Open Questions on the Bernoulli Factory Problem

This version of the document is dated 2022-07-17.

Peter Occil

1 Background

We're given a coin that shows heads with an unknown probability, λ. The goal is to use that coin (and possibly also a fair coin) to build a "new" coin that shows heads with a probability that depends on λ, call it $f(\lambda)$. This is the *Bernoulli factory problem*, and it can be solved only for certain functions f. (For example, flipping the coin twice and taking heads only if exactly one coin shows heads, we can simulate the probability $2\lambda(1-\lambda)$.)

Specifically, the only functions that can be simulated this way **are continuous and polynomially bounded on their domain, and map $[0, 1]$ or a subset thereof to $[0, 1]$**, as well as $f=0$ and $f=1$. These functions are called *factory functions* in this page. (A function $f(x)$ is *polynomially bounded* if both f and $1-f$ are bounded below by min(x^n, $(1-x)^n$) for some integer n (Keane and O'Brien 1994). This implies that f admits no roots on (0, 1) and can't take on the value 0 or 1 except possibly at 0 and/or 1.)

This page contains several questions about the **Bernoulli factory** problem. Answers to them will greatly improve my pages on this site about Bernoulli factories. If you can answer any of them, post an issue in the **GitHub issues page**.

2 Contents

3 Polynomials that approach a factory function "fast"

https://math.stackexchange.com/questions/3904732/what-are-ways-to-compute-polynomials-that-converge-from-above-and-below-to-a-con

https://mathoverflow.net/questions/424272/explicit-and-fast-error-bounds-for-polynomial-approximation

A polynomial $P(x)$ is written in *Bernstein form of degree n* if it is written as— $$P(x)=\sum_{k=0}^n a_k {n \choose k} x^k (1-x)^{n-k},$$ where $a_0, ..., a_k$ are the polynomial's *Bernstein coefficients*.

An **algorithm** simulates a factory function $f(\lambda)$ via two sequences of polynomials that converge from above and below to that function. Roughly speaking, the algorithm works as follows:

1. Generate U, a uniform random variate in $[0, 1]$.
2. Flip the input coin (with a probability of heads of λ), then build an upper and lower bound for $f(\lambda)$, based on the outcomes of the flips so far. In this case, these bounds come from two degree-n polynomials that approach f as n gets large, where n is the number of coin flips so far in the algorithm.
3. If U is less than or equal to the lower bound, return 1. If U is greater than the upper bound, return 0. Otherwise, go to step 2.

The result of the algorithm is 1 with probability *exactly* equal to $f(\lambda)$, or 0 otherwise.

However, the algorithm requires the polynomial sequences to meet certain requirements; among them, the sequences must be of Bernstein-form polynomials that converge from above and below to a factory function. See the formal statement, next.

3.1 Formal Statement

More formally, for the approximation schemes I am looking for, there exist two sequences of polynomials, namely—

- $g_{n}(\lambda): =\sum_{k=0}^{n}a(n, k){n \choose k}\lambda^{k}(1-\lambda)^{n-k}$, and
- $h_{n}(\lambda): =\sum_{k=0}^{n}b(n, k){n \choose k}\lambda^{k}(1-\lambda)^{n-k}$,

for every integer $n\ge1$, such that—

1. $0\le a(n, k)\le b(n, k)\le1$,
2. $\lim_{n}g_{n}(\lambda)=\lim_{n}h_{n}(\lambda)=f(\lambda)$ for every $\lambda\in[0,1]$, and
3. for every $m<n$, both $(g_{n} - g_{m})$ and $(h_{m} - h_{n})$ have non-negative coefficients once g_{n}, g_{m}, h_{n}, and h_{m} are rewritten as degree-n polynomials in Bernstein form,

where $f(\lambda)$ is continuous on $[0, 1]$ (Nacu and Peres 2005; Holtz et al. 2011), and the goal is to find the appropriate values for $a(n, k)$ and $b(n, k)$.

It is allowed for $a(n, k)\lt 0$ for a given n and some k, in which case all $a(n, k)$ for that n are taken to be 0 instead. It is allowed for $b(n, k)\gt 1$ for a given n and some k, in which case all $b(n, k)$ for that n are taken to be 1 instead.

Alternatively, find a way to rewrite $f(\lambda)$ as— $$f(\lambda) = \sum_{n\ge 1} P_n(\lambda) = 1 - \sum_{n\ge 1} Q_n(\lambda),$$ where P_n and Q_n are polynomials of degree n with non-negative Bernstein coefficients.

3.2 A Matter of Efficiency

However, ordinary Bernstein polynomials can't in general converge to a function faster than $O(1/n)$, a result known since Voronovskaya (1932) and a rate that will lead to an **infinite expected number of coin flips in general**. (See also my **supplemental notes**.)

