Supplemental Notes for Bernoulli Factory Algorithms

Peter Occil

Supplemental Notes for Bernoulli Factory Algorithms

This version of the document is dated 2024-05-10.

Peter Occil

1 Contents

2 About This Document

This is an open-source document; for an updated version, see the source code[1] or its rendering on GitHub[2]. You can send comments on this document on the GitHub issues page[3]. See "Open Questions on the Bernoulli Factory Problem[4]" for a list of things about this document that I seek answers to.

[1] https://github.com/peteroupc/peteroupc.github.io/raw/master/bernsupp.md
[2] https://github.com/peteroupc/peteroupc.github.io/blob/master/bernsupp.md
[3] https://github.com/peteroupc/peteroupc.github.io/issues
[4] https://peteroupc.github.io/bernreq.html

My audience for this article is **computer programmers with mathematics knowledge, but little or no familiarity with calculus**. It should be read in conjunction with the article "**Bernoulli Factory Algorithms**[5]".

I encourage readers to implement any of the algorithms given in this page, and report their implementation experiences. In particular, **I seek comments on the following aspects**[6]:

- Are the algorithms in this article (in conjunction with "Bernoulli Factory Algorithms") easy to implement? Is each algorithm written so that someone could write code for that algorithm after reading the article?
- Does this article have errors that should be corrected?
- Are there ways to make this article more useful to the target audience?

Comments on other aspects of this document are welcome.

3 Definitions

This section describes certain math terms used on this page for programmers to understand.

The *closed unit interval* (written as [0, 1]) means the set consisting of 0, 1, and every real number in between.

The following terms can describe a function $f(x)$, specifically how "well-behaved" f is.

- A *continuous* function f has the property that there is a function $h(x, \epsilon)$ (where x is in f's domain and $\epsilon > 0$), such that $f(x)$ and $f(y)$ are less than ϵ apart whenever x and y are in f's domain and less than $h(x, \epsilon)$ apart.Roughly speaking, for each x in f's domain, $f(x)$ and $f(y)$ are "close" if x and y are "close" and belong in the domain.
- If f is continuous, its *derivative* is, roughly speaking, its "slope" function or "velocity" function or "instantaneous-rate-of-change" function. The derivative (or *first derivative*) is denoted f' or $f^{(1)}$. The *second derivative* ("slope-of-slope") of f, denoted f'' or $f^{(2)}$, is the derivative of f'; the *third derivative*, denoted $f^{(3)}$, is the derivative of f''; and so on. The *0-th derivative* of a function f is f itself and denoted $f^{(0)}$.
- A ***Hölder continuous***[7] function (with M being the *Hölder constant* and α being the *Hölder exponent*) is a continuous function f such that $f(x)$ and $f(y)$ are no more than $M^* \ \delta \ ^{\alpha}$ apart whenever x and y are in the function's domain and no more than δ apart.Here, α satisfies $0 < \alpha \leq 1$.Roughly speaking, the function's "steepness" is no greater than that of $M^* x^{\alpha}$.
- A *Lipschitz continuous* function with constant L (the *Lipschitz constant*) is Hölder continuous with Hölder exponent 1 and Hölder constant L.Roughly speaking, the function's "steepness" is no greater than that of $L^* x$.If the function has a derivative on its domain, L can be the maximum of the absolute value of that derivative.
- A *convex* function f has the property that $f((x+y)/2) \leq (f(x)+f(y))/2$ whenever x, y, and $(x+y)/2$ are in the function's domain.Roughly speaking, if the function's "slope" never goes down, then it's convex.
- A *concave* function f has the property that $f((x+y)/2) \geq (f(x)+f(y))/2$ whenever x, y, and $(x+y)/2$ are in the function's domain.Roughly speaking, if the function's "slope" never goes up, then it's concave.

  Note: The "good behavior" of a function can be important when designing Bernoulli factory algorithms. This page mostly cares how f behaves when its domain is the closed unit interval, that is, when $0 \leq x \leq 1$.

[5] https://peteroupc.github.io/bernoulli.html
[6] https://github.com/peteroupc/peteroupc.github.io/issues/18
[7] https://en.wikipedia.org/wiki/Hölder_condition

4 General Factory Functions

As a reminder, the *Bernoulli factory problem* is: Suppose there is a coin that shows heads with an unknown probability, λ , and 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 $f(\lambda)$.

The algorithm for **general factory functions**[8], described in my main article on Bernoulli factory algorithms, works by building randomized upper and lower bounds for a function $f(\lambda)$, based on flips of the input coin. Roughly speaking, the algorithm works as follows:

1. Generate a random variate, U, uniformly distributed, greater than 0 and less than 1.
2. Flip the input coin, then build an upper and lower bound for $f(\lambda)$, based on the outcomes of the flips so far.
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.

These randomized upper and lower bounds come from two sequences of polynomials as follows:

1. One sequence approaches the function $f(\lambda)$ from above, the other from below, and both sequences must converge to f.
2. For each sequence, the first polynomial has degree 1 (so is a linear function), and each other polynomial's degree is 1 higher than the previous.
3. The *consistency requirement* must be met, namely—
   - the difference between the degree-$(n-1)$ upper polynomial and the degree-n upper polynomial, and
   - the difference between the degree-n lower polynomial and the degree-$(n-1)$ lower polynomial,

   must have nonnegative Bernstein coefficients, once each of these differences is rewritten as a polynomial of degree exactly n. (For more on Bernstein coefficients and the Bernstein form of polynomials, see "**Certain Polynomials**[9]" in the main article.)

The consistency requirement ensures that the upper polynomials "decrease" and the lower polynomials "increase". Unfortunately, the reverse is not true in general; even if the upper polynomials "decrease" and the lower polynomials "increase" to f, this does not ensure the consistency requirement by itself.

> **Example (Nacu & Peres [2005][10]):** The polynomial $x^2+(1-x)^2$ is of degree 2 with Bernstein coefficients [1, 0, 1]. The polynomial $x(1-x)$ is of degree 2 with Bernstein coefficients [0, 1/2, 0]. Although $(x^2+(1-x)^2)$ minus $(x(1-x))$ is non-negative, this difference's Bernstein coefficients of degree 2 are not always non-negative, namely, the Bernstein coefficients are [1, -1/2, 1].

In this document, **fbelow**(n, k) and **fabove**(n, k) mean the k^{th} Bernstein coefficient for the lower or upper degree-n polynomial, respectively, where $0 \le k \le n$ is an integer.

The section "Building the Lower and Upper Polynomials" are ways to build sequences of polynomials that appropriately converge to a factory function if that function meets certain conditions.

To determine if the methods are right for $f(\lambda)$, a deep mathematical analysis of f is required; it would be helpful to plot that function using a computer algebra system to see if it is described in the next section.

4.1 Building the Lower and Upper Polynomials

The rest of this section assumes $f(\lambda)$ is not a constant. For examples of functions, see "**Examples of Well-Behaved Functions**", in the appendix.

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

[9] https://peteroupc.github.io/bernoulli.html#Certain_Polynomials

[10] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

Concave functions. If f is concave, then **fbelow**(n, k) can equal $f(k/n)$, thanks to Jensen's inequality.

Convex functions. If f is convex, then **fabove**(n, k) can equal $f(k/n)$, thanks to Jensen's inequality.

Hölder and Lipschitz continuous functions. I have found a way to extend the results of Nacu and Peres (2005)[11] to certain functions with a slope that tends to a vertical slope. The following scheme, proved in the appendix, implements **fabove** and **fbelow** if $f(\lambda)$—

- is ***Hölder continuous***[12] on the closed unit interval, with Hölder constant m or less and Hölder exponent α (see "**Definitions**" as well as "**Examples of Well-Behaved Functions**", in the appendix), and
- on the closed unit interval—
  - has a minimum of greater than 0 and a maximum of less than 1, or
  - is convex and has a minimum of greater than 0, or
  - is concave and has a maximum of less than 1.

For every integer n that's a power of 2:

- $D(n) = m*(2/7)^{\alpha/2}/((2^{\alpha/2}-1)*n^{\alpha/2})$.
- **fbelow**(n, k) $= f(k/n)$ if f is concave; otherwise, min(**fbelow**(4,0), **fbelow**(4,1), ..., **fbelow**(4,4)) if $n < 4$; otherwise, $f(k/n) - D(n)$.
- **fabove**(n, k) $= f(k/n)$ if f is convex; otherwise, max(**fabove**(4,0), **fabove**(4,1), ..., **fabove**(4,4)) if $n < 4$; otherwise, $f(k/n) + D(n)$.

Notes:

1. If α is 1, $D(n)$ can be $m*322613/(250000*\mathrm{sqrt}(n))$, which is an upper bound. If α is 1/2, $D(n)$ can be $m*154563/(40000*n^{1/4})$, which is an upper bound.
2. The function $f(x) = \min(\lambda t, 1-\epsilon)$, where $\epsilon \geq 0$ and $t \geq 1$, is Lipschitz continuous with Lipschitz constant t. Because f is linear between 0 and $1/t$, ways to build polynomials that converge to this kind of function were discussed by Thomas and Blanchet (2012)[13] [14] and Nacu & Peres (2005)[15] [16].

Functions with a Lipschitz continuous derivative. The following method, proved in the appendix, implements **fabove** and **fbelow** if $f(\lambda)$—

- has a Lipschitz continuous derivative (see "**Definitions**" as well as "**Examples of Well-Behaved Functions**", in the appendix), and
- in the closed unit interval—

[11] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[12] https://en.wikipedia.org/wiki/Hölder_condition

[13] Thomas, A.C., Blanchet, J., "**A Practical Implementation of the Bernoulli Factory**", arXiv:1106.2508v3 [stat.AP], 2012. https://arxiv.org/abs/1106.2508v3

[14] Thomas and Blanchet (2012) dealt with building polynomials that approach piecewise linear functions "fast". Their strategy for $f(\lambda) = \min(mult \times \lambda, 1-2\varepsilon)$ is to first compute a low-degree polynomial P satisfying $P(0) = 0$ and otherwise greater than f, and then compute further polynomials of increasing degree that each come between f and the previous polynomial and satisfy the consistency requirement. These polynomials approach f rapidly when λ is near 0, and extremely slowly when λ is near 1. In their strategy, **fbelow(n, k)** is $\min((k/n)*mult, 1-\varepsilon)$, and **fabove($n$, k)** is $\min((k/n)*y/x, y)$, where: $x = -((y-(1-\varepsilon))/\varepsilon)^5/mult + y/mult$. (This formula doesn't appear in their paper, but in the **supplemental source code** uploaded by A. C. Thomas at my request.) y satisfies $0<y<1$ and is chosen so that the degree-n polynomial is between f and the previous polynomial and meets the consistency requirement. The supplemental source code seems to choose y in an *ad hoc* manner. n is the polynomial's degree. https://github.com/acthomasca/rberfac/blob/main/rberfac-public-2.R

[15] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[16] In Nacu and Peres (2005), the following polynomial sequences were suggested to simulate $\min(2\lambda, 1-2\varepsilon)$ (using the algorithms from the section "General Factory Functions" in "**Bernoulli Factory Algorithms**"), provided $\varepsilon < 1/8$, where n is a power of 2. However, with these sequences, each simulation will require an extraordinary number of input coin flips. **fbelow(n, k)** $= \min(2(k/n), 1-2\varepsilon)$. **fabove($n$, k)** $= \min(2(k/n), 1-2\varepsilon) + \frac{2\times\max(0,k/n+3\varepsilon-1/2)}{\varepsilon(2-\sqrt{2})}\sqrt{2/n} + \frac{72\times\max(0,k/n-1/9)}{1-\exp(-2\times\varepsilon^2)}\exp(-2n\times\varepsilon^2)$. https://peteroupc.github.io/bernoulli.html

– has a minimum of greater than 0 and a maximum of less than 1, or
– is convex and has a minimum of greater than 0, or
– is concave and has a maximum of less than 1.

Let m be the Lipschitz constant of f's derivative, or a greater number than that constant. Then for every integer n that's a power of 2:

- For every n such that **fbelow**(n,k) is 0 or greater for every k, **fbelow**(n, k) = $f(k/n)$ if f is concave; otherwise, min(**fbelow**(4,0), **fbelow**(4,1), …, **fbelow**(4,4)) if $n < 4$; otherwise, $f(k/n) - m/(7*n)$. For every other n, **fbelow**(n,k) = 0.
- For every n such that **fabove**(n,k) is 1 or less for every k, **fabove**(n, k) = $f(k/n)$ if f is convex; otherwise, max(**fabove**(4,0), **fabove**(4,1), …, **fabove**(4,4)) if $n < 4$; otherwise, $f(k/n) + m/(7*n)$. For every other n, **fabove**(n,k) = 1.

Certain functions that equal 0 at 0. This approach involves transforming the function f so that it no longer equals 0 at the point 0. This can be done by dividing f by a function (`High`(λ)) that "dominates" f everywhere on the closed unit interval. Unlike for the original function, there might be a polynomial-building scheme described earlier in this section for the transformed function.

More specifically, `High`(λ) must meet the following requirements:

- `High`(λ) is continuous on the closed unit interval.
- `High`(0) = 0. (This is required to ensure correctness in case λ is 0.)
- $1 \geq$ `High`(1) $\geq f(1) \geq 0$.
- $1 >$ `High`(λ) $> f(\lambda) > 0$ whenever $0 < \lambda < 1$.
- If $f(1) = 0$, then `High`(1) = 0. (This is required to ensure correctness in case λ is 1.)

Also, `High` is a Bernoulli factory function that should admit a simple Bernoulli factory algorithm. For example, `High` can be the following degree-n polynomial: $1 - (1 - \lambda)^n$, where $n \geq 1$ is an integer.[17]

The algorithm is now described.

Let $g(\lambda) = \lim \nu \to \lambda\ f(\nu)/$`High`($\nu$) (roughly speaking, the value that $f(\nu)/$`High`(ν) approaches as ν approaches λ .) If—

- $f(0) = 0$ and $f(1) < 1$, and
- $g(\lambda)$ is continuous on the closed unit interval and belongs in one of the classes of functions given earlier,

then f can be simulated using the following algorithm:

1. Run a Bernoulli factory algorithm for `High`. If the call returns 0, return 0. (For example, if `High`(λ) = λ , then this step amounts to the following: "Flip the input coin. If it returns 0, return 0.")
2. Run a Bernoulli factory algorithm for $g(.)$ and return the result of that algorithm. This can be one of the **general factory function algorithms**[18] if there is a way to calculate polynomials that converge to $g(.)$ in a manner needed for that algorithm (for example, if g is described earlier in this section).

