# Requests and Open Questions

This version of the document is dated 2021-01-17.

**Peter Occil**

This page lists questions and issues relating to my articles posted on this site. Any answers to these questions will greatly improve those articles. If you can answer any of them, post an issue in the **GitHub issues page**.

# 1 Contents



# 2 Randomization and Sampling Methods

**Size Reduction Sought:**

Of the articles in this repository, **Randomization and Sampling Methods** and **More Random Sampling Methods** combined are very long (about 230 KB in size combined).

These articles describe numerous algorithms to generate random variates (from discrete and continuous distributions) as well as perform random sampling with and without replacement, shuffling, geometric sampling, and more, assuming a source of "truly" random numbers is available.

I would like to reduce the size of these articles while maintaining the most relevant algorithms for random variate generation.

Here are my goals for both articles:

- To shorten the **Randomization with Real Numbers** section as much as possible, while still describing the most general (and exact) algorithms possible for sampling real numbers of any distribution.
- To put emphasis on algorithms that work with random integers (or, if necessary, rational numbers), rather than random floating-point numbers.
- To put emphasis on algorithms that sample a distribution *exactly*, or at least with a controlled upper bound on the error. For discussion, see "**Exact, Error-Bounded, and Approximate Algorithms**".
- To ensure the documents are easy for programmers to understand and implement.

**Other questions:**

- Is there any non-trivial use of random fixed-point numbers in any applications, other than uniformly distributed numbers?

# 3 Bernoulli Factory Algorithms

**https://peteroupc.github.io/bernoulli.html**

This is a page showing algorithms to turn a coin with an unknown probability of heads into a coin with a different probability of heads, also known as *Bernoulli factories*. A *factory function* is a function that relates the old probability to the new one. Roughly speaking, a function can be a factory function only if it maps the interval [0, 1] to the interval [0, 1], is continuous, and doesn't touch 0 or 1 except possibly at the endpoints (Keane and O'Brien 1994)**(1)**.

Attention is drawn to the requests and open questions on that page:

- **https://peteroupc.github.io/bernoulli.html#Requests_and_Open_Questions**

Among other things, they relate to finding polynomial sequences, probabilities, and other mathematical constructions needed to apply certain Bernoulli factories. These questions are reproduced below.

1. Let a permutation class (such as numbers in descending order) and two probability distributions D and E be given. Consider the following algorithm: Generate a sequence of i.i.d. random numbers (where the first is distributed as D and the rest as E) until the sequence no longer follows the permutation class, then return *n*, which is how many numbers were generated this way, minus 1. In this case:

    1. What is the probability that *n* is returned?
    2. What is the probability that *n* is odd or even or belongs to a certain class of numbers?
    3. What is the distribution function (CDF) of the first generated number given that *n* is odd, or that *n* is even?

    See also "**Probabilities Arising from Certain Permutations**" and my Stack Exchange question **Probabilities arising from permutations**.

2. I request expressions of mathematical functions that can be expressed in any of the following ways:

    - Series expansions for continuous functions that equal 0 or 1 at the points 0 and 1. These are required for Mendo's algorithm for **certain power series**.
    - Series expansions for alternating power series whose coefficients are all in the interval [0, 1] and form a nonincreasing sequence. This is required for another class of power series.
    - Series expansions with non-negative coefficients and for which bounds on the truncation error are available.
    - Upper and lower bound approximations that converge to a given constant or function. These upper and lower bounds must be nonincreasing or nondecreasing, respectively.
    - To apply the algorithms for **general factory functions**, what is needed are two sequences of polynomials in Bernstein form, one of which converges from above

to a given function, the other from below. These sequences must be nonincreasing or nondecreasing, respectively, and the polynomials must be of increasing degree and have Bernstein coefficients that are all rational numbers lying in [0, 1], but the polynomials in each sequence may start closer to the function at some points than at others.

Especially helpful would be an automated procedure to compute such sequences, in terms of their Bernstein coefficients, for a large class of factory functions (such as min($\lambda$, $c$) where $c$ is a constant in (0, 1)). (This is in the sense that when given only information about the desired function, such as the coordinates of the function's piecewise linear graph, the procedure can automatically compute the appropriate sequences without further user intervention.)

I have found **several methods** to compute such sequences, but most of them have issues that I seek clarification on. For example, the method of Holtz et al. (2011)**(2)** requires knowing the function's smoothness class and requires the function to be bounded away from 0 and 1; moreover the method uses several constants, namely $s$, $\theta_\alpha$, and $D$, with no easy lower bounds. As another example, Gal's method (1989)**(3)** produces polynomials that converge too slowly to be practical.

An intriguing suggestion from Thomas and Blanchet (2012)**(4)** is to use multiple pairs of polynomial sequences that converge to $f$, where each pair is optimized for particular ranges of $\lambda$: first flip the input coin several times to get a rough estimate of $\lambda$, then choose the pair that's optimized for the estimated $\lambda$, and run either algorithm in this section on that pair. The paper gives the example of min($\lambda$, 8/10). Are there formulas for computing these sequences efficiently, unlike the paper's approach that requires computing an intersection of a curve with an approximating polynomial, which gets very inefficient as the polynomial's degree gets large?

See also my questions on *Mathematics Stack Exchange*:

- **Computing converging polynomials**.
- **Bounds of Bernstein coefficients**.

- Simple **continued fractions** that express useful constants.

All these expressions should not rely on floating-point arithmetic or the direct use of irrational constants (such as $\pi$ or sqrt(2)), but may rely on rational arithmetic. For example, a series expansion that *directly* contains the constant $\pi$ is not desired; however, a series expansion that converges to a fraction of $\pi$ is.