But Lorentz (1966) showed that if the function is positive and C^k continuous, there are polynomials with non-negative Bernstein coefficients that converge at the rate $O(1/n^{k/2})$ (and thus can enable a **finite expected number of coin flips** if the function is "smooth" enough).

Thus, people have developed alternatives, including iterated Bernstein polynomials, to improve the convergence rate. These include Micchelli (1973), Guan (2009), Güntürk and Li (2021a, 2021b), the "Lorentz operator" in Holtz et al. (2011) (see also "**New coins from old, smoothly**"), and Draganov (2014).

These alternative polynomials usually include results where the error bound is the desired $O(1/n^{k/2})$, but nearly all those results (e.g., Theorem 4.4 in Micchelli; Theorem 5 in Güntürk and Li) have hidden constants with no upper bounds given, making them unimplementable (that is, it can't be known beforehand whether a given polynomial will come close to the target function within a user-specified error tolerance). (**See note 4 in "End Notes".**)

3.3 Questions

Thus the questions are:

1. Are there practical formulas to compute polynomials that—
    - meet the formal statement above, and
    - meet the following error bound? $$|f(x) - P_n(f)(x)| \le \epsilon(f,n,x) = O(1/n^{k/2}),$$ for every $n\ge 1$, where $P_n(f)(x)$ is an approximating degree-n polynomial that can be readily rewritten to Bernstein form (ideally with coefficients in $[0,1]$); k is the number of continuous derivatives; and $\epsilon(f,n,x)$ is a fully determined formula with all constants in the formula having a **known exact value or upper bound**.
2. Are there other practical formulas to approximate specific factory functions with polynomials that meet the formal statement above?
3. Are there practical formulas to compute polynomials that meet the error bound given in question 1 and can be readily rewritten to Bernstein form with coefficients in $[0,1]$?

One example worth pondering is $f(\lambda)=sin(\lambda\pi/2)=cos((1-\lambda)\pi/2)$, which equals 0 at 0 and 1 at 1.

4 New coins from old, smoothly

https://mathoverflow.net/questions/407179/using-the-holtz-method-to-build-polynomials-that-converge-to-a-continuous-functi

Now, we focus on a **specific approximation scheme**, the one **presented by Holtz et al. 2011, in the paper "New coins from old, smoothly"**.

The scheme involves building polynomials that are shifted upward and downward to approximate f from above and below so that the polynomials meet the **Formal Statement** given earlier.

The scheme achieves a convergence rate that generally depends on the smoothness of f; in fact, it can achieve the highest convergence rate possible for functions with that smoothness.

Specifically, Holtz et al. proved the following results:

1. A function $f(\lambda):[0,1]\to(0,1)$ can be approximated, in a manner that solves the Bernoulli factory problem, at the rate $O((\Delta_n(\lambda))^\beta)$ if and only if f is $\lfloor\beta\rfloor$ times differentiable and has a ($\beta-\lfloor\beta\rfloor$)-Hölder continuous $\lfloor\beta\rfloor$-th derivative, where $\beta>0$ is a non-integer and $\Delta_n(\lambda) = \max((\lambda(1-\lambda)/n)^{1/2}, 1/n)$. (Roughly speaking, the rate is $O((1/n)^{\beta})$ when λ is close to 0 or 1, and $O((1/n)^{\beta/2})$ elsewhere.)

2. A function $f(\lambda):[0,1]\to(0,1)$ can be approximated, in a manner that solves the Bernoulli factory problem, at the rate $O((\Delta_n(\lambda))^{r+1})$ only if the rth derivative of f is in the Zygmund class, where $r\ge 0$ is an integer.

The scheme is as follows:

Let f be a function—

- that maps [0, 1] to the open interval (0, 1), and
- whose rth derivative is β-Hölder continuous, where β is in (0, 1).

Let $\alpha = r+\beta$, let $b = 2^s$, and let $s\ge0$ be an integer. Let $Q_{n, r}f$ be a degree $n+r$ approximating polynomial called a *Lorentz operator* (see the paper for details on the Lorentz operator). Let n_0 be the smallest n such that $Q_{n_0, r}f$ has coefficients within [0, 1]. Define the following for every integer $n \ge n_0$ divisible by $n_{0}b$:

- $f_{n_0} = Q_{n_0, r}f$.
- $f_{n} = f_{n/b} + Q_{n, r}(f-f_{n/b})$ for each integer $n > n_0$.
- $\phi(n, \alpha, \lambda) = \frac{\theta_{\alpha}}{n^{\alpha}}+(\frac{\lambda(1-\lambda)}{n})^{\alpha/2}$.

Let $B_{n, k, F}$ be the kth coefficient of the degree-n Bernstein polynomial of F.

Let $C(\lambda)$ be a polynomial as follows: Find the degree-n Bernstein polynomial of $\phi(n, r+\beta, \lambda)$, then elevate it to a degree-$n+r$ Bernstein polynomial.

