Other Open Questions on Probability

Peter Occil

This version of the document is dated 2024-12-24.

This page lists certain open questions on probability. Any answers to these questions will greatly improve my articles posted on this site. If you can answer any of them, post an issue in the **GitHub issues page**[1].

1 Contents

2 Additional Requests and Open Questions

Besides the requests and open questions found here, the following pages list others:

- **Open Questions on the Bernoulli Factory Problem**[2]
- **Requests and Open Questions**[3]

3 Probability distributions computable by pushdown automata

https://cstheory.stackexchange.com/questions/50826/probability-distributions-generated-by-pushdown-automata

This question is about generating random variates, in the form of their binary expansions, on restricted computing models. Specifically, the computing model is based on *pushdown automata* (finite-state machines with a stack) that are driven by flips of a coin and generate the binary expansion of a real number. This results in machines called *pushdown generators*, defined next.

[1] https://github.com/peteroupc/peteroupc.github.io/issues
[2] https://peteroupc.github.io/bernreq.html
[3] https://peteroupc.github.io/requests.html

3.1 Pushdown Generators

A *pushdown generator* has a finite set of *states* and a finite set of *stack symbols*, one of which is called EMPTY, and takes either a fair coin or a coin whose probability of heads is unknown. It starts with a given state and its stack starts with EMPTY. On each iteration:

- The automaton flips the coin.
- Based on the coin flip (HEADS or TAILS), the current state, and the top stack symbol, it moves to a new state (or keeps it unchanged), replaces the top stack symbol with zero, one, or two symbols, and *optionally* gives out a base-N digit. Thus, there are three kinds of *transition rules*:
  - (*state*, *flip*, *symbol*) → (*digit*, *state2*, {*symbol2*}): move to *state2*, write *digit*, replace top stack symbol with same or different one.
  - (*state*, *flip*, *symbol*) → (*digit*, *state2*, {*symbol2*, *new*}): move to *state2*, write *digit*, replace top stack symbol with *symbol2*, then *push* a new symbol (*new*) onto the stack.
  - (*state*, *flip*, *symbol*) → (*digit*, *state2*, {}): move to *state2*, write *digit*, *pop* the top symbol from the stack.

In the transition rules above, *digit* is either a base-N digit or the empty string. Also, the machine terminates with probability 0, and rules that would cause the stack to be empty are not allowed. The infinite "output" of the machine is a real number X in the interval [0, 1], in the form of the base-N digit expansion `0.dddddd...`, where `dddddd...` are the digits produced by the machine from left to right.

See also Yao 1985.

A *finite-state generator* (Knuth and Yao 1976) is the special case of a pushdown generator where the probability of heads is 1/2, each digit is either 0 or 1, rules can't push stack symbols, and only one stack symbol is used. (In other words, a finite-state generator is a *finite automaton* driven by fair coin flips.)

3.2 Distributions Computable by Pushdown Generators

The question that interests me is which probability distributions of real numbers can be computed by pushdown generators, and how they can be constructed.

For example, a pushdown generator with a loop that outputs 0 or 1 at an equal chance produces a *uniform distribution*.

In this sense, there are existing results for finite-state generators. For example:

- Let X be the random variable computable by a finite-state generator, and let $PDF(x)$ be the probability density function of X. Then if $PDF(x)$ is infinitely differentiable on the open interval (0, 1), it must be a polynomial with rational coefficients and not equal 0 at any irrational point on (0, 1) (Kindler and Romik 2004).

Also, I believe that the **following results**[4] are true:

1. Suppose a finite-state generator can generate a probability distribution that takes on finitely many values. Then: Each value occurs with a rational probability; and each value is either rational or transcendental. This belief is in view of the results of Adamczewski et al. 2020.
2. If the distribution function generated by a finite-state generator is continuous and algebraic on the open interval (0, 1), then that function is a piecewise polynomial function.

3.3 Questions

The first question is whether the two results at the end of the previous section are true.