   Notes:

   1. It may happen that $g(0) = 0$. In this case, step 2 of this algorithm can involve running this algorithm again, but with new g and `High` functions that are found based on the current g function. (This will eventually result in $g(0)$ being nonzero if f is a nonconstant Bernoulli factory function.) See the second example below.
   2. `High`(λ) can also equal 1 instead of be described in this section. That leads to the original Bernoulli factory algorithm for $f(\lambda)$.

   Examples:

[17] In this case, an algorithm to simulate `High`(λ) is: Flip the input coin n times or until a flip returns 1, whichever comes first, then output the last coin flip result.

[18] https://peteroupc.github.io/bernoulli.html#General_Factory_Functions

1. If $f(\lambda) = (\sinh(\lambda)+\cosh(\lambda) - 1)/4$, then f is less than or equal to $\texttt{High}(\lambda) = \lambda$, so $g(\lambda)$ is $1/4$ if $\lambda = 0$, and $(\exp(\lambda) - 1)/(4^* \lambda)$ otherwise. The following code in Python that uses the SymPy computer algebra library computes this example: `fx = (sinh(x)+cosh(x)-1)/4; h = x; pprint(Piecewise((limit(fx/h,x,0), Eq(x,0)), ((fx/h).simplify(), True)))`.
2. If $f(\lambda) = \cosh(\lambda) - 1$, then f is less than or equal to $\texttt{High}(\lambda) = \lambda$, so $g(\lambda)$ is 0 if $\lambda = 0$, and $(\cosh(\lambda) - 1)/ \lambda$ otherwise. Now, since $g(0) = 0$, find new functions g and High based on the current g. The current g is less than or equal to $\texttt{High}(\lambda) = \lambda \ ^*3^*(2 - \lambda)/5$ (a degree-2 polynomial that has Bernstein coefficients [0, 6/10, 6/10]), so $G(\lambda) = 5/12$ if $\lambda = 0$, and $-(5^*\cosh(\lambda) - 5)/(3^* \lambda \ ^2*(\lambda - 2))$ otherwise. G is bounded away from 0 and 1, resulting in the following algorithm:
    1. (Simulate High.) Flip the input coin. If it returns 0, return 0.
    2. (Simulate High.) Flip the input coin twice. If both flips return 0, return 0. Otherwise, with probability 4/10 (that is, 1 minus 6/10), return 0.
    3. Run a Bernoulli factory algorithm for G (which might involve building polynomials that converge to G, noticing that G's derivative is Lipschitz continuous) and return the result of that algorithm.

Certain functions that equal 0 at 0 and 1 at 1. Let f, g, and High be functions as defined earlier, except that $f(0) = 0$ and $f(1) = 1$. Define the following additional functions:

- $\texttt{Low}(\lambda)$ is a function that meets the following requirements:
  - $\texttt{Low}(\lambda)$ is continuous on the closed unit interval.
  - $\texttt{Low}(0) = 0$ and $\texttt{Low}(1) = 1$.
  - $1 > f(\lambda) > \texttt{Low}(\lambda) > 0$ whenever $0 < \lambda < 1$.
- $q(\lambda) = \lim \nu \to \lambda \ \texttt{Low}(\nu)/\texttt{High}(\nu)$.
- $r(\lambda) = \lim \nu \to \lambda \ (1 - g(\nu))/(1 - q(\nu))$.

Roughly speaking, Low is a function that bounds f from below, just as High bounds f from above. Low is a Bernoulli factory function that should admit a simple Bernoulli factory algorithm; one example is the polynomial $\lambda \ ^n$ where $n \geq 1$ is an integer. If both Low and High are polynomials of the same degree, q will be a ratio of polynomials with a relatively simple Bernoulli factory algorithm (see "**Certain Rational Functions**[19]").

Now, if $r(\lambda)$ is continuous on the closed unit interval, then f can be simulated using the following algorithm:

1. Run a Bernoulli factory algorithm for High. If the call returns 0, return 0. (For example, if $\texttt{High}(\lambda) = \lambda$, then this step amounts to the following: "Flip the input coin. If it returns 0, return 0.")
2. Run a Bernoulli factory algorithm for $q(.)$. If the call returns 1, return 1.
3. Run a Bernoulli factory algorithm for $r(.)$, and return 1 minus the result of that call. The Bernoulli factory algorithm can be one of the **general factory function algorithms**[20] if there is a way to calculate polynomials that converge to $r(.)$ in a manner needed for that algorithm (for example, if r is described earlier in this section).

    Notes:

    1. Quick proof: Rewrite $f = \text{High} \cdot (q \cdot 1 + (1 - q) \cdot (1 - r)) + (1 - \text{High}) \cdot 0$.
    2. $\texttt{High}(\lambda)$ is allowed to equal 1 if the $r(.)$ in step 3 is allowed to equal 0 at 0.

    Example: If $f(\lambda) = (1 - \exp(\lambda))/(1 - \exp(1))$, then f is less than or equal to $\texttt{High}(\lambda) = \lambda$, and greater than or equal to $\texttt{Low}(\lambda) = \lambda \ ^2$. As a result, $q(\lambda) = \lambda$, and $r(\lambda) = (2 - \exp(1))/(1 - \exp(1))$ if $\lambda = 0$; $1/(\exp(1) - 1)$ if $\lambda = 1$; and $(- \lambda \ ^*(1 - \exp(1)) - \exp(\lambda) + 1)/(\lambda \ ^*(1 - \exp(1))^*(\lambda - 1))$ otherwise. This can be computed using the following code

[19] https://peteroupc.github.io/bernoulli.html#Certain_Rational_Functions
[20] https://peteroupc.github.io/bernoulli.html#General_Factory_Functions

in Python that uses the SymPy computer algebra library: `fx=(1-exp(x))/(1-exp(1)); high=x; low=x**2; q=(low/high); r=(1-fx/high)/(1-q); r=Piecewise((limit(r, x, 0), Eq(x,0)), (limit(r,x,1),Eq(x,1)), (r,True)).simplify(); pprint(r)`.

Other functions that equal 0 or 1 at the endpoint 0, 1, or both. The table below accounts for these Bernoulli factory functions:

If $f(0) =$	And $f(1) =$	Method
> 0 and < 1	1	Use the algorithm for **certain functions that equal 0 at 0**, but with $f(\lambda) = 1 - f(1 - \lambda)$.*Inverted coin*: Instead of the usual input coin, use a coin that does the following: "Flip the input coin and return 1 minus the result."*Inverted result:* If the overall algorithm would return 0, it returns 1 instead, and vice versa.
> 0 and < 1	0	Algorithm for **certain functions that equal 0 at 0**, but with $f(\lambda) = f(1 - \lambda)$. (For example, $\cosh(\lambda) - 1$ becomes $\cosh(1 - \lambda) - 1$.)Inverted coin.
1	0	Algorithm for **certain functions that equal 0 at 0 and 1 at 1**, but with $f(\lambda) = 1 - f(\lambda)$.Inverted result.
1	> 0 and ≤ 1	Algorithm for **certain functions that equal 0 at 0**, but with $f(\lambda) = 1 - f(\lambda)$.Inverted result.

4.2 Another General Algorithm

The algorithm I've developed in this section simulates $f(\lambda)$ when f belongs in a large class of functions, as long as the following is known:

- f is continuous and has a minimum of greater than 0 and a maximum of less than 1.
- There is a family of polynomials ($L_1(f)$, $L_2(f)$, $L_4(f)$, $L_8(f)$, …) that come close to f with a known error bound, where the number after L is the degree of the polynomial.
- There is a way to find the *Bernstein coefficients* of each polynomial $L_n(f)$ in the family of polynomials.

For examples of suitable polynomials, see **"Approximations in Bernstein Form"**[21].

In effect, the algorithm writes f as an infinite sum of polynomials, whose maximums must sum to 1 or less (called T in the algorithm below), then simulates an appropriate **convex combination**[22] (weighted sum) of these polynomials. To build the convex combination, each polynomial in the infinite sum is divided by an upper bound on its maximum (which is why error bounds on $L_n(f)$ are crucial here).[23] To simulate f, the algorithm—

[21] https://peteroupc.github.io/bernapprox.html

[22] https://peteroupc.github.io/bernoulli.html#Convex_Combinations

[23] With this infinite sum and these upper bounds, Lemma 4 of Holtz et al. (2011) says that a Bernoulli factory algorithm for $f(\lambda)$ is possible.

- selects a polynomial in the convex combination with probability proportional to its upper bound, or a "leftover" zero polynomial with probability T, then
- simulates the chosen polynomial (which is easy to do; see Goyal and Sigman (2012)[24]).

- In the algorithm, denote:
  - $\epsilon(f, n)$ as an upper bound on the absolute value of the difference between f and the degree-n polynomial $L_n(f)$. $\epsilon(f, n)$ must increase nowhere as n increases, and must converge to 0.
    - For best results, this should be written as $\epsilon(f, n) = C/n^r$, where C is a constant and $r > 0$ is a multiple of $1/2$, since then it's easy to find the value of ErrShift(f, n), below. In this case, the algorithm should be limited to functions with a continuous $(2r)$-th derivative or a Lipschitz continuous $(2r-1)$-th derivative (see "**Achievable Simulation Rates**", later). (For example, if the error bound is C/n^2, the function f should have a continuous fourth derivative or a Lipschitz continuous third derivative.)
    - For examples of error bounds, see **"Approximations in Bernstein Form"**[25].
  - ErrShift(f, m) as $1.01 \cdot \sum_{i \geq m} \epsilon(f, 2^i)$. The factor 1.01 is needed to ensure each difference polynomial is strictly between 0 and 1.
    - **Example:** If $\epsilon(f, n) = C/n^r$, then ErrShift$(f, m) = 1.01 \cdot C \cdot 2^r/(((2^r) - 1) \cdot 2^{rm})$.
  - DiffWidth(f, m) as $1.01 \cdot 2(\epsilon(f, 2^m) + \epsilon(f, 2^{m+1}))$. This is an upper bound on the maximum difference between the shifted degree-2^m and the shifted degree-(2^{m+1}) polynomial.
- The technique breaks f into a **starting polynomial** and a family of **difference polynomials**.To find the **starting polynomial**:
  1. Set m to 0.
  2. Find the Bernstein coefficients of L_{2^m}, then subtract ErrShift(f, m) from them. If those coefficients now all lie in the closed unit interval, go to the next step. Otherwise, add 1 to m and repeat this step.
  3. Calculate **StartWidth** as ceil$(c \cdot 65536)/65536$, where c is the maximum Bernstein coefficient from step 2, then divide each Bernstein coefficient by **StartWidth**. (65536 is arbitrary and ensures **StartWidth** is a rational number that is close to, but no lower than, the maximum Bernstein coefficient, for convenience.)
  4. The **starting polynomial** now has Bernstein coefficients found in step 3. Set **StartOrder** to m.
- To find the **difference polynomial** of order m:
  1. Find the Bernstein coefficients of L_{2^m}, then subtract ErrShift(f, m) from them. Rewrite them to Bernstein coefficients of degree 2^{m+1}. Call the coefficients **LowerCoeffs**.
  2. Find the Bernstein coefficients of $L_{2^{m+1}}$, then subtract ErrShift$(f, m+1)$ from them. Call the resulting values **UpperCoeffs**.
  3. Subtract **UpperCoeffs** from **LowerCoeffs**, and call the result **DiffCoeffs**. Divide each coefficient in **DiffCoeffs** by DiffWidth(f, m). The result is the Bernstein coefficients of a positive polynomial of degree 2^{m+1} bounded by 0 and 1, but these coefficients are not necessarily bounded by 0 and 1. Thus, if any coefficient in **DiffCoeffs** is less than 0 or greater than 1, add 1 to m and rewrite **DiffCoeffs** to Bernstein coefficients of degree 2^{m+1} until no coefficient is less than 0 or greater than 1 anymore.
  4. The **difference polynomial** now has Bernstein coefficients given by **DiffCoeffs**.
- The probabilities for X are as follows:
  - First, find the **starting polynomial**, then calculate T as **StartWidth** + $\sum_{i \geq 0}$ DiffWidth$(f,$ **StartOrder**$+i)$. If T is greater than 1, this algorithm can't be used.
    - **Example:** If $\epsilon(f, n) = C/n^r$, then $T =$ **StartWidth** $+ 1.01 \cdot (2 \cdot 2^{-r \cdot \text{StartOrder}} C(1+2^{-r}))/(1-2^{-r})$.
  - X is 0 with probability $1 - T$.

[24]Goyal, V. and Sigman, K., 2012. On simulating a class of Bernstein polynomials. ACM Transactions on Modeling and Computer Simulation (TOMACS), 22(2), pp.1-5.

[25]https://peteroupc.github.io/bernapprox.html

- X is 1 with probability equal to **StartWidth**.
- For each $m \geq 2$, X is m with probability equal to DiffWidth(f,**StartOrder** + $m - 2$).

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

1. Generate X at random with the probabilities given above.
2. If X is 0, return 0. Otherwise, if X is 1, find the **starting polynomial** and its Bernstein coefficients. Otherwise (if X is 2 or greater), find the **difference polynomial** of order m and its Bernstein coefficients, where $m = (X - 2)$ + **StartOrder**.
3. Flip the input coin (with probability of heads λ), $n - 1$ times, where n is the number of Bernstein coefficients in the polynomial found in step 2 (its degree plus one), and let j be the number of heads.
4. Return 1 with probability equal to the polynomial's jth Bernstein coefficient (j starts at 0), or 0 otherwise (see also Goyal and Sigman 2012 for an algorithm to simulate polynomials).

If T turns out to be greater than 1 in this algorithm, but still finite, one way to simulate f is to create a coin that simulates $f(\lambda)/T$ instead, and use that coin as the input to a ***linear Bernoulli factory***[26] that simulates $T \cdot (f(\lambda)/T)$. (This is possible especially because $f(\lambda)$ is assumed to have a maximum of less than 1.)