Then the coefficients for the degree $n+r$ polynomial that approximates f are—

- $g(n, r, k) = B_{n+r,k,f_{n}} - D * B_{n+r,k,C}$, and
- $h(n, r, k) = B_{n+r,k,f_{n}} + D * B_{n+r,k,C}$.

However, the Holtz method is not yet implementable, for the following reasons among others:

- The paper doesn't give values or upper bounds for important constants, notably the three constants s, θ_{α}, and D. For example, the paper says only that D should be chosen "large enough".
- The method's results are only asymptotic.
- The paper has no examples of how the scheme works for a selection of functions f.

And I seek ways to make this solution implementable.

4.1 Questions

1. What are practical upper bounds for s, θ_{α}, and D for the "New coins from old, smoothly" method, given a factory function f, with or without additional assumptions on f (such as smoothness or concavity requirements on f and/or its derivatives, or bounds on the derivatives such as in Gevrey's hierarchy)?

2. Given a continuous function f that maps $[0,1]$ to $(0,1)$, is the "New coins from old, smoothly" method valid in the following cases? (Note that the method as written doesn't apply to non-integer α; see also Conjecture 34 of Holtz et al., 2011, which claims the

converse of the second result given above.)

- With $\alpha=1, r=0$, when f is Lipschitz continuous and/or differentiable.
- With $\alpha=2, r=1$, when f has a Lipschitz continuous first derivative.
- With $\alpha=2, r=2$, when f is twice differentiable.
- With $\alpha=4, r=3$, when f has a Lipschitz continuous third derivative.
- With $\alpha=4, r=4$, when f is four times differentiable.
- With $\alpha=5, r=4$, when f has a Lipschitz continuous fourth derivative.

5 Reverse-time martingales

https://mathoverflow.net/questions/385244/reverse-time-martingale-for-non-polynomial-approximating-functions

One way to toss heads with probability $f(\lambda)$ given a coin that shows heads with probability λ is to build randomized upper and lower bounds that converge to f on average. These bounds serve as an unbiased estimator of $f(\lambda)$; the algorithm returns 1 with probability equal to the estimate, and 0 otherwise.

Part of the *reverse-time martingale algorithm* of Łatuszyński et al. (2009/2011) (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 $(n-1)$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 $[L, U]$.)

More technically (Algorithm 4):

1. Obtain L_n and U_n given $\mathcal{F}_{0, n-1}$,
2. Compute $L^{\star}_{n}$ = $\mathbb{E}(L_{n-1} | \mathcal{F}_{n})$ and $U^{\star}_{n}$ = $\mathbb{E}(U_{n-1} | \mathcal{F}_{n})$,

where $\mathcal{F}_n$ is a filtration that depends on L_n and U_n.

Though the paper as well as the section on general factory functions that I linked to above shows how this algorithm can be implemented for polynomials, these parts of the algorithm appear to work for any two sequences of functions that converge to f, where L or $L^\star$ and U or $U^\star$ are their lower and upper bound approximations. An example for **polynomials** follows:

1. Given the number of heads H_n, L_n is the H_nth Bernstein coefficient of the nth lower approximating polynomial, and U_n is the H_nth Bernstein coefficient of the nth upper approximating polynomial.
2. $L^\star_n$ is the H_nth Bernstein coefficient of the $(n-1)$th lower approximating polynomial, and $U^\star_n$ is the H_nth Bernstein coefficient of the $(n-1)$th upper approximating 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 other than polynomials?** Specifically, what if the approximating functions are rational functions with integer coefficients? rational functions with rational coefficients? arbitrary approximating functions?

6 Tossing Heads According to a Concave Function

https://mathoverflow.net/questions/409174/concave-functions-series-representation-

and-converging-polynomials

6.1 Using Two Polynomial Sequences

If f is a concave factory function, there is a simple way to approximate that function from below with polynomials in Bernstein form. Define—

$$a(n,k) = f(k/n).$$

Then the lower polynomials $g_{n}(\lambda)$ will meet all the requirements of the **formal statement** above. Indeed:

$$g_{n}(\lambda)=\sum_{k=0}^{n}f(k/n){n \choose k}\lambda^{k}(1-\lambda)^{n-k}.$$

The trickier part is thus to approximate $f(\lambda)$ from above (that is, to find the upper polynomials h_n). In this sense I am aware of attempts to do so for specific functions: Nacu & Peres (2005) for linear functions, and Flegal & Herbei (2012) for a twice differentiable elbow function. I am also aware of a result by Holtz et al. 2011, given in the section "New coins from old, smoothly".

I believe a more general solution is to somehow find the maximum difference between $g_{n}(\lambda)$ and $f(\lambda)$, for each n, and to shift the lower polynomial g_n upward by that difference to get the upper polynomial $h_{n}(\lambda)$. I believe this solution works for the function $\min(\lambda,1-\lambda)$ (by getting the maximum difference at $\lambda=1/2$), but I don't know if it works for more general concave functions.