[4] https://peteroupc.github.io/bernsupp.html#Finite_State_and_Pushdown_Generators

The following questions ask what kinds of distributions are possible with these generators (both when the coin driving the generators is fair, and when it has an unknown probability):

1. Of the probability distributions that a finite-state generator can generate, what is the exact class of:
   - Discrete distributions (those that cover a finite or countably infinite set of values)?
   - Absolutely continuous distributions (those with a probability density function such as the uniform or triangular distribution)?
   - Singular distributions (covering an uncountable but measure-zero set)?
   - Absolutely continuous distributions with *continuous* density functions?
2. Same question as 1, but for pushdown generators.
3. Of the probability distributions that a pushdown generator can generate, what is the exact class of distributions with piecewise density functions whose pieces are infinitely differentiable? (The answer is known for finite-state generators.)

4 Checking if a shape covers a box

https://math.stackexchange.com/questions/3882545/what-conditions-ensure-that-checking-if-a-shape-covers-a-box-can-be-done-just-by

I have described an **algorithm for generating random points inside an arbitrary shape**[5] (such as a circle, polygon, or an arbitrary closed curve) contained within a box. It involves checking whether the box is outside or partially or fully inside the shape, and then—

- generating a uniform random point inside the box if the box is inside the shape,
- rejecting the box and starting over if the box is outside the shape, and
- subdividing the box, choosing a random subbox, and repeating this process for that subbox otherwise.

This algorithm uses a function called **InShape** that determines whether a shape covers an axis-aligned bounding box. It takes such a bounding box as input and returns—

- *YES* if the box is entirely inside the shape;
- *NO* if the box is entirely outside the shape; and
- *MAYBE* if the box is partly inside and partly outside the shape.

Now, take a particular implementation of **InShape** that has certain knowledge about a particular shape. Assume the following:

- The shape is closed, has nonzero finite volume, and has a boundary of measure zero.
- The **InShape** implementation can determine only *pointwise* whether a point is either outside the shape, or on or inside the shape.
- The **InShape** implementation has access to arbitrary-precision arithmetic, as well as interval arithmetic using arbitrary-precision rational numbers. See **my library**[6], for example.
- Other than this, it doesn't matter how the shape is described – it could be described as a sequence of line segments, curve segments, or both describing the shape's outline; as a signed distance function; as an inequality; as a union or intersection of multiple shapes; etc.

The **InShape** implementation is given an axis-aligned bounding box as input. The goal is to correctly classify the box just by evaluating the shape *pointwise*.

Under certain conditions, this is trivial to do. For example, if the shape is enclosed by a 1×1 rectangle, the point (0, 0) is on the shape, and every horizontal or vertical line crosses the shape (inside the rectangle)

[5] https://peteroupc.github.io/exporand.html#Uniform_Distribution_Inside_N_Dimensional_Shapes
[6] https://github.com/peteroupc/peteroupc.github.io/blob/master/interval.py

at most once (think of one quarter of a circle centered at the origin), then the box can be correctly classified just by checking the point's corners. The algorithm (Algorithm 1) is thus to return—

- *YES* if all the box's vertices are on or inside the shape;
- *NO* if none of the box's vertices are on or inside the shape; and
- *MAYBE* in any other case.

More generally, I believe Algorithm 1 will work if—

- the shape is enclosed by a hypercube $[0,1] \times [0,1] \times ... \times [0,1]$,
- the point $(0,0,...,0)$ is on or inside the shape, and
- every open axis-aligned line segment that begins in one face of the hypercube and ends in another face crosses the shape at most once (an example is one quarter of a circular disk whose center is at $(0,0)$),

Or if—

- the shape is enclosed by a 2-dimensional rectangle $[0,1] \times [0,1]$,
- the line segment $((0,0),(1,1))$ is entirely on or inside the shape,
- the shape is convex and symmetric about that line segment, and
- the box being tested arose out of a recursive subdivision of the 2-dimensional rectangle into smaller boxes with half the size, $\frac{1}{4}$ the size, etc.

However, for more general convex shapes (which are the shapes that I care about most), this is not so easy. For example, if the shape is convex and the point $(0,0)$ is on the shape, the correct algorithm to classify the shape (Algorithm 2) is to return—

- *YES* if all the box's vertices are on or inside the shape;
- *NO* if none of the box's vertices are on or inside the shape *and if the shape's boundary does not intersect the box's boundary*; and
- *MAYBE* in any other case.