> **Example**: The following parameters allow this algorithm to work if f is concave, has a maximum of less than 1, and has a Lipschitz-continuous derivative with Lipschitz constant M or less. In this case, it is allowed that $f(0) = 0$, $f(1) = 0$, or both.
>
> - The family of polynomials $L_n(f)$ is simply the family of Bernstein polynomials of f. The Bernstein coefficients of $L_n(f)$ are $f(0/n), f(1/n), ..., f(n/n)$.
> - The error bound is $\epsilon(f, n) = M/(8n)$ (Lorentz 1963)[27].
> - The **starting polynomial** is found as follows. Let $c = \max(f(0), f(1))$. Then the starting polynomial has two Bernstein coefficients both equal to c; **StartWidth** is equal to ceil($c \cdot 65536$)/65536, and **StartOrder** is equal to 0.
> - ErrShift(f, m) = 0. The reason for 0 is that f is concave, so its Bernstein polynomials naturally "increase" with increasing degree (Temple 1954)[28], (Moldovan 1962)[29].
> - DiffWidth(f, m) = $1.01 \cdot 3M/(8 \cdot 2^m)$. For the same reason as the previous point, and because the Bernstein polynomials are always "below" f, DiffWidth(f, m) can also equal $1.01 \cdot \epsilon(f, 2^m) = 1.01 \cdot M/(8 \cdot 2^m)$. This is what is used to calculate T, below.
> - T is calculated as **StartWidth** + $1.01 \cdot M/4$.

4.3 Request for Additional Methods

Readers are requested to let me know of additional solutions to the following problem:

*Let $f(\lambda)$ be continuous and satisfy $0 < f(\lambda) < 1$ whenever $0 \leq \lambda \leq 1$. Given that $f(\lambda)$ belongs to a large class of functions (for example, it has a continuous, Lipschitz continuous, concave, or nowhere decreasing k-th derivative for some integer k, or any combination of these), compute the Bernstein coefficients for two sequences of polynomials as follows: one of them approaches $f(\lambda)$ from above, the other from below, and the consistency requirement must be met (see "**General Factory Functions**").*

The polynomials need to be computed only for degrees 2, 4, 8, 16, and higher powers of 2.

The rate of convergence must be no slower than $1/n^{r/2}$ if the given class has only functions with continuous r-th derivative.

Methods that use only integer arithmetic and addition and multiplication of rational numbers are preferred (thus, methods that involve cosines, sines, π, exp*, and* ln *are not preferred).*

[26] https://peteroupc.github.io/bernoulli.html#Linear_Bernoulli_Factories

[27] G.G. Lorentz, "Inequalities and saturation classes for Bernstein polynomials", 1963.

[28] Temple, W.B., "Steltjes integral representation of convex functions", 1954.

[29] Moldovan, E., "Observations sur la suite des polynômes de S. N. Bernstein d'une fonction continue", 1962.

See also the **open questions**[30].

5 Approximate Bernoulli Factories

An **approximate Bernoulli factory** for a function $f(\lambda)$ is a Bernoulli factory algorithm that simulates another function, $g(\lambda)$, that approximates f in some sense.

Usually g is a polynomial, but can also be a rational function (ratio of polynomials) or another function with an easy-to-implement Bernoulli factory algorithm.

Meanwhile, $f(\lambda)$ can be any function that maps the closed unit interval to itself, even if it isn't continuous or a factory function (examples include the "step function" 0 if $\lambda < 1/2$ and 1 otherwise, or the function 2*min(λ , $1 - \lambda$)). If the function is continuous, it can be approximated arbitrarily well by an approximate Bernoulli factory (as a result of the so-called "Weierstrass approximation theorem"), but generally not if the function is discontinuous.

To build an approximate Bernoulli factory with a polynomial:

1. First, find a polynomial in Bernstein form of degree n that is close enough to the desired function $f(\lambda)$.

   The simplest choice for this polynomial, known simply as a *Bernstein polynomial*, has n+1 Bernstein coefficients and its j^{th} coefficient (starting at 0) is found as $f(j/n)$. For this choice, if f is continuous, the polynomial can be brought arbitrarily close to f by choosing n high enough.

   Other choices for this polynomial are given in the page **"Approximations in Bernstein Form"**[31].

   Whatever polynomial is used, the polynomial's Bernstein coefficients must all lie on the closed unit interval.

   The polynomial can be in *homogeneous form* (also known as *scaled Bernstein* form (Farouki and Rajan 1988)[32]) instead of in Bernstein form, with *scaled Bernstein coefficients* $s_0, ..., s_n$, as long as $0 \le s_i \le \binom{n}{i}$ where $0 \le i \le n$.

2. The rest of the process is to toss heads with probability equal to that polynomial, given its Bernstein coefficients. To do this, first flip the input coin n times, and let j be the number of times the coin returned 1 this way.

3. Then, with probability equal to—

   - the polynomial's Bernstein coefficient at position j (which will be $f(j/n)$ in the case of the Bernstein polynomial $B_n(f)$), or
   - the polynomial's scaled Bernstein coefficient at position j, divided by choose(n, j),

   return 1. Otherwise, return 0. Here, $0 \le j \le n$.

   If the probability can be an irrational number, see "**Algorithms for General Irrational Constants**[33]" for ways to exactly sample a probability equal to that irrational number.

   Notes:

   1. More sophisticated ways to implement steps 2 and 3 are found in the section "**Certain Polynomials**[34]" in the main article "Bernoulli Factory Algorithms".

[30] https://peteroupc.github.io/bernreq.html#Polynomials_that_approach_a_factory_function_fast

[31] https://peteroupc.github.io/bernapprox.html

[32] Farouki, Rida T., and V. T. Rajan. "**Algorithms for polynomials in Bernstein form**". Computer Aided Geometric Design 5, no. 1 (1988): 1-26. https://www.sciencedirect.com/science/article/pii/0167839688900167

[33] https://peteroupc.github.io/bernoulli.html#Algorithms_for_General_Irrational_Constants

[34] https://peteroupc.github.io/bernoulli.html

2. There are other kinds of functions, besides polynomials and rational functions, that serve to approximate continuous functions. But many of them work poorly as approximate Bernoulli factory functions because their lack of "smoothness" means there is no simple Bernoulli factory for them. For example, a *spline*, which is a continuous function made up of a finite number of polynomial pieces, is generally not "smooth" at the points where the spline's pieces meet.
3. Bias and variance are the two sources of error in a randomized estimation algorithm. Let $g(\lambda)$ be an approximation of $f(\lambda)$. The original Bernoulli factory for f, if it exists, has bias 0 and variance $f(\lambda)$*$(1 - f(\lambda))$, but the approximate Bernoulli factory has bias $g(\lambda) - f(\lambda)$ and variance $g(\lambda)$*$(1 - g(\lambda))$. ("Variance reduction" methods are outside the scope of this document.) An estimation algorithm's *mean squared error* equals variance plus square of bias.
4. There are two known approximations to the linear function $f(\lambda) = 2\lambda$ using a polynomial in Bernstein form of degree n that maps the open interval (0, 1) to itself. In each case, if $g(\lambda)$ is that polynomial and if $0 \le \lambda \le 1/2$, then the error in approximating $f(\lambda)$ is no greater than $1 - g(1/2)$.
    - In Henderson and Glynn (2003, Remark 4)[35], the polynomial's j^{th} Bernstein coefficient (starting at 0) is min((j/n)*2, $1 - 1/n$). The polynomial g can be computed with the SymPy computer algebra library as follows: `from sympy.stats import *; g=2*E( Min(sum(Bernoulli(("B%d" % (i)),z) for i in range(n))/n,(S(1)-S(1)/n)/2))`.
    - In Nacu and Peres (2005, section 6)[36], the polynomial's j^{th} Bernstein coefficient (starting at 0) is min((j/i)*2, 1). It corresponds to the following algorithm: Flip the input coin n times or until the ratio of "heads" to "flips" becomes at least 1/2, whichever comes first, then if n flips were made without the ratio becoming at least 1/2, return 0; otherwise, return 1.

6 Achievable Simulation Rates

In general, the number of input coin flips needed by any Bernoulli factory algorithm for a factory function $f(\lambda)$ depends on how "smooth" the function f is.

The following table summarizes the rate of simulation (in terms of the number of input coin flips needed) that can be achieved *in theory* depending on $f(\lambda)$, assuming the input coin's probability of heads is unknown. In the table below:

- λ , the unknown probability of heads, is ε or greater and $(1 - \varepsilon)$ or less for some $\varepsilon > 0$.
- The simulation makes use of unbiased random bits in addition to input coin flips.
- $\Delta\ (n, r, \lambda) = \max(\mathrm{sqrt}(\lambda\ *(1 - \lambda)/n), 1/n)^r$.

Property of simulation	Property of f
Requires no more than n input coin flips.	If and only if f can be written as a polynomial of degree n with Bernstein coefficients in the closed unit interval (Goyal and Sigman 2012)[37].

[35] Henderson, S.G., Glynn, P.W., "Nonexistence of a class of variate generation schemes", *Operations Research Letters* 31 (2003).

[36] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[37] Goyal, V. and Sigman, K., 2012. On simulating a class of Bernstein polynomials. ACM Transactions on Modeling and Computer Simulation (TOMACS), 22(2), pp.1-5.

Property of simulation	Property of f
Requires a finite number of flips on average. Also known as "realizable" by Flajolet et al. (2010)[38].	Only if f is Lipschitz continuous (Nacu and Peres 2005)[39].Whenever f admits a fast simulation (Mendo 2019)[40].
Number of flips required, raised to power of r, is bounded by a finite number on average and has a tail that drops off uniformly over f's domain.	Only if f has continuous r-th derivative (Nacu and Peres 2005)[41].
Requires more than n flips with at most a probability proportional to $\Delta\ (n,\ r+1,\ \lambda)$, for integer $r \geq 0$ and every λ , and for large enough n. (The greater r is, the faster the simulation.)	Only if f has an r-th derivative that is continuous and in the Zygmund class (see note below) (Holtz et al. 2011, Theorem 13)[42].
Requires more than n flips with at most a probability proportional to $\Delta\ (n,\ \alpha\ ,\ \lambda)$, for non-integer $\alpha > 0$ and every λ , and for large enough n. (The greater α is, the faster the simulation.)	If and only if f has an r-th derivative that is Hölder continuous with Hölder exponent $(\alpha - r)$ or greater, where $r = \text{floor}(\alpha)$ (Holtz et al. 2011, Theorem 8)[43]. Assumes f is bounded away from 0 and 1.
"Fast simulation" (requires more than n flips with a probability that decays exponentially as n gets large). Also known as "strongly realizable" by Flajolet et al. (2010)[44].	If and only if f is analytic at every point in its domain (see note below) (Nacu and Peres 2005)[45].
Average number of flips greater than or equal to $(f'(\lambda))^{2} * \lambda * (1-\lambda)/(f(\lambda)*(1-f(\lambda)))$, where f' is the first derivative of f.	Whenever f admits a fast simulation (Mendo 2019)[46].

Note: A function $f(\lambda)$ is:

- *Analytic* at a point z if there is a positive number r such that f is writable as—

$$f(\lambda) = f(z) + f^{(1)}(z)(\lambda - z)^1/1! + f^{(2)}(z)(\lambda - z)^2/2! + ...,$$

[38]Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010. https://arxiv.org/abs/0906.5560

[39]Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[40]Mendo, Luis. "An asymptotically optimal Bernoulli factory for certain functions that can be expressed as power series." Stochastic Processes and their Applications 129, no. 11 (2019): 4366-4384.

[41]Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[42]Holtz, O., Nazarov, F., Peres, Y., "**New Coins from Old, Smoothly**", *Constructive Approximation* 33 (2011). https://link.springer.com/content/pdf/10.1007/s00365-010-9108-5.pdf

[43]Holtz, O., Nazarov, F., Peres, Y., "**New Coins from Old, Smoothly**", *Constructive Approximation* 33 (2011). https://link.springer.com/content/pdf/10.1007/s00365-010-9108-5.pdf

[44]Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010. https://arxiv.org/abs/0906.5560

[45]Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[46]Mendo, Luis. "An asymptotically optimal Bernoulli factory for certain functions that can be expressed as power series." Stochastic Processes and their Applications 129, no. 11 (2019): 4366-4384.

whenever abs($\lambda - z$) $< r$, where $f^{(i)}$ is the i-th derivative of f.

- In the *Zygmund class* if it is continuous and there is a positive number D with the following property: For each step size $\epsilon > 0$, abs($f(x-h) + f(x+h) - 2f(x)$) $\leq D \times \epsilon$ wherever the left-hand side is defined and $0 < h \leq \epsilon$. The Zygmund class includes the two "smaller" classes of Lipschitz continuous functions (see "Definitions") and functions with a continuous derivative.

7 Notes

8 Appendix

8.1 Examples of Well-Behaved Functions

In the examples given in this section, $f(\lambda)$ is a function defined on the closed unit interval.

Generally, how well-behaved a function is depends on how many continuous derivatives it has, and whether those derivatives are Lipschitz continuous or Hölder continuous, among other things. The following lists several classes of functions, from worst to best behaved and from "largest" to "smallest":

Functions continuous except possibly at one point → Continuous functions → Hölder continuous functions → Lipschitz continuous functions → With continuous first derivative → With Hölder continuous first derivative → With Lipschitz continuous first derivative → With continuous second derivative → With infinitely many derivatives → Analytic functions.

Concave and convex functions. The following table shows examples of functions that are convex, concave, or neither. All these functions are continuous. Also, review the **definitions**.

Function $f(\lambda)$	Convex or concave?	Notes
$1 - \lambda^{2}$.	Concave.	
λ^{2}.	Convex.	
$\lambda^{2} - \lambda^{3}$.	Neither.	
λ^{z}, where $0 < z \leq 1$.	Concave.	
λ^{z}, where $z \geq 1$.	Convex.	
$\exp(-\lambda/4)$.	Concave.	

Hölder and Lipschitz continuous functions. The following table shows some functions that are Hölder continuous and some that are not. Also, review the **definitions**.

Function $f(\lambda)$:	Hölder exponent (α) and an upper bound of the Hölder constant (H):	Notes
$\lambda^{z} \cdot t$.	$\alpha = z$. L=abs(t).	$0 < z \leq 1$, $t \neq 0$.
$\lambda^{z} \cdot t$.	$\alpha = 1$ (Lipschitz continuous). $L = z$*abs(t).	$z \geq 1$, t is a real number.
$\lambda^{1/3}/4 + \lambda^{2/3}/5$.	$\alpha = 1/3$. $L = 9/20$.	α is the minimum of Hölder exponents, min(1/3, 2/3), and L is the sum of Hölder constants, 1/4+1/5.
$1/2 - (1-2\lambda)^{z}/2$ if $\lambda < 1/2$ and $1/2 + (2\lambda - 1)^{z}/2$ otherwise.	$\alpha = z$. $L = 2^{z}/2$.	$0 < z \leq 1$. In this example, f has a "vertical" slope at 1/2, unless z is 1.