6.2 Using a Series Expansion

There is another way to simulate a concave f. This involves rewriting the concave function as a series expansion as follows:

$$f(\lambda)=\sum_{a\ge 0} \gamma_a(\lambda) = \sum_{a\ge 0} \frac{\gamma_a(\lambda)}{\pi(a, p)} \pi(a, p), \tag{1}$$

where—

- $\gamma_a(\lambda) = g_{n_{a}}(\lambda) - g_{n_{a-1}}(\lambda)$,
- $\pi(a, p) = p (1-p)^a$ is the probability of getting a non-negative integer a in step 1 of the following algorithm,
- g_n is the Bernstein polynomial for f of degree n, with $g_{0} := 0$,
- (n_a) is an increasing sequence of positive integers, with $n_{-1} := 0$,
- p is a rational number in $(0, 1)$, and
- $\frac{\gamma_a(\lambda)}{\pi(a, p)}$, which will be a polynomial, must map $[0, 1]$ to $[0, 1]$, and must not equal 0 or 1 anywhere on $(0, 1)$ unless it's a constant. In the case of concave functions, this polynomial will always be non-negative.

Then an algorithm to toss heads with probability equal to f would be:

1. Flip a coin that shows heads with probability p until that coin shows heads. Set a to the number of tails.
2. Write $\frac{\gamma_a(\lambda)}{\pi(a, p)}$ as a polynomial in Bernstein form of degree n_{a} (or a higher degree such that the Bernstein coefficients are all in [0, 1]). Flip the biased coin (with probability of heads λ) n times, where n is the polynomial's degree, and let j be the number of heads.
3. Return 1 with probability equal to the polynomial's jth Bernstein coefficient, or 0 otherwise (see also Goyal and Sigman 2012 for an algorithm to simulate polynomials).

(This algorithm has similarities to the one used in the proof in Keane and O'Brien 1994.)

However, using this technique for a given concave f requires finding the appropriate sequence for n_a and the appropriate value of p so that the series expansion can be formed. Here is an

example for $\min(\lambda, 1-\lambda)$ which *appears* to be correct:

- $n_a = 2 * 2^a$.
- $p = 0.27$.

Finding these parameters was far from rigorous, though, and the process will have to be repeated for each concave function.

It's also possible for a to be generated in step 1 differently, perhaps with a Poisson distribution. In that case, $\pi(a, p)$ will have to be changed to the corresponding probability of getting a under the new distribution.

Note: Some concave functions can be rewritten as—

$$f(\lambda)=g_{n_k}(\lambda) + \sum_{a\gt k} \frac{\gamma_a(\lambda)}{\pi(a, p)} \pi(a, p), \tag{2}$$

for some integer $k\ge 0$, if they satisfy the series expansion (1) except that $\frac{\gamma_a(\lambda)}{\pi(a, p)}$ is allowed to equal 1 or greater for some p in $(0, 1)$ and some $a\le k$. This way of writing f is acceptable for my purposes.

6.3 Questions

1. Given that a factory function $f(\lambda)$ is concave and C^α continuous, is there a formula to find the amount by which to shift the lower polynomials g_n upward so that the upper polynomials h_n meet the formal statement above (or to otherwise convert the lower polynomials to upper polynomials that meet that statement)? By Holtz's results, this formula would have to behave asymptotically like $O((\Delta_n(\lambda))^\alpha)$, but I am looking for nonasymptotic results that achieve this rate of convergence.
2. Given that a factory function $f(\lambda):[0, 1] \to (0, 1)$ is concave and continuous, is it enough to shift $g_{n}(\lambda)$ upward by the maximum difference between $g_{n}(\lambda)$ and $f(\lambda)$, for each n, to get the corresponding upper polynomial $h_{n}(\lambda)$? If not, for which concave functions does this work?
3. Given that a factory function $f(\lambda):[0, 1] \to [0, 1)$ is concave and continuous, what values of n_a and p will allow that function to have the series expansion (1) or (2)? I suspect that a formula for this question will depend on the smoothness of f, due to Holtz's results.

See also Note 1.

7 Simulable and strongly simulable functions

https://mathoverflow.net/questions/404961/from-biased-coins-and-nothing-else-to-biased-coins

There are two kinds of factory functions:

- A function $f(\lambda)$ is *simulable* if an algorithm exists to toss heads with probability $f(\lambda)$ given a coin with probability of heads λ (the "biased coin") as well as a fair coin.
- A function $f(\lambda)$ is *strongly simulable* if an algorithm exists to toss heads with probability $f(\lambda)$ given **only** a coin with probability of heads λ.

Every strongly simulable function is simulable, but not vice versa.