This is not so easy because checking whether a box intersects a shape might not be robust especially if the shape is described by an inequality (such as $x^2 + y^2 - 1 <= 0$). Under certain cases, the algorithm might miss an intersection even though it's present. But at least when the shape is convex and when **InShape** uses interval arithmetic and builds one interval for each dimension of the box (here, $[x, x+\epsilon]$ and $[y, y+\epsilon]$), and evaluates the inequality only once with the intervals, **InShape** can still get robust results. In this algorithm (Algorithm 3), **InShape** returns—

- *YES* if the result's upper bound is less than 0;
- *NO* if the result's lower bound is greater than 0; and
- *MAYBE* in any other case.

4.1 Questions

Thus my questions are:

1. What are necessary or sufficient conditions (such as convexity or regularity conditions, or other requirements on the shape) that allow Algorithm 1 to work correctly? Are the sufficient conditions I gave above for this algorithm correct? If so, can they be relaxed?

2. What are necessary or sufficient conditions that allow Algorithm 2 to work correctly, if the **InShape** method can only evaluate the shape point-by-point? In particular, how can Algorithm 2 robustly check for intersections as required to determine whether to return *NO* or *MAYBE*?

3. What are other conditions that allow **InShape** to correctly classify whether a box is outside or on or inside a shape when **InShape** can only evaluate the shape point-by-point, or when **InShape** proceeds as in Algorithm 3?

4. Is it possible (or what additional conditions make it possible) to correctly classify a bounding box as *NO* or *MAYBE*, using only *pointwise* evaluation of the shape, if—

   - the shape is enclosed by a hypercube $[0,1] \times [0,1] \times ... \times [0,1]$,
   - the point $(0,0,...,0)$ is on or inside the shape,
   - the shape is convex, and
   - the box being tested arose out of a recursive subdivision of the hypercube into smaller boxes with half the size, $\frac{1}{4}$ the size, etc.?

   (Note that in this case, classifying a bounding box as *YES* is trivial; just check its four corners. On the other hand, I know that it's not enough to classify the box as *NO* or *MAYBE* this way.)

4.2 Examples

Take the following shapes, all of which are convex and equal 0 at the origin:

- $v^2 - (u/v)^{1.4-1} * exp(-(u/v)) \le 0$ - Ratio-of-uniforms shape for the gamma(1.4) distribution
- $v^2 - exp(-(u/v)^2/2) \le 0$ - Ratio-of-uniforms shape for the normal distribution
- $v^2 - (u/v)^2 * exp(-(u/v)^2/2) \le 0$ - Ratio-of-uniforms shape for the Maxwell distribution

All three shapes don't work under Algorithm 1, but they appear to give correct results under Algorithm 3, even without the intersection checks required by Algorithm 2.

5 Probabilities arising from permutations

https://stats.stackexchange.com/questions/499864/probabilities-arising-from-permutations

Certain interesting probability functions can arise from permutations. For example, permutations that are sorted or permutations that form a cycle.

Inspired by the so-called *von Neumann schema* given in a paper called "**On Buffon machines and numbers**[7]" by Flajolet and colleagues (2010), we can describe the following algorithm. To describe it, the following definition is needed:

- A *permutation class* is a rule that describes how a sequence of numbers must be ordered. The ordering of the numbers is called a *permutation*. Two examples of permutation classes cover permutations sorted in descending order, and permutations whose highest number appears first. When checking whether a sequence follows a permutation class, only less-than and greater-than comparisons between two numbers are allowed.

The algorithm produces a discrete random variate based on a permutation class. Let D and E be absolutely continuous distributions.

1. Create an empty list.
2. If the list is empty, generate a random variate distributed as D. Otherwise, generate a random variate distributed as E. Either way, append the random variate to the end of the list.
3. Let n be the number of items in the list minus 1. If the items in the list do not form a permutation that meets the permutation class's requirements, return n. Otherwise, go to step 2.