Function $f(\lambda)$:	Hölder exponent (α) and an upper bound of the Hölder constant (H):	Notes
$3/4 - \sqrt{\lambda(1-\lambda)}$.	α =1/2.L=1.	Has a "vertical" slope at 0 and 1.
Continuous and **piecewise linear**.	α =1.L is the greatest absolute value of the slope among all pieces' slopes.	$f(\lambda)$ is *piecewise linear* if it's made up of multiple linear functions defined on a finite number of "pieces", or non-empty subintervals, that together make up f's domain (in this case, the closed unit interval).
Piecewise linear; equals 0 at 0, 3/4 at 2/3 and 1/4 at 1, and these points are connected by linear functions.	α =1.L = 1.5.	L = max(abs((3/4 − 0)/(2/3)), abs((1/4 − 3/4)/(1/3))).Concave because the first piece's slope is greater than the second piece's.
min(λ **mult*, 1 − ε).	α =1.L = *mult*.	*mult* > 0, ε > 0. Piecewise linear; equals 0 at 0, 1 − ε at (1 − ε)/*mult*, and 1 − ε at 1.L=max(*mult*, 0).Concave.
1/10 if λ is 0; − 1/(2*ln(λ /2)) + 1/10 otherwise.	Not Hölder continuous.	Has a slope near 0 that's steeper than every "nth" root.

Note: A Hölder continuous function with Hölder exponent α and Hölder constant L is also Hölder continuous with Hölder exponent β and Hölder constant bounded above by L, where $0 < \beta < \alpha$.

Finding parameters α and L for Hölder continuous functions. If $f(\lambda)$ is continuous, the following is one way to find the Hölder exponent (α) and Hölder constant (L) of f, to determine whether f is Hölder continuous, not just continuous.

First, if f has a continuous first derivative on its domain, then α is 1 (f is **Lipschitz continuous**) and L is the maximum of the absolute value of that derivative.

Otherwise, consider the function $h(\lambda, c) = \text{abs}(f(\lambda) - f(c))/((\text{abs}(\lambda - c))^\alpha)$, or 0 if $\lambda = c$, where $0 < \alpha \leq 1$ is a Hölder exponent to test. For a given α, let $g(\lambda)$ be the maximum of $h(\lambda, c)$ over all points c where f has a "vertical slope" or the "steepest slope exhibited". If $g(\lambda)$ is bounded for a given α on f's domain (in this case, the closed unit interval), then f is Hölder continuous with Hölder exponent α and Hölder constant (L) equal to or greater than the maximum value of $g(\lambda)$ on its domain.

The following example, which uses the SymPy computer algebra library, plots $\max(h(\lambda, 0), h(\lambda, 1))$ when $f = \sqrt{\lambda(1-\lambda)}$ and $\alpha = 1/2$: `lamda,c=symbols('lamda c'); func=sqrt(lamda*(1-lamda)); alpha=S(1)/2; h=Abs(func-func.subs(lamda,c))/Abs(lamda-c)**alpha; plot(Max(h.subs(c, 0), h.subs(c,1)), (lamda, 0, 1))`.

Functions with a Hölder continuous or Lipschitz continuous derivative. The following table shows some functions whose derivatives are Hölder continuous, and others where that is not the case. (In the SymPy library, a function's derivative can be found using the `diff` method.) In the table below, if f has a continuous *second* derivative on its domain, then α is 1 (the first derivative is Lipschitz continuous) and L is the maximum of the absolute value of that *second* derivative.

Function $f(\lambda)$	Derivative's Hölder exponent (α) and an upper bound of the derivative's Hölder constant (L):	Notes
$\lambda^{1+\beta}$	$\alpha = \beta$.$L = 1+ \beta$.	$0 < \beta \leq 1$.
$3/4 - \sqrt{\lambda(1-\lambda)}$.	Derivative is not Hölder continuous.	Derivative is not Hölder continuous because f is not Lipschitz continuous.
$\cosh(\lambda) - 3/4$.	α =1 (derivative is Lipschitz continuous).$L = \cosh(1)$.	Continuous second derivative, namely $\cosh(\lambda)$. Convex. `cosh` is the hyperbolic cosine function.
$\lambda \cdot \sin(z\lambda)/4 + 1/2$.	α =1.$L = z(2 + z\lambda)/4$.	Continuous second derivative. L is an upper bound of its absolute value. $z > 0$.
$\sin(z\lambda)/4 + 1/2$.	α =1.$L = (z^2)/4$.	Continuous second derivative; L is an upper bound of its absolute value, namely $\text{abs}(-\sin(z\lambda) \cdot z^2/4)$. $z > 0$.
$\lambda^2/2 + 1/10$ if $\lambda \leq 1/2$; $\lambda/2 - 1/40$ otherwise.	α =1.$L = 1$.	Lipschitz continuous derivative, with Lipschitz constant 1.
$\exp(-\lambda)$.	α =1.$L = 1$.	Lipschitz continuous derivative, with Lipschitz constant 1.[47]
$\lambda/2$ if $\lambda \leq 1/2$; $(4^*\lambda - 1)/(8^*\lambda)$ otherwise.	α =1.L=1.	Concave. Lipschitz continuous derivative with Lipschitz constant 2.

8.2 Results Used in Approximate Bernoulli Factories

See the appendix of the page **"Approximations in Bernstein Form"**[48].

8.3 How Many Coin Flips Are Needed to Simulate a Polynomial?

Let $p(\lambda)$ be a polynomial that maps the closed unit interval to itself and satisfies $0 < p(\lambda) < 1$ whenever $0 < \lambda < 1$.

Then p's *coin-flipping degree* (Wästlund 1999)[49] is the smallest value of n such that p's Bernstein coefficients of degree n lie in the closed unit interval. [50] (This is broader than the use of the term in Wästlund, where a polynomial can have a coin-flipping degree only if its "power" coefficients are integers.) The coin-flipping degree is the smallest value of n such that the algorithm of Goyal and Sigman (2012)[51] can toss heads with probability $p(\lambda)$ using exactly n biased coin flips (in addition to a fair coin).

The following results give upper bounds on p's coin-flipping degree.

Suppose p is in Bernstein form of degree m with Bernstein coefficients $b_0, ..., b_m$. Then:

- If $0 \leq \min(b_0, ..., b_m) \leq \max(b_0, ..., b_m) \leq 1$, then the coin-flipping degree is bounded above by m.

[47] This function's second derivative's absolute value can be plotted using the SymPy library as follows: `plot(diff(Abs(exp(-x)),(x,2)),(x,0,1))`. In this plot, the maximum is 1, the same as the first derivative's Lipschitz constant.

[48] https://peteroupc.github.io/bernapprox.html

[49] Wästlund, J., "**Functions arising by coin flipping**", 1999.

[50] The coin-flipping degree is very similar to the so-called *Bernstein degree* or *Lorentz degree*, which is the smallest integer n such that p's Bernstein coefficients of degree n are all non-negative, assuming that p is non-negative.

[51] Goyal, V. and Sigman, K., 2012. On simulating a class of Bernstein polynomials. ACM Transactions on Modeling and Computer Simulation (TOMACS), 22(2), pp.1-5.

- If $0 \le \min(b_0, ..., b_m)$ and $\max(b_0, ..., b_m) > 1$, then the coin-flipping degree is bounded above by—

$$m + \text{iceil}\left(\frac{m(m-1)}{2}\frac{\max(1-b_0, ..., 1-b_m)}{1-\text{Pmax}} - m\right),$$

where $\text{iceil}(x)$ is $x+1$ if x is an integer, or $\text{ceil}(x)$ otherwise, and where Pmax is the maximum value of $p(\lambda)$ on the closed unit interval (Powers and Reznick 2001)[52].

- If $\min(b_0, ..., b_m) < 0$ and $\max(b_0, ..., b_m) \le 1$, then the coin-flipping degree is bounded above by—

$$m + \text{iceil}\left(\frac{m(m-1)}{2}\frac{\max(b_0, ..., b_m)}{\text{Pmin}} - m\right),$$

where Pmin is the *minimum* value of $p(\lambda)$ on the closed unit interval (Powers and Reznick 2001)[53].

- Suppose $m \ge 2$, that $b_0 = 0$ or $b_m = 0$ or both, and that the following necessary conditions are satisfied (Mok and To 2008; Theorem 1 and Corollary 3)[54]:
  - For every i such that $b_i < 0$, if $b_m = 0$, there must be $j > i$ such that $b_j > 0$.
  - For every i such that $b_i < 0$, if $b_0 = 0$, there must be $j < i$ such that $b_j > 0$.
  - For every i such that $1 - b_i < 0$, if $1 - b_m = 0$, there must be $j > i$ such that $1 - b_j > 0$.
  - For every i such that $1 - b_i < 0$, if $1 - b_0 = 0$, there must be $j < i$ such that $1 - b_j > 0$.

  Then the coin-flipping degree is bounded above by—

$$\max(M(b_0, ..., b_m), M(1-b_0, ..., 1-b_m)),$$

  where—

$$M(\beta_0, ..., \beta_m) = \text{ceil}\left(\max\left(2m, \frac{m(m-1)}{2(1-c)}\frac{a_{\max}}{a_{\min}}\right)\right),$$

  and where:

  - $a_{\max} = \max(\max(0, \beta_0), ..., \max(0, \beta_m))$.
  - $a_{\min}$ is the minimum of $(\beta_i \binom{m}{i})$ over all values of i such that $\beta_i > 0$.
  - c is the smallest number r that satisfies $FN(\lambda)/FP(\lambda) \le r$ where $0 < \lambda < 1$. c can also be a greater number but less than 1.
  - $FP(\lambda) = \sum_{k=0}^{m} \max(0, \beta_k)\binom{m}{k}\lambda^k(1-\lambda)^{m-k}$.
  - $FN(\lambda) = \sum_{k=0}^{m} \text{abs}(\min(0, \beta_k))\binom{m}{k}\lambda^k(1-\lambda)^{m-k}$.

  (Mok and To 2008; Theorem 2 and remark 1.5(v))[55].

  Examples:

  1. Let $p(\lambda) = 1 - 8\lambda + 20\lambda^2 - 13\lambda^3$, a polynomial of degree $m = 3$. p's Bernstein coefficients are $b_0 = 1, b_1 = -5/3, b_2 = 7/3, b_3 = 0$, and its coin-flipping degree is 46 (Wästlund 1999, Example 4.4)[56]. p meets the conditions to use the coin-flipping degree derived from Mok

[52] Powers, V., Reznick, B., "**A new bound for Pólya's Theorem with applications to polynomials positive on polyhedra**", *Journal of Pure and Applied Algebra* 164 (24 October 2001). https://www.sciencedirect.com/science/article/pii/S0022404900001559

[53] Powers, V., Reznick, B., "**A new bound for Pólya's Theorem with applications to polynomials positive on polyhedra**", *Journal of Pure and Applied Algebra* 164 (24 October 2001). https://www.sciencedirect.com/science/article/pii/S0022404900001559

[54] Mok, H-N., To, W-K., "**Effective Pólya semi-positivity for non-negative polynomials on the simplex**", *Journal of Complexity* 24 (2008). https://doi.org/10.1016/j.jco.2008.01.003

[55] Mok, H-N., To, W-K., "**Effective Pólya semi-positivity for non-negative polynomials on the simplex**", *Journal of Complexity* 24 (2008). https://doi.org/10.1016/j.jco.2008.01.003

[56] Wästlund, J., "**Functions arising by coin flipping**", 1999.

and To (2008)[57]. In this case, after some calculations, the coin-flipping degree is bounded above by—

$$\text{ceil}\left(\max\left(\max\left(2\cdot 3, \frac{3(3-1)}{2(1-0.94492)}\frac{7/3}{1}\right), \max\left(2\cdot 3, \frac{3(3-1)}{2(1-0.70711)}\frac{8/3}{1}\right)\right)\right) \le 128.$$

2. An exhaustive search shows that 46 is the highest possible coin-flipping degree for a degree-3 polynomial whose "power" coefficients are integers.
3. The degree-4 polynomial $-43\lambda^4 + 81\lambda^3 - 47\lambda^2 + 9\lambda$ has a coin-flipping degree of 5284.

Note: If a polynomial's "power" coefficients can be rational numbers (ratios of two integers), even a degree-2 polynomial can have an arbitrarily high coin-flipping degree. An example is the family of degree-2 polynomials $r\lambda - r\lambda^2$, where r is a rational number greater than 0 and less than 4.

Lemma: Let $p(\lambda) = a_0\lambda^0 + ... + a_n\lambda^n$ be a polynomial that maps the closed unit interval to itself. Then the values $a_0, ..., a_n$ must sum to a value that is 0 or greater and 1 or less.

Proof: This can be seen by evaluating $p(1) = a_0 + ... + a_n$. If $p(1)$ is less than 0 or greater than 1, then p does not meet the hypothesis of the lemma. [ ]

In the following lemmas, let $p(\lambda) = a_0\lambda^0 + ... + a_n\lambda^n$ be a polynomial that maps the closed unit interval to itself and satisfies $0 < p(\lambda) < 1$ whenever $0 < \lambda < 1$.

Lemma: If p's coin-flipping degree is n, then $|a_i| \le 2^i\binom{n}{i}$.

Proof: Consider the matrix that transforms a polynomial's Bernstein coefficients to "power" coefficients, which is $n \times n$ if the polynomial's degree is n (Ray and Nataraj 2012, eq. (8))[58]. Given the hypothesis of the lemma, each Bernstein coefficient must lie in the closed unit interval and the required matrix size is n, which is p's coin-flipping degree. For each row of the matrix ($0 \le i \le n$), the corresponding "power" coefficient of the polynomial equals a linear combination of that row with a vector of Bernstein coefficients. Thus, the i-th power coefficient equals a_i and its absolute value is bounded above by $\sum_{m=0}^{i}\binom{n}{m}\binom{n-m}{i-m} = 2^i\binom{n}{i}$. [ ]

Lemma: $|a_i| \le |b_i|$, where b_i is the corresponding power coefficient of the following polynomial:

$$q(\lambda) = b_0\lambda^0 + ... + b_n\lambda^n = (T_n(1-2\lambda)+1)/2,$$

and where $T_n(x)$ is the **Chebyshev polynomial of the first kind**[59] of degree n.

See *MathOverflow* for a **proof of this lemma**[60] by Fedor Petrov.

8.4 Proofs for Polynomial-Building Schemes

This section shows mathematical proofs for some of the polynomial-building schemes of this page.

In the following results:

- A *strictly bounded factory function* means a continuous function on the closed unit interval, with a minimum of greater than 0 and a maximum of less than 1.