In fact, Keane and O'Brien (1994) showed already that $f(\lambda)$ is strongly simulable if f is simulable and neither 0 nor 1 is in f's domain (that is, if the biased coin doesn't show heads every time or tails every time). And it's also easy to show that if f is strongly simulable, then $f(0)$ and $f(1)$ must each be 0, 1, or undefined.

However, it's not so trivial to find the exact class of strongly simulable functions when f's domain includes 0, 1, or both.

As one illustration of this, the proof of Keane and O'Brien relies on generating a geometric random variate and using that variate to control which "part" of the target function $f(\lambda)$ to simulate. This obviously works on all of [0, 1] if the algorithm uses both the biased coin and a separate fair coin. However, if only the biased coin is used in the algorithm, the geometric random variate is generated using fair bits via the von Neumann method, which however will never terminate if λ is either 0 or 1.

As another illustration, I managed to find the following **result**:

- If $f(\lambda)$ is a factory function that meets the following conditions, then f is strongly simulable.

    1. $f(0)$ and $f(1)$ must each be 0, 1, or undefined.
    2. If $f(0) = 0$ or $f(1) = 0$ or both, then either f must be constant on its domain or there must be a polynomial $g(\lambda)$ in Bernstein form whose coefficients are computable and in the interval [0, 1], such that $g(0) = f(0)$ and $g(1) = f(1)$ whenever 0 or 1, respectively, is in the domain of f, and such that $g(\lambda)$ > $f(\lambda)$ for every λ in the domain of f, except at 0 and 1.
    3. If $f(0) = 1$ or $f(1) = 1$ or both, then either f must be constant on its domain or there must be a polynomial $h(\lambda)$ in Bernstein form whose coefficients are computable and in the interval [0, 1], such that $h(0) = f(0)$ and $h(1) = f(1)$ whenever 0 or 1, respectively, is in the domain of f, and such that $h(\lambda)$ < $f(\lambda)$ for every λ in the domain of f, except at 0 and 1.

And the proof proceeds by showing, among other things, that the Bernoulli factory for f must flip the input coin and get 0 and 1 before it simulates any fair coin flips via the von Neumann trick.

Question: **Does the result just given describe all the functions that are strongly simulable (using nothing but the biased coin) when the biased coin can show heads every time and/or tails every time? If not, what is the exact class of strongly simulable functions?** Examples of functions to ponder are those that are continuous but not Lipschitz continuous at 0 (and λ can equal 0), such as λ^α where $\alpha \in (0, 1)$, or $\lim_{z\to\lambda} z-z \ln(z)$, or $\lim_{z\to\lambda} -1/(2 \ln(z/2))$.

8 Multiple-Output Bernoulli Factories

https://mathoverflow.net/questions/412772/from-biased-coins-to-biased-coins-as-efficiently-as-possible

Let J be a closed interval on $(0, 1)$, and let $f(\lambda):J \to (0, 1)$ be continuous.

Then by Keane and O'Brien, f admits an algorithm that solves the Bernoulli factory problem for f (using only the biased coin, in fact). A related problem is a Bernoulli factory that takes a coin with unknown probability of heads $\lambda \in J$ and produces *one or more* samples, at a time, of the probability $f(\lambda)$. This question calls it a *multiple-output Bernoulli factory*.

Obviously, any single-output Bernoulli factory can produce multiple outputs by running itself multiple times. But for some functions f, it may be that producing multiple outputs at a time may use fewer coin flips than producing one output multiple times.

Define the entropy bound as— $$h(f(\lambda))/h(\lambda),$$ where— $$h(x)=-x \ln(x)-(1-x) \ln(1-x),$$ is related to the Shannon entropy function.

8.1 Questions

1. Given that a function $f(\lambda)$ is continuous and maps a closed interval in (0, 1) to (0, 1),

is there a multiple-output Bernoulli factory algorithm for f with an expected number of coin flips per sample that is arbitrarily close to the entropy bound, uniformly for every λ in f's domain? Call such a Bernoulli factory an *optimal factory*. (See Nacu and Peres 2005, Question 1.)
2. Does the answer to question 1 change if the algorithm can also use a fair coin in addition to the biased coin?

8.2 Functions with Optimal Factories

So far, the following functions do admit an optimal factory:

- The functions λ and $1-\lambda$.
- Constants in [0, 1]. As Nacu and Peres (2005) already showed, any such constant c admits an optimal factory: generate unbiased random bits using Peres's iterated von Neumann extractor (Peres 1992), then build a binary tree that generates 1 with probability c and 0 otherwise (Knuth and Yao 1976).

It is easy to see that if an optimal factory exists for $f(\lambda)$, then one also exists for $1-f(\lambda)$: simply change all ones returned by the $f(\lambda)$ factory into zeros and vice versa.

