Requests and Open Questions

This version of the document is dated 2021-04-11.

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 continuous probability distributions D and E be given. Consider the following algorithm: Generate a sequence of independent 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?

    Obviously, these answers depend on the specific permutation class and/or distributions D and E. Thus, answers that work only for particular classes and/or distributions are welcome. See also 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.
    - A series expansion with non-negative terms that can be "tucked" under a discrete probability mass function.
    - Series expansions for alternating power series whose coefficients are all in the interval [0, 1] and form a nonincreasing sequence.
    - 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.

These upper and lower bounds must be nonincreasing or nondecreasing, respectively.

- Sequences of approximating functions (such as rational functions) that converge from above and below to a given function. These sequences must be nonincreasing or nondecreasing, respectively (but the approximating functions themselves need not be).

- To apply the algorithms for **general factory functions**, what is needed are two sequences of polynomials written in Bernstein form that converge from above and below to a function as follows: (a) Each sequence's polynomials must have coefficients lying in [0, 1], and be of increasing degree; (b) the degree-n polynomials' coefficients must lie at or "inside" those of the previous upper polynomial and the previous lower one (once the polynomials are elevated to degree n). For a formal statement of these polynomials, see my **question on Mathematics Stack Exchange**. The **supplemental notes** include formulas for computing these polynomials for the vast majority of functions likely to occur in practice, but not all of them.

    - Are there Bernoulli Factory functions used in practice that are not covered by the schemes in the section "Approximation Schemes" in the notes, where suitable polynomials that converge to those functions are built?
    - Are there specific functions (especially those in practical use) for which there are practical and faster formulas for building polynomials that converge to those functions in a manner needed for the Bernoulli factory problem (besides those I list in that section or the main **Bernoulli Factory Algorithms** article)? See the formal statement in the Math Stack Exchange question linked below.

    An intriguing suggestion from Thomas and Blanchet (2012)**(2)** is to use multiple pairs of polynomial sequences that converge to f, where each pair is optimized for particular ranges of λ: first flip the input coin several times to get a rough estimate of λ, then choose the pair that's optimized for the estimated λ, and simulate $f(\lambda)$ using the chosen polynomials. 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?

- 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 π or sqrt(2)), but may rely on rational arithmetic. For example, a series expansion that *directly* contains the constant π is not desired; however, a series expansion that converges to a fraction of π 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. Part of the reverse-time martingale algorithm of Łatuszyński et al. (2009/2011)**(3)** (see "**General Factory Functions**") to simulate a factory function $f(\lambda)$ is as follows. For each n starting with 1:

    1. Flip the input coin, and compute the n^{th} upper and lower bounds of f given the number of heads so far, call them L and U.
    2. Compute the $(n-1)^{\text{th}}$ upper and lower bounds of f given the number of heads so

far, call them L' and U'. (These bounds must be the same regardless of the outcomes of future coin flips, and the interval $[L', U']$ must equal or entirely contain the interval $[L, U]$.)

These parts of the algorithm appear to work for any two sequences of functions (not just polynomials) that converge to f, where L or L' and U or U' are their lower and upper bound approximations. The section on general factory functions shows how this algorithm can be implemented for polynomials. Specifically:

1. Given the number of heads H_n, L is the H_n Bernstein coefficient of the n^{th} lower approximating polynomial, and U is that of the corresponding upper polynomial.
2. L' is the $H_n{}^{\text{th}}$ Bernstein coefficient of the $(n-1)^{\text{th}}$ lower approximating polynomial, and U' is that of the corresponding upper polynomial, after elevating both polynomials to degree n.

But how do these steps work when the approximating functions (the functions that converge to f) are rational functions with integer coefficients? Rational functions with rational coefficients? Arbitrary approximating functions?

5. According to (Mossel and Peres 2005)[(4)], a pushdown automaton can take a coin with unknown probability of heads of λ 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[(5)]. 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. The following is an open question on PSRNs.

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?

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 $\text{choose}(n,n/2)*\lambda^{n/2}*(1-\lambda)^{n/2}*k$, 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: how many degree elevations are enough? A **MathOverflow answer** showed that r is at least $m = (n/f(1/2)^2)/(1-f(1/2)^2)$, but is it true that $\text{floor}(m)+1$ elevations are enough?

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] Thomas, A.C., Blanchet, J., "**A Practical Implementation of the Bernoulli Factory**", arXiv:1106.2508v3 [stat.AP], 2012.
- [3] Łatuszyński, K., Kosmidis, I., Papaspiliopoulos, O., Roberts, G.O., "**Simulating events of unknown probabilities via reverse time martingales**", arXiv:0907.4018v2 [stat.CO], 2009/2011.
- [4] Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724.
- [5] Icard, Thomas F., "Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy." Journal of Mathematical Psychology 95 (2020): 102308.

8 License