[57] Mok, H-N., To, W-K., "**Effective Pólya semi-positivity for non-negative polynomials on the simplex**", *Journal of Complexity* 24 (2008). https://doi.org/10.1016/j.jco.2008.01.003

[58] S. Ray, P.S.V. Nataraj, "**A Matrix Method for Efficient Computation of Bernstein Coefficients**", Reliable Computing 17(1), 2012. https://interval.louisiana.edu/reliable-computing-journal/volume-17/reliable-computing-17-pp-40-71.pdf

[59] https://mathworld.wolfram.com/ChebyshevPolynomialoftheFirstKind.html

[60] https://mathoverflow.net/questions/449135

- A function $f(\lambda)$ is *polynomially bounded* if both $f(\lambda)$ and $1 - f(\lambda)$ are greater than or equal to min(λ ^{n}, $(1 - \lambda)^n$) for some integer n (Keane and O'Brien 1994)[61]. For examples, see "**About Bernoulli Factories**[62]".
- A *modulus of continuity* of a function f means a nonnegative and nowhere decreasing function ω on the closed unit interval, for which ω (0) = 0, and for which abs(f(x) − f(y)) ≤ ω (abs($x - y$)) whenever x and y are in f's domain. Loosely speaking, a modulus of continuity ω (δ) is greater than or equal to f's maximum range in a window of size δ .

Lemma 1. Omitted.

Lemma 6(i) of Nacu and Peres (2005)[63] can be applied to continuous functions beyond just Lipschitz continuous functions. This includes the larger class of *Hölder continuous* functions (see "**Definitions**").

Lemma 2. *Let f(λ) be a continuous function that maps the closed unit interval to itself, let X be a hypergeometric(2*n, k, n) random variable, and let $n \geq 1$ be an integer.*

1. *Let ω (x) be a modulus of continuity of f. If ω is continuous and concave, then the expression— abs(**E**[f(X/n)] − f(k/(2*n))), (1)is less than or equal to each of the following:*
   - _ ω (sqrt(1/(8*n − 4)))._
   - _ ω (sqrt(1/(7*n))) if n ≥ 4._
   - _ ω (sqrt(1/(2*n)))._
   - _ ω (sqrt((k/(2*n)) * (1 − k/(2*n)) / (2*n − 1)))._
2. *If f is Hölder continuous with Hölder constant M and with Hölder exponent α such that $0 < \alpha \leq 1$, then the expression (1) is less than or equal to—*
   - *M*(1/(2*n))α/2,*
   - *M*(1/(7*n))α/2 if n ≥ 4, and*
   - *M*(1/(8*n − 4))α/2.*
3. *If f has a Lipschitz continuous derivative with Lipschitz constant M, then the expression (1) is less than or equal to—*
   - *(M/2)*(1/(7*n)) if n ≥ 4, and*
   - *(M/2)*(1/(8*n − 4)).*

Proof.

1. ω is assumed to be nonnegative because absolute values are nonnegative. To prove the first and second bounds: abs(**E**[$f(X/n)$] − $f(k/(2 * n))$) ≤ **E**[abs($f(X/n)$ − $f(k/(2 * n))$] ≤ **E**[ω (abs($X/n - k/(2 * n)$)] (by the definition of ω) ≤ ω (**E**[abs($X/n - k/(2 * n)$)]) (by Jensen's inequality and because ω is concave) ≤ ω (sqrt(**E**[abs($X/n - k/(2 * n)$)]^2)) = ω (sqrt(**Var**[X/n])) = ω (sqrt((k*(2 * n − k)/(4*(2 * n − 1)*n^2)))) ≤ ω (sqrt((n^2/(4*(2 * n − 1)*n^2)))) = ω (sqrt((1/(8*n − 4)))) = ρ , and for every integer $n \geq 4$, $\rho \leq \omega$ (sqrt(1/(7*n))). To prove the third bound: abs(**E**[$f(X/n)$] − $f(k/(2 * n))$) ≤ ω (sqrt(**Var**[X/n])) ≤ ω (sqrt(1/(2*n))). To prove the fourth bound: abs(**E**[$f(X/n)$] − $f(k/(2 * n))$) ≤ ω (sqrt((n^2/(4*(2 * n − 1)*n^2)))) = ω (sqrt(($k/(2*n)$) * (1 − $k/(2*n)$) / (2*n − 1))).
2. By the definition of Hölder continuous functions, take ω (x) = M^*x^α. Because ω is a concave modulus of continuity on the closed unit interval, the result follows from part 1.
3. (Much of this proof builds on Nacu and Peres 2005, Proposition 6(ii)[64].) The expected value (see note 1) of X is $E[X/n] = k/(2n)$. Since $E[X/n - k/(2n)] = 0$, it follows that $f'(X/n)E(X/n - k/(2n)) = 0$.

[61] Keane, M. S., and O'Brien, G. L., "A Bernoulli factory", *ACM Transactions on Modeling and Computer Simulation* 4(2), 1994.

[62] https://peteroupc.github.io/bernoulli.html#About_Bernoulli_Factories

[63] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[64] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

Moreover, $|f(x) - f(s) - f'(x)(x-s)| \leq (M/2)(x-s)^2$ (see Micchelli 1973, Theorem 3.2)[65], so—

$$E[|f(X/n) - f(k/(2n))|] = |E[f(X/n) - f(k/(2n)) - f'(k/(2n))(X/n - k/(2n))]|$$

$$\leq (M/2)(X/n - k/(2n))^2 \leq (M/2)Var(X/n).$$

By part 1's proof, it follows that $(M/2)$*$\mathbf{Var}[X/n] = (M/2)$*$(k$*$(2 * n - k)/(4$*$(2 * n - 1)$*$n^2)) \leq (M/2)$*$(n^2/(4$*$(2 * n - 1)$*$n^2)) = (M/2)$*$(1/(8$*$n - 4)) = \rho$. For every integer $n \geq 4$, $\rho \leq (M/2)$*$(1/(7$*$n))$. [ ]

Notes:

1. $\mathbf{E}[.]$ means expected value ("long-run average"), and $\mathbf{Var}[.]$ means variance. A hypergeometric$(2 * n, k, n)$ random variable is the number of "good" balls out of n balls taken uniformly at random, all at once, from a bag containing $2 * n$ balls, k of which are "good".
2. Parts 1 through 3 exploit a tighter bound on $\mathbf{Var}[X/n]$ than the bound given in Nacu and Peres (2005, Lemma 6(i) and 6(ii), respectively)[66]. However, for technical reasons, different bounds are proved for different ranges of integers n.
3. All continuous functions that map the closed unit interval to itself, including all of them that admit a Bernoulli factory, have a modulus of continuity. The proof of part 1 remains valid even if $\omega(0) > 0$, because the bounds proved remain correct even if ω is overestimated. The following functions have a simple modulus of continuity that satisfies the lemma:
    1. If f is strictly increasing and convex, $\omega(x)$ can equal $f(1) - f(1-x)$ (Gal 1990)[67]; (Gal 1995)[68].
    2. If f is strictly decreasing and convex, $\omega(x)$ can equal $f(0) - f(x)$ (Gal 1990)[69]; (Gal 1995)[70].
    3. If f is strictly increasing and concave, $\omega(x)$ can equal $f(x) - f(0)$ (by symmetry with 2).
    4. If f is strictly decreasing and concave, $\omega(x)$ can equal $f(1-x) - f(1)$ (by symmetry with 1).
    5. If f is concave and is strictly increasing then strictly decreasing, then $\omega(h)$ can equal $(f(\min(h, \sigma))+(f(1 - \min(h, 1 - \sigma)) - f(1))$, where σ is the point where f stops increasing and starts decreasing (Anastassiou and Gal 2012)[71].

There are weaker bounds for Lemma 2, part 1, which work even if f's modulus of continuity ω is not concave. According to Pascu et al. (2017, Lemma 5.1)[72]:

$$|\mathbb{E}[f(Y)] - f(\mathbb{E}[Y])| \leq \omega(\delta) + \omega(\delta)\mathrm{Var}[Y]/\delta^2,$$

where f is a continuous function, Y is a discrete random variable on a closed interval, and $\delta > 0$. Given that $Y = X/n$ (where X is as in Lemma 2), taking $\delta = 1/n^{1/2}$ leads to:

$$|\mathbb{E}[f(Y)] - f(\mathbb{E}[Y])| = |\mathbb{E}[f(Y)] - f(k/(2n))| \leq \omega(1/n^{1/2})(1 + n \cdot \mathrm{Var}[Y]),$$

and in turn, plugging in bounds for $\mathrm{Var}[Y]$ leads to the following bounds for Lemma 2, part 1:

[65] Micchelli, Charles. "The saturation class and iterates of the Bernstein polynomials." Journal of Approximation Theory 8, no. 1 (1973): 1-18.

[66] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[67] Gal, S.G., "Calculus of the modulus of continuity for nonconcave functions and applications", *Calcolo* 27 (1990)

[68] Gal, S.G., 1995. Properties of the modulus of continuity for monotonous convex functions and applications. *International Journal of Mathematics and Mathematical Sciences* 18(3), pp.443-446.

[69] Gal, S.G., "Calculus of the modulus of continuity for nonconcave functions and applications", *Calcolo* 27 (1990)

[70] Gal, S.G., 1995. Properties of the modulus of continuity for monotonous convex functions and applications. *International Journal of Mathematics and Mathematical Sciences* 18(3), pp.443-446.

[71] Anastassiou, G.A., Gal, S.G., *Approximation Theory: Moduli of Continuity and Global Smoothness Preservation*, Birkhäuser, 2012.

[72] Pascu, M.N., Pascu, N.R., Tripşa, F., "**A new Bernstein-type operator based on Pólya's urn model with negative replacement**", arXiv:1710.08818 [math.CA], 2017. https://arxiv.org/abs/1710.08818

- $\omega(1/n^{1/2})(1 + n/(8n - 4))$.
- $\omega(1/n^{1/2})(1 + n/(7n)) = \frac{8}{7}\omega(1/n^{1/2})$ if n $\geq$ 4.
- $\omega(1/n^{1/2})(1 + n/(2n)) = \frac{3}{2}\omega(1/n^{1/2})$.

Lemma 2A. _Let f(λ) map the closed unit interval to itself, and let $C = 15$. Suppose f is in the Zygmund class with constant D or less. Then, for every integer $n \geq 1$, the expression (1) in Lemma 2 is less than or equal to $(C/2)D\sqrt{1/(8n - 4)}$.

Proof. Strukov and Timan (1977)[73] proved the following bound:

$$|\mathbb{E}[f(Y)] - f(\mathbb{E}[Y])| \leq C\omega_2((\mathrm{Var}[Y])^{1/2}/2),$$

where Y is a random variable and $\omega_2(.)$ is a *second-order modulus of continuity* of f (see note below), and where C is 3 if Y takes on any value in the real line, or 15 if Y takes on only values in a closed interval, such as the closed unit interval in this case.

Suppose $Y = X/n$, where X is as in Lemma 2. Then Y's variance (**Var**[Y]) is less than or equal to 1/(8*n − 4), and the left-hand side of Strukov and Timan's bound is the same as the expression (1).

Since f is in the Zygmund class, there is an ω_2 for it such that $\omega_2(h) \leq Dh$. Therefore, applying Strukov and Timan's bound and the bound on Y's variance leads to—

$$\mathrm{abs}(\mathbb{E}[f(Y)] - f(\mathbb{E}[Y])) \leq C\omega_2((\mathrm{Var}[Y])^{1/2}/2)$$

$$\leq CD((\mathrm{Var}[Y])^{1/2}/2) = CD\sqrt{1/(8n - 4)}/2.$$

[ ]

> **Note:** A *second-order modulus of continuity* is a nonnegative and nowhere decreasing function $\omega_2(h)$ with $h \geq 0$, for which $\omega_2(0) = 0$, and for which $\mathrm{abs}(f(x) + f(y) - 2f((x + y)/2)) \leq \omega_2(\mathrm{abs}((y - x)/2))$ whenever f is continuous and x and y are in f's domain.

Theorem 1. *Let f be a strictly bounded factory function, let $n_0 \geq 1$ be an integer, and let $\phi(n)$ be a function that takes on a nonnegative value. Suppose f is such that the expression (1) in Lemma 2 is less than or equal to $\phi(n)$ whenever $n \geq n_0$ is an integer power of 2. Let—*

$$\eta(n) = \sum_{k \geq \log_2(n)} \phi(2^k),$$

for every integer $n \geq 1$ that's a power of 2. If the series (n) converges to a finite value for each such n, and if it converges to 0 as n gets large, then the following scheme for f(λ) is valid in the following sense:

*There are polynomials g_n and h_n (where $n \geq 1$ is an integer power of 2) as follows. The k-th Bernstein coefficient of g_n and h_n is **fbelow**(n, k) and **fabove**(n, k), respectively (where $0 \leq k \leq n$), where:*

If $n_0 = 1$:

- ***fbelow**(n, k) = $f(k/n) - \eta(n)$.*
- ***fabove**(n, k) = $f(k/n) + \eta(n)$.*

If $n_0 > 1$:

- ***fbelow**(n, k) = $\min($**fbelow**$(n_0,0)$, **fbelow**$(n_0,1)$, …, **fbelow**$(n_0,n_0))$ if $n < n_0$; $f(k/n) - \eta(n)$ otherwise.*
- ***fabove**(n, k) = $\max($**fabove**$(n_0,0)$, **fabove**$(n_0,1)$, …, **fbelow**$(n_0,n_0))$ if $n < n_0$; $f(k/n) + \eta(n)$ otherwise.*

[73] Strukov, L.I., Timan, A.F., "Mathematical expectation of continuous functions of random variables. Smoothness and variance", *Siberian Mathematical Journal* 18 (1977).

The polynomials g_n and h_n satisfy:

1. $g_n \leq h_n$.
2. g_n *and* h_n *converge to* f *as* n *gets large.*
3. $(g_{n+1} - g_n)$ *and* $(h_n - h_{n+1})$ *are polynomials with nonnegative Bernstein coefficients once they are rewritten to polynomials in Bernstein form of degree exactly* $n + 1$.

Proof. For simplicity, this proof assumes first that $n_0 = 1$.

For the series $\eta\ (n)$ in the theorem, because $\phi(n)$ is nonnegative, each term of the series is nonnegative making the series nonnegative and, by the assumption that the series converges, $\eta\ (n)$ is nowhere increasing with increasing n.

Item 1 is trivial. If $n \geq n_0$, g_n is simply the Bernstein polynomial of f minus a nonnegative value, and h_n is the Bernstein polynomial of f plus that same value, and if n is less than n_0, g_n is a constant value not less than the lowest point reachable by the lower polynomials, and h_n is a constant value not less than the highest point reachable by the upper polynomials.