Also, as Yuval Peres (Jun. 24, 2021) told me, there is an efficient multiple-output Bernoulli factory for $f(\lambda) = \lambda/2$: the key is to flip the input coin enough times to produce unbiased random bits using his extractor (Peres 1992), then multiply each unbiased bit with another input coin flip to get a sample from $\lambda/2$. Given that the sample is equal to 0, there are three possibilities that can "be extracted to produce more fair bits": either the unbiased bit is 0, or the coin flip is 0, or both are 0. This algorithm, though, might not count as an *optimal factory*, and Peres described this algorithm only incompletely. Indeed, the correctness might depend on how the three possibilities are "extracted to produce more fair bits"; after all, the number of coin flips per sample, for every λ, must not surpass the entropy bound.

In any case, I believe that not all factory functions admit an optimal factory described here; especially because—

- the question may depend on f's range, and
- the efficiency of even a *single-output* Bernoulli factory depends on f's smoothness (e.g., $O(1/n^{(r+\alpha)/2})$ only if f's rth derivative is α-Hölder continuous for some $\alpha \in (0, 1)$; Holtz et al. 2011).

See an **appendix in one of my articles** for more information on my progress on the problem.

9 From coin flips to algebraic functions via pushdown automata

https://cstheory.stackexchange.com/questions/50853/from-coin-flips-to-algebraic-functions-via-pushdown-automata

This section is about solving the Bernoulli factory problem on a restricted computing model, namely the model of *pushdown automata* (finite-state machines with a stack) that are driven by flips of a coin and produce new probabilities.

9.1 Pushdown Automata

A *pushdown automaton* has a finite set of *states* and a finite set of *stack symbols*, one of which is called EMPTY and takes a biased coin with an unknown probability of heads. 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) and replaces the top stack symbol with zero, one, or two symbols. Thus, there are three kinds of *transition rules*:
  - (*state*, *flip*, *symbol*) → (*state2*, {*symbol2*}): move to *state2*, replace top stack symbol with same or different one.
  - (*state*, *flip*, *symbol*) → (*state2*, {*symbol2*, *new*}): move to *state2*, replace top stack symbol with *symbol2*, then *push* a new symbol (*new*) onto the stack.
  - (*state*, *flip*, *symbol*) → (*state2*, {}): move to *state2*, *pop* the top symbol from the stack.

When the stack is empty, the machine stops and returns either 0 or 1 depending on the state it ends up at. (For the questions below, let *flip* be HEADS, TAILS, or a rational number in [0, 1]; this likewise reduces to the definition above. The rest of this question assumes the pushdown automaton terminates with probability 1.)

9.2 Algebraic Functions

Let $f: (0, 1) \to (0, 1)$ be continuous. Mossel and Peres (2005) showed that a pushdown automaton can simulate f only if f is *algebraic over the rational numbers* (there is a nonzero polynomial $P(x, y)$ in two variables and whose coefficients are rational numbers, such that $P(x, f(x)) = 0$ for every x in the domain of f). The algebraic function generated by pushdown automata corresponds to a system of polynomial equations, as described by Mossel and Peres (2005) and Esparza et al. 2004.

Let $\mathcal{C}$ be the class of continuous functions that map (0, 1) to (0, 1) and are algebraic over rationals. The constants 0 and 1 are also in $\mathcal{C}$.

Let $\mathcal{D} \subseteq \mathcal{C}$ be the class of functions that a pushdown automaton can simulate.

I don't yet know whether $\mathcal{D}=\mathcal{C}$ (and that was also a question of Mossel and Peres).

The following section of my open-source page, **https://peteroupc.github.io/morealg.html#Pushdown_Automata_and_Algebraic_Functions,** contains information on the question. As that section shows, I could establish the following about the class $\mathcal{D}$:

- $\sqrt{\lambda}$ is in $\mathcal{D}$, and so is every rational function in $\mathcal{C}$.
- If $f(\lambda)$ and $g(\lambda)$ are in $\mathcal{D}$, then so are their product and composition.
- If $f(\lambda)$ is in $\mathcal{D}$, then so is every Bernstein-form polynomial in the variable $f(\lambda)$ with coefficients in $\mathcal{D}$.
- If a pushdown automaton can generate a discrete distribution of *n*-letter words, then that distribution's probability generating function is in $\mathcal{D}$ (cf. Dughmi et al. 2021).
- If a pushdown automaton can generate a discrete distribution of *n*-letter words of the same letter, it can generate that distribution conditioned on a finite set of word lengths, or a periodic infinite set of word lengths (e.g., odd word lengths only).
- Every quadratic irrational in (0, 1) is in $\mathcal{D}$.

9.3 Questions

1. For every function in class $\mathcal{C}$, is there a pushdown automaton that can simulate that function? (In other words, is $\mathcal{D}=\mathcal{C}$?).
2. In particular, is min(λ, $1-\lambda$) in class $\mathcal{D}$? What about $\lambda^{1/p}$ for some prime $p\ge 3$?