If D and E are both uniform(0, 1), this algorithm returns the number n with the following probability:

$$G(n) = (1 - \frac{V(n+1)}{V(n)(n+1)})(1 - \sum_{j=0}^{n-1} G(j))$$

[7] https://arxiv.org/abs/0906.5560

$$= \frac{V(n)(n+1) - V(n+1)}{V(0)(n+1)!},$$

where $V(n) \in (0, n!]$ is the number of permutations of size n that meet the permutation class's requirements. $V(n)$ can be a sequence associated with an *exponential generating function* (EGF) for the kind of permutation involved in the algorithm. (Examples of permutation classes include permutations whose numbers are sorted in descending order, or permutations whose first number is highest.) For example, if we use the class of permutations sorted in descending order, the EGF is $\exp(\lambda)$, so that $V(n) = 1$.

For this algorithm, if D and E are both uniform(0, 1), the probability that the generated n—

- Is odd is $1 - 1/EGF(1)$, or
- is even is $1/EGF(1)$, or
- is less than k is $\frac{V(0)-V(k)/k!}{V(0)}$.

Thus, for example, if we allow sorted permutations, the algorithm returns an odd number with probability that is *exactly* $1 - \exp(-1)$.

Depending on the permutation class, the distributions D and E, and which values of n we care about, different probabilities and different distributions of numbers will arise. For example:

- If the class is sorted permutations, both D and E are the uniform distribution, and given that the return value n is odd, it is known since von Neumann's 1951 algorithm that that number has an exponential distribution limited to the interval [0, 1].
- If the class is sorted permutations, both D and E are arbitrary distributions, and given that the return value n is odd, then Forsythe (1972) and Monahan (1979) have characterized the distribution function of the sequence's first number.

See the tables in my section "**Probabilities Arising from Certain Permutations**[8]" for further examples.

5.1 Questions

For a given permutation class, a given distribution D, and a given distribution E—

- what is the probability that the algorithm will return a particular n?
- what is the probability that the algorithm will return an n that belongs to a particular class of values (such as odd numbers or even numbers)?
- what is the probability that the first number in the sequence is less than x given that the algorithm returns n (or one of a particular class of values of n)?
- what is the probability that the last number in the sequence is less than x given that the algorithm returns n (or one of a particular class of values of n)?

Note that the third part of the question is equivalent to: What is the CDF of the first number's distribution given that n is returned? Similarly for the fourth part of the question.

6 Questions on Estimation Algorithms

https://stats.stackexchange.com/questions/522429/estimating-f-mathbbex-with-a-guaranteed-error-performance

https://stats.stackexchange.com/questions/555066/a-generalized-randomized-mean-estimate-based-on-the-chebyshev-inequality

[8] https://peteroupc.github.io/bernoulli.html#Probabilities_Arising_from_Certain_Permutations

Let X be a random variable that does not take on a single value with probability 1. Let "black-box" i.i.d. sample access to the random variable X be given. Let $f(x)$ be a known function belonging to a given class of functions.

1. Suppose $f(x)$ is continuous, and suppose X is unbounded and meets additional assumptions, such as—

   - being unimodal (having one peak) and symmetric (mirrored on each side of the peak), or
   - following a geometric distribution, or
   - having decreasing or nowhere increasing probabilities,

   or any combination of these. Then, is there an algorithm, besides the algorithm of Kunsch et al. (2019)—

   - whose output is within ϵ of $f(\mathbb{E}[X])$ in terms of absolute error with probability at least 1 minus δ, or
   - whose output has an expected absolute error or mean squared error not more than ϵ,

   where ϵ and δ are user-specified values? (Relative error means $\mathrm{abs}(\widehat{\mu}/f(\mathbb{E}[X]) - 1)$ where $\widehat{\mu}$ is the estimate.)

   Notice that merely having finite moments is not enough (Theorem 3.4, Kunsch et al.). My article on **estimation algorithms**[9] already gives a relative-error algorithm for the geometric distribution in a note.

2. Let M_k be an upper bound on the kth central absolute moment of X, for $k > 1$. **Based on the Chebyshev inequality**[10] (as well as Hickernell et al. 2013; Kunsch et al. 2018), is the mean $\mathbb{E}[X]$ within ϵ of the mean of n i.i.d. samples, where—

   $$n = \left\lceil \frac{M_k}{\delta \epsilon^k} \right\rceil,$$

   with probability at least $1 - \delta$?