Item 2 is likewise trivial. A well known result is that the Bernstein polynomials of f converge to f as their degree n gets large. And because the series η (in Theorem 1) sums to a finite value that goes to 0 as n increases, the upper and lower shifts will converge to 0 so that g_n and h_n converge to the degree-n Bernstein polynomials and thus to f.

Item 3 is the *consistency requirement* described earlier in this page. This is ensured as in Proposition 10 of Nacu and Peres (2005)[74] by bounding, from below, the offset by which to shift the approximating polynomials. This lower bound is $\eta\ (n)$, a solution to the equation $0 = \eta\ (n) - \eta\ (2\ *\ n) - \phi\ (n)$ (see note below), where $\phi\ (n)$ is a function that takes on a nonnegative value.

$\phi\ (n)$ is, roughly speaking, the minimum distance between one polynomial and the next so that the consistency requirement is met between those two polynomials. Compare the assumptions on ϕ in Theorem 1 with equations (10) and (11) in Nacu and Peres (2005).

The solution for $\eta(n)$ given in the statement of the theorem is easy to prove by noting that this is a recursive process: we start by calculating the series for $n = 2^*n$, then adding $\phi\ (n)$ to it (which will be positive), in effect working backwards and recursively, and we can easily see that we can calculate the series for $n = 2^*n$ only if the series converges, hence the assumption of a converging series.

Now to prove the result assuming that $n_0 > 1$.

Doing this involves taking advantage of the observation in Remark B of Nacu and Peres (2005)[75] that we can start defining the polynomials at any n greater than 0, including $n = n_0$; in that case, the upper and lower polynomials of degree 1 or greater, but less than n_0, would be constant functions, so that the Bernstein coefficients of each polynomial would be equal. The lower constants are no greater than g_{n_0}'s lowest Bernstein coefficient, and the upper constants are no less than g_{n_0}'s highest Bernstein coefficients; they meet Item 3 because these lower and upper constants, when rewritten as polynomials in Bernstein form of degree n_0, have Bernstein coefficients that are still no greater or no less, respectively, than the corresponding degree-n_0 polynomial. With the ϕ given in this theorem, the series $\eta\ (n)$ in the theorem remains nonnegative. Moreover, since η is assumed to converge, $\eta\ (n)$ still decreases with increasing n. [ ]

Notes:

1. There is only one solution $\eta\ (n)$ in the case at hand. Unlike so-called ***functional equations***[76] and linear recurrences, with a solution that varies depending on the starting value,

[74] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[75] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[76] https://math.stackexchange.com/questions/3993739

there is only one solution in the case at hand, namely the solution that makes the series converge, if it exists at all. Alternatively, the equation can be expanded to $0 = \eta\ (n) - \eta\ (4\ *\ n) - \phi\ (2{*}n) - \phi\ (n) = \eta\ (n) - \eta\ (8\ *\ n) - \phi\ (4{*}n) - \phi\ (2{*}n) - \phi\ (n) = \ldots$

2. $\log_2(n)$ is the number x such that $2^x = n$.

Proposition 1A. *If a scheme satisfies Theorem 1, the polynomials g_n and h_n in the scheme can be made to satisfy conditions (i), (iii), and (iv) of Proposition 3 of Nacu and Peres (2005)*[77] *as follows:*

- *$g_n = g_{n-1}$ and $h_n = h_{n-1}$ whenever n is an integer greater than 1 and not a power of 2.*
- *If **fabove**$(n, k) > 1$ for a given n and some k, the Bernstein coefficients of h_n (the upper polynomial) are all 1.*
- *If **fbelow**$(n, k) < 0$ for a given n and some k, the Bernstein coefficients of g_n (the lower polynomial) are all 0.*

Proof: Condition (i) of Proposition 3 says that each Bernstein coefficient of the polynomials must be 0 or greater and 1 or less. This is ensured starting with a large enough value of n greater than 0 that's a power of 2, call it n_1, as shown next.

Let ε be a positive distance between 0 and the minimum or between 1 and the maximum of f, whichever is smaller. This ε exists by the assumption that f is bounded away from 0 and 1. Because the series η (in Theorem 1) sums to a finite value that goes to 0 as n increases, $\eta\ (n)$ will eventually stay less than ε . And if $n \geq n_0$ is a power of 2 (where n_0 is as in Theorem 1), the `f(k/n)` term is bounded by the minimum and maximum of f by construction. This combined means that the lower and upper polynomials' Bernstein coefficients will eventually be bounded by 0 and 1 for every integer n starting with n_1.

For n less than n_1, condition (i) is ensured by setting the lower or upper polynomial's Bernstein coefficient to 0 or 1, respectively, whenever a Bernstein coefficient of the degree-n polynomial would otherwise be less than 0 or greater than 1, respectively.

Condition (iii) of Proposition 3 is mostly ensured by item 2 of Theorem 1. The only thing to add is that for n less than n_1, the lower and upper polynomials g_n and h_n can be treated as 0 or 1, respectively, without affecting convergence, and that for n other than a power of 2, defining $g_n = g_{n-1}$ and $h_n = h_{n-1}$ maintains condition (iii) by Remark B of Nacu and Peres (2005)[78].

Condition (iv) of Proposition 3 is mostly ensured by item 3 of Theorem 1. For $n{=}n_1$, condition (iv) is maintained by noting that the degree-n_1 polynomial's Bernstein coefficients must be bounded by 0 and 1 by condition (i) so they will likewise be bounded by those of the lower and upper polynomials of degree less than n_1, and those polynomials are the constant 0 and the constant 1, respectively, as are their Bernstein coefficients. Finally, for n other than a power of 2, defining $g_n = g_{n-1}$ and $h_n = h_{n-1}$ maintains condition (iv) by Remark B of Nacu and Peres (2005)[79]. [ ]

> **Note:** The last condition of Proposition 3, condition (ii), says **fabove**(n, k)*choose(n,k) and **fbelow**(n, k)*choose(n,k) must be integers. [80] But Proposition 3 assumes only the biased coin and no other randomness is used, and that the coin doesn't show heads every time or tails every time. Therefore, $f(0)$, if it exists, must be an integer, and the same is true for $f(1)$, so that

[77] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[78] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[79] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[80] choose$(n, k) = (1{*}2{*}3{*}\ldots{*}n)/((1{*}\ldots{*}k){*}(1{*}\ldots{*}(n-k))) = n!/(k!\ *\ (n-k)!) = \binom{n}{k}$ is a *binomial coefficient*, or the number of ways to choose k out of n labeled items. It can be calculated, for example, by calculating $i/(n - i{+}1)$ for each integer i satisfying $n - k{+}1 \leq i \leq n$, then multiplying the results (Yannis Manolopoulos. 2002. "Binomial coefficient computation: recursion or iteration?", SIGCSE Bull. 34, 4 (December 2002), 65–67. DOI: **https://doi.org/10.1145/820127.820168**). For every $m{>}0$, choose$(m, 0)$ = choose$(m, m) = 1$ and choose$(m, 1)$ = choose$(m, m - 1) = m$; also, in this document, choose(n, k) is 0 when k is less than 0 or greater than n.$n! = 1{*}2{*}3{*}\ldots{*}n$ is also known as n factorial; in this document, $(0!) = 1$.

condition (ii) is redundant with condition (iii) due to a result that goes back to Kantorovich (1931)[81]; see also Remark C of Nacu and Peres (2005)[82].

Corollary 1. *Let f(λ) be a strictly bounded factory function. If that function is Hölder continuous with Hölder constant M and Hölder exponent α, then the following scheme determined by* ***fbelow*** *and* ***fabove*** *is valid in the sense of Theorem 1:*

- ***fbelow****(n, k) = f(k/n) − D(n).*
- ***fabove****(n, k) = f(k/n) + D(n).*

Where $D(n) = \frac{M}{((2^{\alpha/2}-1)n^{\alpha/2}}$.

Or:

- ***fbelow****(n, k) = min(****fbelow****(4,0),* ***fbelow****(4,1), …,* ***fbelow****(4,4)) if n < 4; otherwise, f(k/n) − (n).*
- ***fabove****(n, k) = max(****fabove****(4,0),* ***fabove****(4,1), …,* ***fabove****(4,4)) if n < 4; otherwise, f(k/n) + (n).*

Where $\eta(n) = M(2/7)^{\alpha-2}/((2^{\alpha/2}-1)n^{\alpha/2})$.

Proof. Because f is Hölder continuous, it admits the modulus of continuity $\omega(x) = Mx^{\alpha}$. By part 1 of lemma 2:

- For each integer $n \geq 1$ that's a power of 2 ($n_0 = 1$ in Theorem 1), $\phi(n) = \omega(\sqrt{1/(2n)}) = M(1/(2n))^{\alpha/2}$ can be taken for each such integer n, and thus $\eta(n) = D(n) = \frac{M}{((2^{\alpha/2}-1)n^{\alpha/2}}$ (where $\eta(n)$ is as in Theorem 1).
- For each integer $n \geq 4$ that's a power of 2 ($n_0 = 4$ in Theorem 1), $\phi(n) = \omega(\sqrt{1/(2n)}) = M(1/(7n))^{\alpha/2}$ can be taken for each such integer n, and thus $\eta(n) = \mathrm{M}^*(2/7)^{\alpha/2}/((2^{\alpha/2} - 1)^*\mathrm{n}^{\alpha/2})$.

In both cases $\eta(n)$ is finite and converges to 0 as n increases.

The result then follows from Theorem 1. [ ]

Note: For specific values of α , the equation $D(n) = D(2 * n) + \phi\ (n)$ can be solved via linear recurrences; an example for $\alpha = 1/2$ is the following code in Python that uses the SymPy computer algebra library: `alpha=(S(1)/2); rsolve(Eq(f(n), f(n+1)+z*(1/(2*2**n))**(alpha/2)), f(n)).subs(n,ln(n,2)).simplify()`. Trying different values of α suggested the following formula for Hölder continuous functions with α of $1/j$ or greater: $(M^* \sum_{i=0}^{2j-1} 2^{i/(2j)})/n^{\ 1/(2)} = M\ /\ ((2^{\ 1/(2)} - 1)^* n^{\ 1/(2)}\ )$; and generalizing the latter expression led to the term in the theorem.

Corollary 2. *Let f(λ) be a strictly bounded factory function. If that function is Lipschitz continuous with Lipschitz constant M, then the following scheme determined by* ***fbelow*** *and* ***fabove*** *is valid in the sense of Theorem 1:*

- ***fbelow****(n, k) = f(k/n) − M/((sqrt(2) − 1)*sqrt(n)).*
- ***fabove****(n, k) = f(k/n) + M/((sqrt(2) − 1)*sqrt(n)).*

Or:

- ***fbelow****(n, k) = min(****fbelow****(4,0),* ***fbelow****(4,1), …,* ***fbelow****(4,4)) if n < 4; otherwise, f(k/n) − M*sqrt(2/7)/((sqrt(2) − 1)*sqrt(n)).*
- ***fabove****(n, k) = max(****fabove****(4,0),* ***fabove****(4,1), …,* ***fabove****(4,4)) if n < 4; otherwise, f(k/n) + M*sqrt(2/7)/((sqrt(2) − 1)*sqrt(n)).*

Proof. Because Lipschitz continuous functions are Hölder continuous with Hölder constant M and exponent 1, the result follows from Corollary 1. [ ]

[81] Kantorovich, L.V., "Some remarks on the approximation of functions by means of polynomials with integer coefficients", 1931.

[82] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

Note: The first scheme given here is a special case of Theorem 1 that was already found by Nacu and Peres (2005)[83].

Corollary 3. *Let f(λ) be a strictly bounded factory function. If that function has a Lipschitz continuous derivative with Lipschitz constant L, then the following scheme determined by* ***fbelow*** *and* ***fabove*** *is valid in the sense of Theorem 1:*

- ***fbelow****(n, k) = min(****fbelow****(4,0),* ***fbelow****(4,1), …,* ***fbelow****(4,4)) if n < 4; otherwise, f(k/n) − L/(7*n).*
- ***fabove****(n, k) = max(****fabove****(4,0),* ***fabove****(4,1), …,* ***fabove****(4,4)) if n < 4; otherwise, f(k/n) + L/(7*n).*

Proof. By part 3 of lemma 2, for each integer $n \geq 4$ that's a power of 2 ($n_0 = 4$ in Theorem 1), $\phi(n) = (L/2)(1/(7n))$ can be taken for each such integer n, and thus $\eta(n) = L/(7n)$ (where $\eta(n)$ is as in Theorem 1). $\eta(n)$ is finite and converges to 0 as n increases. The result then follows from Theorem 1. [ ]

Note: Nacu and Peres (2005)[84] already proved a looser scheme in the case when f has a second derivative on the closed unit interval that is not greater than a constant (a slightly stronger condition than having a Lipschitz continuous derivative on that domain).

Theorem 2. _Let f(λ) be a strictly bounded factory function. If that function is convex and nowhere decreasing, then Theorem 1 remains valid with φ (n) = **E**[f(Y/n)] (where Y is a hypergeometric(2*n, n, n) random variable), rather than as given in that theorem._

Proof. Follows from Theorem 1 and part 4 of Lemma 2 above. With the ϕ given in this theorem, the series η (n) in Theorem 1 remains nonnegative; also, this theorem adopts Theorem 1's assumption that the series converges, so that η (n) still decreases with increasing n. [ ]

Proposition 1.

1. *Let f be as given in Theorem 1 or 2 or Corollary 1 to 3, except that f must be concave and polynomially bounded and may have a minimum of 0. Then the schemes of those results remain valid if* ***fbelow****(n, k) = f(k/n), rather than as given in those results.*
2. *Let f be as given in Theorem 1 or 2 or Corollary 1 to 3, except that f must be convex and polynomially bounded and may have a maximum of 1. Then the schemes of those results remain valid if* ***fabove****(n, k) = f(k/n), rather than as given in those results.*
3. *Theorems 1 and 2 and Corollaries 1 to 3 can be extended to all integers n ≥ 1, not just those that are powers of 2, by defining—*
   - ***fbelow****(n, k) = (k/n)**** fbelow****(n − 1, max(0, k − 1)) + ((n − k)/n)**** fbelow****(n − 1, min(n − 1, k)), and*
   - ***fabove****(n, k) = (k/n)**** fabove****(n − 1, max(0, k − 1)) + ((n − k)/n)**** fabove****(n − 1, min(n − 1, k)),*

   for every integer n ≥ 1 other than a power of 2. Parts 1 and 2 of this proposition still apply to the modified scheme.