See also Notes 2 and 3.

10 Other Questions

Simple simulation algorithms:

- What simulations exist that are "relatively simple" and succeed with an irrational probability between 0 and 1? What about "relatively simple" Bernoulli factory algorithms for factory functions? Here, "relatively simple" means that the algorithm:
    - Should use only uniform random integers (or bits) and integer arithmetic.
    - Does not use floating-point arithmetic or make direct use of square root or transcendental functions.
    - Should not use rational arithmetic or increasingly complex approximations, except as a last resort.

    See also Flajolet et al., "On Buffon machines and numbers", 2010. There are many ways to describe the irrational probability or factory function. I seek references to papers or books that describe irrational constants or factory functions in any of the following ways:

    - For irrational constants:
        - Simple **continued fraction** expansions.
        - Closed shapes inside the unit square whose area is an irrational number. (Includes algorithms that tell whether a box lies inside, outside, or partly inside or outside the shape.) **Example.**
        - Generate a uniform (x, y) point inside a closed shape, then return 1 with probability x. For what shapes is the expected value of x an irrational number? **Example.**
        - Functions that map [0, 1] to [0, 1] whose integral (area under curve) is an irrational number.
    - Bernoulli factory functions with any of the following series expansions, using rational arithmetic only:
        - Series with non-negative terms where $f(0)$ is 0 and $f(1)$ is rational or vice versa (see "**Certain Power Series**").
        - Series with non-negative terms that can be "tucked" under a discrete probability mass function (see "**Convex Combinations**").
        - Alternating power series (see "**Certain Power Series**").
        - Series with non-negative terms and bounds on the truncation error (see "**Certain Converging Series**").

11 End Notes

Note 1: Besides the questions on concave functions given above, there is also the question of whether the solution terminates with a finite expected running time. In this sense, Nacu & Peres showed that a finite expected time is possible only if f is Lipschitz continuous, and I strongly suspect it's not possible either unless f has a Hölder continuous fourth derivative, in view of the results by Holtz given earlier.

Note 2: On pushdown automata: Etessami and Yannakakis (2009) showed that pushdown automata with rational probabilities are equivalent to recursive Markov chains (with rational transition probabilities), and that for every recursive Markov chain, the system of polynomial equations has nonnegative coefficients. But this paper doesn't deal with the case of recursive Markov chains where the transition probabilities cannot just be rational, but can also be λ and $1-\lambda$ where λ is an unknown rational or irrational probability of heads.

Note 3: On pushdown automata: Banderier and Drmota (2014) showed the asymptotic behavior of power series solutions $f(\lambda)$ of a polynomial system, where both the series and the system have nonnegative real coefficients. Notably, functions of the form $\lambda^{1/p}$ where $p\ge 3$ is not a power of 2, are not possible solutions, because their so-called "critical exponent" is not dyadic. But the result seems not to apply to *piecewise* power series such as $\min(\lambda,1-\lambda)$, which are likewise algebraic functions.

Note 4: An exception is Chebyshev interpolants, but my implementation experience shows that

Chebyshev interpolants are far from being readily convertible to Bernstein form without using transcendental functions or paying attention to the difference between first vs. second kind, Chebyshev points vs. coefficients, and the interval [-1, 1] vs. [0, 1]. By contrast, other schemes (which are of greater interest to me) involve polynomials that are already in Bernstein form or that use only rational arithmetic to transform to Bernstein form (these include so-called "iterated Bernstein" polynomials and "one-bit" polynomials). Indeed, unlike with rational arithmetic (where arbitrary precision is trivial), transcendental functions require special measures to support arbitrary accuracy, such as constructive/recursive reals — floating-point numbers won't do for purposes of these open questions.

12 My Attempt

The Python script **lorentz.py** shows my attempt to implement the Holtz approximation scheme. It implements an algorithm to toss heads with probability equal to a C2 or C4 continuous piecewise polynomial factory function. However, it relies on an unproven conjecture (Conjecture 34) in the Holtz paper.

Based on this attempt, the C4 continuous case is efficient enough for my purposes, but the case of functions with lesser regularity is not so efficient (such as Lipschitz or C2).

Here is my current progress for the Lorentz operator for $\alpha=2$, so $r=2$ (which applies to twice-differentiable functions with Hölder continuous second derivative, even though the paper appears not to apply when α is an integer). Is the following work correct?