3. Is there a simpler or faster way to implement the base-2 or natural logarithm of binomial coefficients? See the example in the section "**Certain Converging Series**".

4. According to (Mossel and Peres 2005)**(5)**, a pushdown automaton can take a coin with unknown probability of heads of $\lambda$ and turn it into a coin with probability of heads of $f(\lambda)$ only if $f$ is a factory function and can be a solution of a polynomial system with rational coefficients. (See "**Certain Algebraic Functions**".) Are there any results showing whether the converse is true; namely, can a pushdown automaton simulate *any* $f$ of this kind? Note that this question is not quite the same as the question of which algebraic functions can be simulated by a context-free grammar (either in general or restricted to those of a certain ambiguity and/or alphabet size), and is not quite the same as the question of which *probability*

*generating functions* can be simulated by context-free grammars or pushdown automata, although answers to those questions would be nice. (See also Icard 2019[(6)]. Answering this question might involve ideas from analytic combinatorics; e.g., see the recent works of Cyril Banderier and colleagues.)

# 4 Partially-Sampled Random Numbers for Accurate Sampling of Continuous Distributions

**https://peteroupc.github.io/exporand.html**

A *partially-sampled random number* (PSRN) is a data structure holding the initial digits of a random number that is built up digit by digit. There are some open questions on PSRNs.

1. Are there constructions for PSRNs other than for cases given earlier in this document? (The constructions include uniform PSRNs, where the digits are generated uniformly at random; as well as exponential PSRNs or e-rands, where the PSRN follows an exponential distribution.)
2. Doing an arithmetic operation between two PSRNs is akin to doing an interval operation between those PSRNs, since a PSRN is ultimately a random number that lies in an interval. However, as explained in "**Arithmetic and Comparisons with PSRNs**", the result of the operation is an interval that bounds a random number that is *not* always uniformly distributed in that interval. For example, in the case of addition this distribution is triangular with a peak in the middle, and in the case of multiplication this distribution resembles a trapezoid. What are the exact distributions of this kind for other interval arithmetic operations, such as division, ln, exp, sin, or other mathematical functions?
3. Are the conjectures in the section "**Setting Digits by Digit Probabilities**" true? See also my Stack Exchange question **On random variables made up of independent random digits**.

# 5 More Algorithms for Arbitrary-Precision Sampling

**https://peteroupc.github.io/morealg.html**

This page has more algorithms for sampling using partially-sampled random numbers, as well as more Bernoulli factory algorithms. The following are requests and open questions for this article.

1. We would like to see new implementations of the following:
    - Algorithms that implement **InShape** for specific closed curves, specific closed surfaces, and specific signed distance functions. Recall that **InShape** determines whether a box lies inside, outside, or partly inside or outside a given curve or surface.
    - Descriptions of new arbitrary-precision algorithms that use the skeleton given in the section "Building an Arbitrary-Precision Sampler".
2. The appendix contains implementation notes for **InShape**, which determines whether a box is outside or partially or fully inside a shape. However, practical implementations of **InShape** will generally only be able to evaluate a shape pointwise. What are necessary and/or sufficient conditions that allow an implementation to correctly classify a box just by evaluating the shape pointwise?

See also my related Stack Exchange question: **How can we check if an arbitrary shape covers a box (partially, fully, or not) if we can only evaluate the shape pointwise?**.

3. Take a polynomial $f(\lambda)$ of even degree $n$ of the form choose$(n,n/2)*\lambda^{n/2}(1-\lambda)^{n/2}*k$, where $k$ is greater than 1 (thus all $f$'s Bernstein coefficients are 0 except for the middle one, which equals $k$). Suppose $f(1/2)$ lies in the interval (0, 1). If we do the degree elevation, described in the appendix, enough times (at least $r$ times), then $f$'s Bernstein coefficients will all lie in [0, 1]. The question is: at least how many degree elevations are needed? I conjecture that floor$(n/3)+1$ elevations are enough, since experiments show that $r/n$ tends to 1/3 as $n$ gets large.

# 6 Color Topics for Programmers

**https://peteroupc.github.io/colorgen.html**

Should this document cover the following topics, and if so, how?

- The CAM02 color appearance model.
- Color rendering metrics for light sources, including color rendering index (CRI) and the metrics given in TM-30-15 by the Illuminating Engineering Society.

Does any of the following exist?

- A method for performing color calibration and color matching using a smartphone's camera and, possibly, a color calibration card and/or white balance card, provided that method is not covered by any active patents or pending patent applications.
- Reference source code for a method to match a desired color on paper given spectral reflectance curves of the paper and of the inks being used in various concentrations, provided that method is not covered by any active patents or pending patent applications.

# 7 Notes


- (1) Keane, M. S., and O'Brien, G. L., "A Bernoulli factory", ACM Transactions on Modeling and Computer Simulation 4(2), 1994.
- (2) Holtz, O., Nazarov, F., Peres, Y., "New Coins from Old, Smoothly", *Constructive Approximation* 33 (2011).
- (3) Gal, S.G., "Constructive approximation by monotonous polynomial sequences in LipMα, with α ∈ (0, 1]", *Journal of Approximation Theory* 59 (1989).
- (4) Thomas, A.C., Blanchet, J., "**A Practical Implementation of the Bernoulli Factory**", arXiv:1106.2508v3 [stat.AP], 2012.
- (5) Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724.
- (6) Icard, Thomas F., "Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy." Journal of Mathematical Psychology 95 (2020): 102308.


# 8 License