Proof. Parts 1 and 2 follow from Theorem 1 or 2 or Corollary 1 to 3, as the case may be. For part 1, the lower polynomials are replaced by the degree-n Bernstein polynomials of f, and they meet the conditions in those theorems by Jensen's inequality. For part 2, the upper polynomials are involved instead of the lower polynomials. Part 3 also follows from Remark B of Nacu and Peres (2005)[85]. [ ]

The following lemma shows that if a scheme for $f(\lambda)$ shifts polynomials upward and downward, the pre-shifted polynomials are close to $f(\lambda)$ by the amount of the shift.

[83] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[84] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[85] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

Lemma 3. *Let f be a strictly bounded factory function. Let S be an infinite set of positive integers. For each integer $n \geq 1$, let $W_n(\lambda)$ be a function, and let $\epsilon_n(f)$ be a nonnegative constant that depends on f and n. Suppose that there are polynomials g_n and h_n (for each n in S) as follows:*

1. *g_n and h_n have Bernstein coefficients $W_n(k/n) - \epsilon_n(f)$ and $W_n(k/n) + \epsilon_n(f)$, respectively ($0 \leq k \leq n$).*
2. *$g_n \leq h_n$.*
3. *g_n and h_n converge to f as n gets large.*
4. *$(g_m - g_n)$ and $(h_n - h_m)$ are polynomials with nonnegative Bernstein coefficients once they are rewritten to polynomials in Bernstein form of degree exactly m, where m is the smallest number greater than n in S.*

Then for each n in S, $|f(\lambda) - B_n(W_n(\lambda))| \leq \epsilon_n(f)$ whenever $0 \leq \lambda \leq 1$, where $B_n(W_n(\lambda))$ is the Bernstein polynomial of degree n of the function $W_n(\lambda)$.

Proof: $W_n(k/n)$ is the k-th Bernstein coefficient of $B_n(W_n(\lambda))$, which is g_n and h_n before they are shifted downward and upward, respectively, by $\epsilon_n(f)$. Moreover, property 4 in the lemma corresponds to condition (iv) of Nacu and Peres (2005)[86], which implies that, for every $m > n$, $g_n(\lambda) \leq g_m(\lambda) \leq f(\lambda)$ (the lower polynomials "increase") and $h_n(\lambda) \geq h_m(\lambda) \geq f(\lambda)$ (the upper polynomials "decrease") for every $n \geq 1$ (Nacu and Peres 2005, Remark A)[87].

Then if $B_n(W_n(\lambda)) < f(\lambda)$ for some λ in the closed unit interval, shifting the left-hand side upward by $\epsilon_n(f)$ (a nonnegative constant) means that $h_n = B_n(W_n(\lambda)) + \epsilon_n(f) \geq f(\lambda)$, and rearranging this expression leads to $f(\lambda) - B_n(W_n(\lambda)) \leq \epsilon_n(f)$.

Likewise, if $B_n(W_n(\lambda)) > f(\lambda)$ for some λ in the closed unit interval, shifting the left-hand side downward by $\epsilon_n(f)$ means that $g_n = B_n(W_n(\lambda)) - \epsilon_n(f) \leq f(\lambda)$, and rearranging this expression leads to $B_n(W_n(\lambda)) - f(\lambda) \leq \epsilon_n(f)$.

This combined means that $|f(x) - B_n(W_n(\lambda))| \leq \epsilon_n(f)$ whenever $0 \leq \lambda \leq 1$. [ ]

Corollary 4. *If $f(\lambda)$ satisfies a scheme given in Theorem 1 with $n_0 \geq 1$, then $B_n(f(\lambda))$ comes within $\eta(n)$ of f for every integer $n \geq n_0$ that's a power of 2; that is, $|B_n(f(\lambda))| \leq \eta(n)$ for every such n.*

Lemma 5. Let $n \geq 1$ be an integer. Suppose g_{2n} and g_n are polynomials in Bernstein form of degree $2n$ and n, respectively, and their domain is the closed unit interval. Suppose g_{2n} and g_n satisfy the property:

- $(g_{2n} - g_n)$ is a polynomial with nonnegative Bernstein coefficients once it is rewritten to a polynomial in Bernstein form of degree exactly $2n$.

Then for every $x \geq 0$, $g_{2n} + x$ and g_n satisfy that property, and for every $x \geq 1$, $g_{2n} \cdot x$ and g_n do as well.

The proof follows from two well-known properties of polynomials in Bernstein form: adding x to g_{2n} amounts to adding x to its Bernstein coefficients, and multiplying g_{2n} by x amounts to multiplying its Bernstein coefficients by x. In either case, g_{2n}'s Bernstein coefficients become no less than they otherwise would, so that $(g_{2n} - g_n)$ continues to have non-negative Bernstein coefficients as required by the property.

It is also true that, for every $x \geq 0$, $g_{2n} \cdot x$ and $g_n \cdot x$ satisfy the same property, but a detailed proof of this is left as an exercise to anyone interested. (If $x = 0$, $g_n \cdot x = g_{2n} \cdot x = 0$, so that the property is trivially satisfied.)

Finally, it is true that, for every real number x, $g_{2n} + x$ and $g_n + x$ satisfy the same property, but, again, a detailed proof of this is left as an exercise to anyone interested. (If $x = 0$, $g_n + x = g_n$ and $g_{2n} + x = g_{2n}$, so that the property is trivially satisfied.)

[86] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[87] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

8.4.1 A Conjecture on Polynomial Approximation

The following conjecture suggests there may be a way to easily adapt other approximating polynomials, besides the ordinary Bernstein polynomials, to the Bernoulli factory problem.

Conjecture.

Let $r \geq 1$, and let f be a strictly bounded factory function whose r-th derivative is continuous. Let M be the maximum of the absolute value of f and its derivatives up to the r-th derivative. Let $W_{2^0}(\lambda), W_{2^1}(\lambda), ..., W_{2^n}(\lambda), ...$ be functions on the closed unit interval that converge uniformly to f (that is, for every tolerance level, all W_{2^i} after some value i are within that tolerance level of f at all points on the closed unit interval).

For each integer $n \geq 1$ that's a power of 2, suppose that there is $D > 0$ such that—

$$|f(\lambda) - B_n(W_n(\lambda))| \leq DM/n^{r/2},$$

whenever $0 \leq \lambda \leq 1$, where $B_n(W_n(\lambda))$ is the degree-n Bernstein polynomial of $W_n(\lambda)$.

Then there is $C_0 \geq D$ such that for every $C \geq C_0$, there are polynomials g_n (for each $n \geq 1$) as follows:

1. g_n has Bernstein coefficients $W_n(k/n) - CM/n^{r/2}$ $(0 \leq k \leq n)$, if n is a power of 2, and $g_n = g_{n-1}$ otherwise.
2. g_n converges to f as n gets large.
3. $(g_{n+1} - g_n)$ is a polynomial with nonnegative Bernstein coefficients once it is rewritten to a polynomial in Bernstein form of degree exactly $n + 1$.

Equivalently (see also Nacu and Peres 2005), there is $C_1 > 0$ such that, for each integer $n \geq 1$ that's a power of 2—

$$\left|\left(\sum_{i=0}^{k} W_n\left(\frac{i}{n}\right)\sigma_{n,k,i}\right) - W_{2n}\left(\frac{k}{2n}\right)\right| \leq \frac{C_1 M}{n^{r/2}}, \tag{PB}$$

whenever $0 \leq k \leq 2n$, so that $C = \frac{C_1}{1-\sqrt{2/2^{r+1}}}$. Here, $\sigma_{n,k,i} = \binom{n}{i}\binom{n}{k-i}/\binom{2n}{k}$ is the probability that a hypergeometric(2*n, k, n) random variable equals i.

It is further conjectured that the same value of C_0 (or C_1) suffices when f has a Lipschitz continuous $(r-1)$-th derivative and M is the maximum of the absolute value of f and the Lipschitz constants of f and its derivatives up to the $(r-1)$-th derivative.

Notes:

1. If $W_n(0) = f(0)$ and $W_n(1) = f(1)$ for every n, then (PB) is automatically true when $k = 0$ and $k = 2n$, so that the statement has to be checked only for $0 < k < 2n$. If, in addition, W_n is symmetric about 1/2, so that $W_n(\lambda) = W_n(1 - \lambda)$ whenever $0 \leq \lambda \leq 1$, then the statement has to be checked only for $0 < k \leq n$ (since the values $\sigma_{n,k,i}$ are symmetric in that they satisfy $\sigma_{n,k,i} = \sigma_{n,k,k-i}$).
2. If W_n is a "linear operator", the left-hand side of (PB) is not greater than $|(\sum_{i=0}^{k}(W_n(\frac{i}{n}))\sigma_{n,k,i}) - W_n(k/(2n))| + |W_n(k/(2n)) - f(k/(2n))| + |W_{2n}(k/(2n)) - f(k/(2n))|$.
3. If W_n is a "linear operator" and satisfies $|f(\lambda) - W_n(\lambda)| \leq DM/n^{r/2}$, then the left-hand side of (PB) is not greater than $|(\sum_{i=0}^{k}(W_n(\frac{i}{n}))\sigma_{n,k,i}) - W_n(k/(2n))| + \frac{DM(2^{r/2}+1)}{2^{r/2}}\frac{1}{n^{r/2}}$.
4. By Lemma 3, $B_n(W_n(f(\lambda)))$ would be close to $f(\lambda)$ by at most $C_0 M/n^{r/2}$. Properties 2 and 3 above correspond to (iii) and (iv) in Nacu and Peres (2005, Proposition 3)[88].

[88] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

The following lower bounds on C_0, given certain polynomials, can be shown. In the table:

- M_r is the maximum of the absolute value of $f(\lambda)$ and its derivatives up to the r-th derivative.
- The bounds are valid only if n is a power-of-two integer and, unless otherwise specified, only if $n \geq 1$.

For a description of the polynomials in the third column, see **"Approximations in Bernstein Form"**[89].

If r is…	And…	With the following polynomial's Bernstein coefficients:	Then C_0 must be greater than:	And C_0 is conjectured to be:	Because of this counterexample:
3	$M = M_3$	$U_{n,2}$	0.29004	$\frac{3}{16-4\sqrt{2}} <$ 0.29005.	$2\lambda\,(1-\lambda)$
3	$M = M_3$, $n \geq 4$	$U_{n,2}$	0.08287	0.09	$2\lambda\,(1-\lambda)$
4	$M = M_4$	$U_{n,2}$	0.24999	0.25	$2\lambda\,(1-\lambda)$
4	$M = M_4$, $n \geq 4$	$U_{n,2}$	0.14	0.15	$2\lambda\,(1-\lambda)$
5	$M = M_5$	$U_{n,3}$	0.26	0.27	$2\lambda\,(1-\lambda)$
5	$M = M_5$, $n \geq 4$	$U_{n,3}$	0.1226	0.13	λ^3
6	$M = M_6$	$U_{n,3}$	0.25	0.26	λ^3
6	$M = M_6$, $n \geq 4$	$U_{n,3}$	0.25	0.26	λ^3
3	$M = M_3$, $n \geq 8$	$L_{2,n/2}$	0.0414	0.08	$\frac{1}{2} - (1-2\lambda)^{3.00001}/2$ if $\lambda < 1/2$; $\frac{1}{2} - (2\lambda-1)^{3.00001}/2$ otherwise.

8.4.2 Example of Polynomial-Building Scheme

The following example uses the results above to build a polynomial-building scheme for a factory function.

Let $f(\lambda) = 0$ if λ is 0, and $(\ln(\lambda\ /\exp(3)))\ ^{-2}$ otherwise. (This function is not Hölder continuous; its slope is exponentially steep at the point 0.) Then the following scheme is valid in the sense of Theorem 1:

- $\eta\ (k) = \Phi(1, 2, (\ln(k)+\ln(7)+6)/\ln(2))*4/\ln(2)^2$.
- **fbelow**(n, k) = f(k/n).
- **fabove**(n, k) = max(**fabove**(4,0), **fabove**(4,1), …, **fabove**(4,4)) if n < 4; otherwise, f(k/n) + $\eta\ (n)$.

Where $\Phi(.)$ is a function called the *Lerch transcendent*.

The first step is to find a concave modulus of continuity of f (called $\omega\ (h)$). Because f is strictly increasing and concave, and because $f(0) = 0$, $\omega\ (h) = f(h)$ can be taken for ω .

Now, plugging sqrt(1/(7*n)) into ω leads to the following for Theorem 2 (assuming $n \geq 0$):

- $\phi\ (n) = 1/(\ln(\text{sqrt}(7/n)/7) - 3)^2$.

Now, by applying Theorem 1, compute $\eta\ (k)$ by substituting n with 2^n, summing over $[k, \infty)$, and substituting k with $\ln(k)/\ln(2)$. η converges, resulting in:

[89] https://peteroupc.github.io/bernapprox.html

- $\eta\ (k) = \Phi(1, 2, (\ln(k)+\ln(7)+6)/\ln(2))*4/\ln(2)^2$,

where $\Phi(.)$ is the Lerch transcendent. This η matches the η given in the scheme above. That scheme then follows from Theorems 1 and 2, as well as from part 1 of Proposition 1 because f is concave.

the following code in Python that uses the SymPy computer algebra library is an example of finding the parameters for this polynomial-building scheme.

```
px=Piecewise((0,Eq(x,0)),((ln(x/exp(3))**-2),True))
<h1>omega is modulus of continuity.  Since</h1>

<h1>px is strictly increasing, concave, and px(0)=0,</h1>

<h1>take omega as px</h1>

omega=px
omega=piecewise_fold(omega.rewrite(Piecewise)).simplify()
<h1>compute omega</h1>

phi=omega.subs(x,sqrt(1/(7*n)))
pprint(phi)
<h1>compute eta</h1>

eta=summation(phi.subs(n,2**n),(n,k,oo)).simplify()
eta=eta.subs(k,log(k,2)) # Replace k with ln(k)/ln(2)
pprint(eta)
for i in range(20):
  # Calculate upper bounds for eta at certain points.
  try:
    print("eta(2^%d) ~= %s" % (i,ceiling(eta.subs(k,2**i)*10000000).n()/10000000))
  except:
    print("eta(2^%d) ~= [FAILED]" % (i))
```

8.5 Which functions don't require outside randomness to simulate?

The function $f(\lambda)$ is *strongly simulable* if it admits a Bernoulli factory algorithm that uses nothing but the input coin as a source of randomness (Keane and O'Brien 1994)[90]. See "**Randomized vs. Non-Randomized Algorithms**[91]".

Strong Simulability Statement. A function $f(\lambda)$ is strongly simulable only if—

1. f is constant on its domain, or is continuous and polynomially bounded on its domain, and
2. f maps the closed unit interval or a subset of it to the closed unit interval, and
3. $f(0)$ equals 0 or 1 whenever 0 is in the domain of f, and
4. $f(1)$ equals 0 or 1 whenever 1 is in the domain of f.