The Lorentz operator for $r=2$ finds the degree-n Bernstein polynomial for the target function f, elevates it to degree $n+r$, then shifts the coefficient at $k+1$ by $-f\prime\prime(k/n) A(n,k)$ (but the coefficients at 0 and $n+r$ are not shifted this way), where:

$$A(n,k) = (1/(4n)) \times 2 \times (n+2-k)/((n+1)\times (n+2)),$$

where k is an integer in $[0,n+r]$. Observing that $A(n,k)$ equals 0 at 0 and at $n+r$, and has a peak at $(n+r)/2$, the shift will be no greater (in absolute value) than $A(n,(n+r)/2)*F$, where F is the maximum absolute value of the second derivative of $f(\lambda)$. $A(n, (n+r)/2)$ is bounded above by $(3/16)/n$ for $n\ge 1$, and by $(3/22)/n$ for $n\ge 10$.

Now, find θ_α for $\alpha=2$, according to Lemma 25:

Let $0<\gamma<2^{\alpha/2}-1$ ($\gamma<1$ if $\alpha=2$).

Solve for K: $$(1-(1+\gamma)/2^{\alpha/2} - 4/(4*K)) = 0.$$ The solution for $\alpha=2$ is $K = 2/(1-\gamma)$.

Now find: $$\theta_a = ((4/4) K^{\alpha/2}/n^\alpha) / ((1-(1+\gamma)/2^\alpha)/n^\alpha)$$ The solution for $\alpha=2$ is $\theta_a = 8/((\gamma-3) (\gamma-1))$.

For $\gamma=1/100$ and $\alpha=2$, $\theta_a = 80000/29601 < 2.703$.

There's no need to check whether the output polynomials have Bernstein coefficients in $(0, 1)$, since out-of-bounds polynomials will be replaced with 0 or 1 as necessary — which is more practical and convenient.

Now all that remains is to find D given $\alpha=2$. I believe this will involve the following:

1. Find upper bounds for the constants C_{19}, C_{21}, and C_{24} (used in Lemmas 19, 21, and 24, respectively) given α and r (and for C_{19} at least, the best I can do is a visual inspection of the plot).
2. Calculate $t = C_{24} C_{19} (1+2 C_{21}) b^{\alpha/2} M$, where M is the Hölder constant of f's rth derivative (see "Step 4" in "The Iterative Construction").
3. Find the maximum value of $(t (x+(1-x))^r B_n \Delta_n{\alpha}) / ((x+(1-x))^r B_n\phi_n)$ (the best I can do is a visual inspection), and set D to that value.

Moreover, there remains to find the parameters for the Lorentz operator when r is 0, 1, 2, or 4. (When $r=0$ or $r=1$, the Lorentz operator is simply the Bernstein polynomial of degree n, elevated r degrees to degree $n+r$.)

13 References

- 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.
- Keane, M. S., and O'Brien, G. L., "A Bernoulli factory", *ACM Transactions on Modeling and Computer Simulation* 4(2), 1994.
- Holtz, O., Nazarov, F., Peres, Y., "New Coins from Old, Smoothly", Constructive Approximation 33 (2011).
- Nacu, Şerban, and Yuval Peres. "Fast simulation of new coins from old", The Annals of Applied Probability 15, no. 1A (2005): 93-115.
- 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.
- Peres, Y., "**Iterating von Neumann's procedure for extracting random bits**", Annals of Statistics 1992,20,1, p. 590-597.
- Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.
- Icard, Thomas F., "Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy." Journal of Mathematical Psychology 95 (2020): 102308.
- Dughmi, Shaddin, Jason Hartline, Robert D. Kleinberg, and Rad Niazadeh. "Bernoulli Factories and Black-box Reductions in Mechanism Design." Journal of the ACM (JACM) 68, no. 2 (2021): 1-30.
- Etessami, K. And Yannakakis, M., "Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations", Journal of the ACM 56(1), pp.1-66, 2009.
- Banderier, C. And Drmota, M., 2015. Formulae and asymptotics for coefficients of algebraic functions. Combinatorics, Probability and Computing, 24(1), pp.1-53.
- Esparza, J., Kučera, A. and Mayr, R., 2004, July. Model checking probabilistic pushdown automata. In Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004. (pp. 12-21). IEEE.
- Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560v2 [math.PR], 2010.
- von Neumann, J., "Various techniques used in connection with random digits", 1951.
- G.G. Lorentz, "The degree of approximation by polynomials with positive coefficients", 1966.
- Micchelli, C. (1973). The saturation class and iterates of the Bernstein polynomials. Journal of Approximation Theory, 8(1), 1-18.
- Guan, Zhong. "**Iterated Bernstein polynomial approximations**." arXiv preprint arXiv:0909.0684 (2009).
- Güntürk, C. Sinan, and Weilin Li. "**Approximation with one-bit polynomials in Bernstein form**", arXiv:2112.09183 (2021).
- Güntürk, C. Sinan, and Weilin Li. "**Approximation of functions with one-bit neural networks**", arXiv:2112.09181 (2021).
- Draganov, Borislav R. "On simultaneous approximation by iterated Boolean sums of Bernstein operators." Results in Mathematics 66, no. 1 (2014): 21-41.