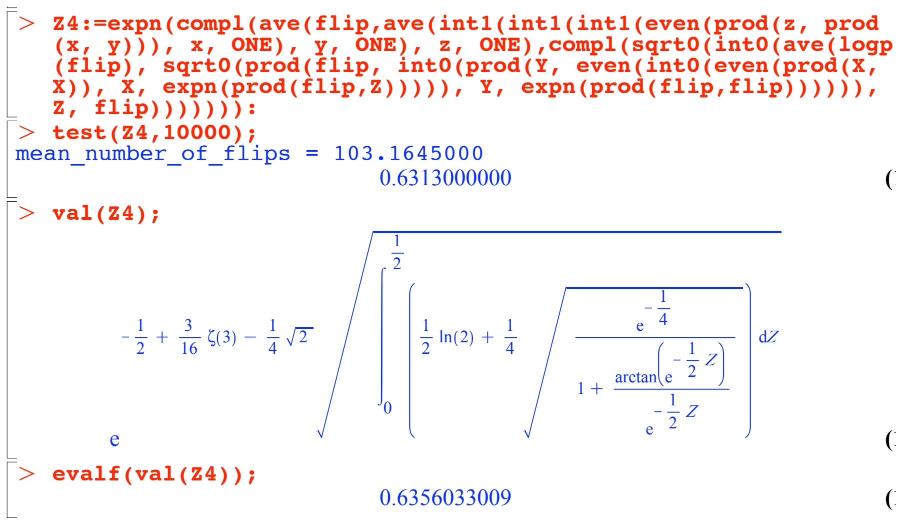
$$\frac{\pi}{4} = \arctan\left(\frac{1}{2}\right) + \arctan\left(\frac{1}{3}\right), \tag{15}$$

being especially suitable for a short program and is easily compiled *in silico*. With 10^6 simulations, we obtained an estimate $\pi/4 \approx \mathbf{0.785}98$, to be compared to the exact value $\pi/4 = \mathbf{0.785}39\cdots$; that is, an error of about $0.5 \cdot 10^{-4}$, well within normal statistical fluctuations. The empirically measured average number of flips per generation of this Machin-like $\Gamma B(\pi/4)$ turned out to be about 6.45 coin flips. Figure 5 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 $\mathbb{P}(C = k) \approx 10^{-k/8}$.

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

5. Experiments and conclusions

We have built a complete protype implementation under the MAPLE symbolic manipulation system, in order to test and validate the ideas expounded above; see Figure 6. A generator such as $Z4$ 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 intructions for each construction. In accordance with the general conditions of our contract, only simple register manipulations are ever used, so that a transcription in C or Java would be of a comparable size.



```
> Z4:=expn(compl(ave(flip,ave(int1(int1(int1(even(prod(z, prod
  (x, y))), x, ONE), y, ONE), z, ONE),compl(sqrt0(int0(ave(logp
  (flip), sqrt0(prod(flip, int0(prod(Y, even(int0(even(prod(X,
  X)), X, expn(prod(flip,Z))))), Y, expn(prod(flip,flip)))))),
  Z, flip))))))):
> test(Z4,10000);
mean_number_of_flips = 103.1645000
                                  0.6313000000
> val(Z4);
```

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

```
> evalf(val(Z4));
                                  0.6356033009
```

FIGURE 6. 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 ⋯ .

We see here that even a complicated constant such as the "value" of the probability associated with $Z4$ is effectively simulated with an error of $0.4\,10^{-3}$, which is consistent with normal statistical fluctuations. For such a complicated constant, the observed mean number of flips is here 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., even(f) $\equiv \frac{1}{1+f}$), we readily program an arc-tangent as a one-liner,

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

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 9 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:

(16)

$\mathrm{Li}_2(1/2)$	Rama	arcsin $[1; \frac{1}{\sqrt{2}}; \frac{1}{2}]$			arctan $[1/2+1/3; 1]$		$\zeta(4)$	$\zeta(2)$
$\frac{\pi^2}{24}$	$\frac{1}{\pi}$	$\frac{\pi}{4}$	$\frac{\pi}{8}$	$\frac{\pi}{12}$	$\frac{\pi}{4}$	$\frac{\pi}{8}$	$\frac{7\pi^4}{720}$	$\frac{\pi^2}{12}$
7.9	10.8	76.5 (∞)	16.2	4.9	4.5	26.7 (∞)	6.2	7.2.

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

References

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

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

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

[10] Flajolet, P., and Sedgewick, R. *Analytic Combinatorics.* Cambridge University Press, 2009. 824 pages (ISBN-13: 9780521898065); also available electronically from the authors' home pages.

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

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

[13] 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.

[14] 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.

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

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

[17] 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.

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

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

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

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

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

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

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

[25] 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.

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

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

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

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

[30] Wästlund, J. Functions arising by coin flipping. Technical Report, 1999.

[31] 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.