   If so: Let $f(x)$ be uniformly continuous on the real line. Let $m(\epsilon)$ be an inverse modulus of continuity of f, that is, a function that satisfies $\mathrm{abs}(f(y) - f(z)) < \epsilon$ whenever $\mathrm{abs}(y - z) < m(\epsilon)$. Then is $f(\mathbb{E}[X])$ within ϵ of the mean of f on n i.i.d. samples, where—

   $$n = \left\lceil \frac{M_k}{\delta (m(\epsilon))^k} \right\rceil,$$

   with probability at least $1 - \delta$? In both questions, ϵ and δ are user-specified values.

3. Let g be a known piecewise continuous function on [0, 1], and suppose X lies on the interval [0, 1]. How can **a Stack Exchange answer**[11] be adapted to g, so that the algorithm estimates $g(\mathbb{E}[X])$ with either a high probability of a "small" absolute error or one of a "small" relative error at all points in [0, 1] except at a "negligible" area around g's discontinuities? Is it enough to replace g with a continuous function f that equals g everywhere except at that "negligible" area? Here, the accuracy tolerances for small error, high probability, and "negligible" area are user-specified. Perhaps the tolerance could be defined as the integral of absolute differences between f and g instead of "negligible area"; in that case, how should the continuous f be built?

4. If X is Bernoulli with unknown mean $0 < \lambda \leq 1$, is the following algorithm an unbiased estimator of $1/\lambda$? Take random variates i.i.d. until a 1 is taken, then count the number of variates taken this way. This question is asked because the results of Jacob and Thiery (2015) don't cover the case of whether a nonnegative unbiased estimator of $f(\mathbb{E}[X])$ exists when $f : (a, b] \to [0, \infty)$ is unbounded, as opposed to when f is bounded or when f's *domain* is unbounded or a *closed* interval.

[9] https://peteroupc.github.io/estimation.html

[10] https://stats.stackexchange.com/questions/555066/a-generalized-randomized-mean-estimate-based-on-the-chebyshev-inequality

[11] https://stats.stackexchange.com/a/523355/296678

7 References

- Hickernell, F.J., Jiang, L., et al., "**Guaranteed Conservative Fixed Width Intervals via Monte Carlo Sampling**[12]", arXiv:1208.4318v3 [math.ST], 2012/2013.
- Kunsch, Robert J., Erich Novak, and Daniel Rudolf. "Solvable integration problems and optimal sample size selection." Journal of Complexity 53 (2019): 40-67.
- Forsythe, G.E., "Von Neumann's Comparison Method for Random Sampling from the Normal and Other Distributions", *Mathematics of Computation* 26(120), October 1972.
- Monahan, J. "Extensions of von Neumann's method for generating random variables." Mathematics of Computation 33 (1979): 1065-1069.
- Knuth, Donald E. And Andrew Chi-Chih Yao. "The complexity of nonuniform random number generation", in Algorithms and Complexity: New Directions and Recent Results, 1976.
- Vatan, F., "Distribution functions of probabilistic automata", in Proceedings of the thirty-third annual ACM symposium on Theory of computing (STOC '01), pp. 684-693, 2001.
- Kindler, Guy and D. Romik, "On distributions computable by random walks on graphs," SIAM Journal on Discrete Mathematics 17 (2004): 624-633.
- Adamczewski, B., Cassaigne, J. And Le Gonidec, M., 2020. On the computational complexity of algebraic numbers: the Hartmanis–Stearns problem revisited. Transactions of the American Mathematical Society, 373(5), pp.3085-3115.
- Yao, Andrew C. "Context-free grammars and random number generation." In Combinatorial algorithms on words, pp. 357-361. Springer, Berlin, Heidelberg, 1985.
- Jacob, P.E., Thiery, A.H., "On nonnegative unbiased estimators", Ann. Statist., Volume 43, Number 2 (2015), 769-784.

8 License

[12] https://arxiv.org/abs/1208.4318v3

[13] https://creativecommons.org/publicdomain/zero/1.0/