Keane and O'Brien already showed that f is strongly simulable if conditions 1 and 2 are true and neither 0 nor 1 are included in the domain of f. Conditions 3 and 4 are required because λ (the probability of heads) can be 0 or 1 so that the input coin returns 0 or 1, respectively, every time. This is called a "degenerate" coin. When given just a degenerate coin, no algorithm can produce one value with probability greater than 0, and another value with the opposite probability. Rather, the algorithm can only produce a constant value with probability 1. In the Bernoulli factory problem, that constant is either 0 or 1, so a Bernoulli factory

[90] Keane, M. S., and O'Brien, G. L., "A Bernoulli factory", *ACM Transactions on Modeling and Computer Simulation* 4(2), 1994.

[91] https://peteroupc.github.io/bernoulli.html#Randomized_vs_Non_Randomized_Algorithms

algorithm for f must return 1 with probability 1, or 0 with probability 1, when given just a degenerate coin and no outside randomness, resulting in conditions 3 and 4.

To show that f is strongly simulable, it's enough to show that there is a Bernoulli factory for f that must flip the input coin and get 0 and 1 before it uses any outside randomness.

Proposition 1. *If $f(\lambda)$ is described in the strong simulability statement and is a polynomial with computable Bernstein coefficients, it is strongly simulable.*

Proof: If f is the constant 0 or 1, the proof is trivial: simply return 0 or 1, respectively.

Otherwise: Let $a[j]$ be the j^{th} Bernstein coefficient of the polynomial in Bernstein form. Consider the following algorithm, modified from (Goyal and Sigman 2012)[92].

1. Flip the input coin n times, and let j be the number of times the coin returned 1 this way.
2. If 0 is in the domain of f and if j is 0, return $f(0)$. (By condition 3, $f(0)$ must be either 0 or 1.)
3. If 1 is in the domain of f and if j is n, return $f(1)$. (By condition 4, $f(1)$ must be either 0 or 1.)
4. With probability $a[j]$, return 1. Otherwise, return 0. (For example, generate a uniformly distributed random variate, greater than 0 and less than 1, then return 1 if that variate is less than $a[j]$, or 0 otherwise. $a[j]$ is the Bernstein coefficient j of the polynomial written in Bernstein form), or 0 otherwise.

(By the properties of the Bernstein form, $a[0]$ will equal $f(0)$ and $a[n]$ will equal $f(1)$ whenever 0 or 1 is in the domain of f, respectively.)

Step 4 is done by first generating unbiased bits (such as with the von Neumann trick of flipping the input coin twice until the flip returns 0 then 1 or 1 then 0 this way, then taking the result as 0 or 1, respectively (von Neumann 1951)[93]), then using the algorithm in "**Digit Expansions**[94]" to produce the probability $a[j]$. The algorithm computes $a[j]$ bit by bit and compares the computed value with the generated bits. Since the coin returned both 0 and 1 in step 1 earlier in the algorithm, we know the coin isn't degenerate, so that step 4 will finish with probability 1. Now, since the Bernoulli factory used only the input coin for randomness, this shows that f is strongly simulable. [ ]

Proposition 2. *If $f(\lambda)$ is described in the strong simulability statement, and if either f is constant on its domain or f meets the additional conditions below, then f is strongly simulable.*

1. *If $f(0) = 0$ or $f(1) = 0$ or both, then there is a polynomial $g(\lambda)$ whose Bernstein coefficients are computable and in the closed unit interval, 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.*
2. *If $f(0) = 1$ or $f(1) = 1$ or both, then there is a polynomial $h(\lambda)$ whose Bernstein coefficients are computable and in the closed unit interval, 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.*

Lemma 1. *If $f(\lambda)$ is described in the strong simulability statement and meets the additional condition below, then f is strongly simulable.*

- *There is a polynomial $g(\lambda)$ whose Bernstein coefficients are computable and in the closed unit interval, 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.*

Proof: Consider the following algorithm.

1. If f is 0 everywhere in its domain or 1 everywhere in its domain, return 0 or 1, respectively.

[92] Goyal, V. and Sigman, K., 2012. On simulating a class of Bernstein polynomials. ACM Transactions on Modeling and Computer Simulation (TOMACS), 22(2), pp.1-5.

[93] von Neumann, J., "Various techniques used in connection with random digits", 1951.

[94] https://peteroupc.github.io/bernoulli.html#Digit_Expansion

2. Otherwise, use the algorithm given in Proposition 1 to simulate $g(\lambda)$. If the algorithm returns 0, return 0. By the additional condition in the lemma, 0 will be returned if λ is either 0 or 1.

   Now, we know that the input coin's probability of heads is neither 0 nor 1.

   By the conditions in the lemma, both $f(\lambda)>0$ and $g(\lambda)>0$ whenever $0 < \lambda < 1$ and λ is in f's domain.

   Now let $h(\lambda) = f(\lambda)/g(\lambda)$. By the conditions in the lemma, h will be positive everywhere in that interval.

3. Return 1 if h has the following property: $h(\lambda) = 0$ whenever $0 < \lambda < 1$ and λ is in f's domain.

4. Otherwise, we run a Bernoulli factory algorithm for $h(\lambda)$ that uses the input coin (and possibly outside randomness). Since h is continuous and polynomially bounded and the input coin's probability of heads is neither 0 nor 1, h is strongly simulable; we can replace the outside randomness in the algorithm with unbiased random bits via the von Neumann trick.

Thus, f admits an algorithm that uses nothing but the input coin as a source of randomness, and so is strongly simulable. [ ]

Lemma 2. *If $f(\lambda)$ is described in the strong simulability statement and meets the additional conditions below, then f is strongly simulable.*

1. *There are two polynomials $g(\lambda)$ and $\omega\ (\lambda)$ such that both polynomials' Bernstein coefficients are computable and all in the closed unit interval.*
2. *$g(0) = \omega\ (0) = f(0) = 0$ (so that 0 is in the domain of f).*
3. *$g(1) = \omega\ (1) = f(1) = 1$ (so that 1 is in the domain of f).*
4. *For every λ in the domain of f, except at 0 and 1, $g(\lambda) > f(\lambda)$.*
5. *For every λ in the domain of f, except at 0 and 1, $\omega\ (\lambda) < f(\lambda)$.*

Proof: First, assume g and ω have the same degree. If not, rewrite the the polynomial with lesser degree to a polynomial in Bernstein form with the same degree as the other polynomial.

Now, let $g[j]$ and $\omega\ [j]$ be the j^{th} Bernstein coefficient of the polynomial g or ω , respectively. Consider the following algorithm, which is similar to the algorithm in Proposition 1.

1. Flip the input coin n times, and let j be the number of times the coin returned 1 this way.
2. If 0 is in the domain of f and if j is 0, return $g(0) = \omega\ (0) = 0$.
3. If 1 is in the domain of f and if j is n, return $g(1) = \omega\ (1) = 0$.
4. Generate a uniformly distributed random variate, greater than 0 and less than 1, then return 1 if that variate is less than $\omega\ [j]$, or return 0 if that variate is greater than $g[j]$. This step is carried out via the von Neumann method, as in Proposition 1.

If the algorithm didn't return a value, then by now we know that the input coin's probability of heads is neither 0 nor 1, since step 2 returned a value (either 0 or 1), which can only happen if the input coin didn't return all zeros or all ones.

Now let $r(\lambda) = (f(\lambda) - \omega\ (\lambda))\ /\ (g(\lambda) - \omega\ (\lambda))$. By the conditions in the lemma, $h(\lambda)$ will be positive wherever $0 < \lambda < 1$ and λ is in the domain of f.

Now, run a Bernoulli factory algorithm for $r(\lambda)$ that uses the input coin (and possibly outside randomness). Since r is continuous and polynomially bounded and the input coin's probability of heads is neither 0 nor 1, r is strongly simulable; we can replace the outside randomness in the algorithm with unbiased random bits via the von Neumann trick.

Thus, f admits an algorithm that uses nothing but the input coin as a source of randomness, and so is strongly simulable. [ ]

Proof of Proposition 2: The following cases can occur:

1. If neither 0 nor 1 are in the domain of f, then f is strongly simulable by the discussion above.
2. If f is 0 everywhere in its domain or 1 everywhere in its domain: Return 0 or 1, respectively.
3. If 0 but not 1 is in the domain of f: If $f(0) = 0$, apply Lemma 1. If $f(0) = 1$, apply Lemma 1, but take $f = 1 - f$ and return 1 minus the output of the lemma's algorithm (this will bring $f(0) = 0$ and satisfy the lemma.)
4. If 1 but not 0 is in the domain of f: If $f(1) = 0$, apply Lemma 1. If $f(1) = 1$, apply Lemma 1, but take $f = 1 - f$ and return 1 minus the output of the lemma's algorithm (this will bring $f(1) = 0$ and satisfy the lemma.)
5. $f(0) = f(1) = 0$: Apply Lemma 1.
6. $f(0) = f(1) = 1$: Apply Lemma 1, but take $f = 1 - f$ and return 1 minus the output of the lemma's algorithm.
7. $f(0) = 0$ and $f(1) = 1$: Apply Lemma 2.
8. $f(0) = 1$ and $f(1) = 0$: Apply Lemma 2, but take $f = 1 - f$ and return 1 minus the output of the lemma's algorithm.

[ ]

Proposition 3. *If f(λ) is described in the strong simulability statement and is Lipschitz continuous, then f is strongly simulable.*

Lemma 3. *If f(λ) is described in the strong simulability statement, is Lipschitz continuous, and is such that f(0) = 0 and f(1) = 0 whenever 0 or 1, respectively, is in the domain of f, then f is strongly simulable.*

Proof: If f is 0 everywhere in its domain or 1 everywhere in its domain: Return 0 or 1, respectively. Otherwise, let—

- M be the Lipschitz constant of f (its derivative's maximum of the absolute value if f is continuous), or a computable number greater than this.
- l be either 0 if 0 is in the domain of f, or 1 otherwise, and
- u be either 0 if 1 is in the domain of f, or 1 otherwise.

To build g, take its degree as ceil(M)+1 or greater (so that g's Lipschitz constant is greater than M and g has ceil(M) + 2 Bernstein coefficients), then set the first Bernstein coefficient as l, the last one as u, and the remaining ones as 1. (As a result, the polynomial g will have computable Bernstein coefficients.) Then g will meet the additional condition for Lemma 1 and the result follows from that lemma. [ ]

Lemma 4. *If f(λ) is described in the strong simulability statement, is Lipschitz continuous, and is such that f(0) = 0 and f(1) = 1 (so that 0 and 1 are in the domain of f), then f is strongly simulable.*

Proof: Let M and l be as in Lemma 3.

To build g and ω , take their degree as ceil(M)+1 or greater (so that their Lipschitz constant is greater than M and each polynomial has ceil(M) + 2 Bernstein coefficients), then for each polynomial, set its first Bernstein coefficient as l and its last as 1. The remaining Bernstein coefficients of g are set as 1 and the remaining ones of ω are set as 0. (As a result, the polynomial g will have computable Bernstein coefficients.) Then g and ω will meet the additional conditions for Lemma 2 and the result follows from that lemma. [ ]

Proof of Proposition 3: In the proof of proposition 2, replace Lemma 1 and Lemma 2 with Lemma 3 and Lemma 4, respectively. [ ]

Conjecture 1. *The conditions of Proposition 2 are necessary for $f(\lambda)$ to be strongly simulable.*

A condition such as "0 is not in the domain of f, or there is a number $\epsilon > 0$ and a Lipschitz continuous function $g(x)$ on the interval $0 \leq x < \epsilon$ such that $g(x) = f(x)$ whenever x is in both g's and f's domains" does not work. An example that shows this is $f(x) = (\sin(1/x)/4 + 1/2) \cdot (1 - (1 - x)^n)$ for $n \geq 1$ (and $f(0) = 0$), which is strongly simulable at 0 even though the condition just quoted is not satisfied for f. ($(1 - x)^n$ is the probability of the biased coin showing zero n times in a row.)

8.6 Multiple-Output Bernoulli Factory

A related topic is a Bernoulli factory that takes a coin with unknown probability of heads λ and produces one or more samples of the probability $f(\lambda)$. This section 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 input coin flips than producing one output multiple times.

Let a and b be real numbers satisfying $0 < a < b < 1$, such as a=1/100, b=99/100. 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. The question is:

When the probability λ is such that $a \leq \lambda \leq b$, is there a multiple-output Bernoulli factory for $f(\lambda)$ with an expected ("long-run average") number of input coin flips per sample that is arbitrarily close to the entropy bound? Call such a Bernoulli factory an ***optimal factory***.

(See Nacu and Peres (2005, Question 2)[95].)

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

- The functions λ and $1 - \lambda$.
- Constants c satisfying $0 \leq c \leq 1$. As Nacu and Peres (2005)[96] already showed, any such constant admits an optimal factory: generate unbiased random bits using Peres's iterated von Neumann extractor (Peres 1992)[97], then build a binary tree that generates 1 with probability c and 0 otherwise (Knuth and Yao 1976)[98].

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)[99], 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, doesn't count as an *optimal factory*, and Peres described this algorithm only incompletely. By simulation and trial and error I found an improved version of the algorithm. It uses two randomness extractors (extractor 1 and extractor 2) that produce unbiased random bits from biased data (which is done using a method given later in this section). The extractors must be asymptotically optimal (they must approach the entropy limit as closely as desired); one example is the iterated von Neumann construction in Peres (1992)[100]. The algorithm consists of doing the following in a loop until the desired number of outputs is generated.

1. If the number of outputs generated so far is divisible by 20, do the following:

[95] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[96] Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115. https://projecteuclid.org/euclid.aoap/1106922322

[97] Peres, Y., "**Iterating von Neumann's procedure for extracting random bits**", Annals of Statistics 1992,20,1, p. 590-597. https://projecteuclid.org/euclid.aos/1176348543

[98] 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.

[99] Mendo, Luis. "An asymptotically optimal Bernoulli factory for certain functions that can be expressed as power series." Stochastic Processes and their Applications 129, no. 11 (2019): 4366-4384.

[100] Peres, Y., "**Iterating von Neumann's procedure for extracting random bits**", Annals of Statistics 1992,20,1, p. 590-597. https://projecteuclid.org/euclid.aos/1176348543

- Generate an unbiased random bit (see below). If that bit is zero, output 0, then repeat this step unless the desired number of outputs has been generated. If the bit is 1, flip the input coin and output the result.

2. Otherwise, do the following:
    1. Generate an unbiased random bit (see below), call it *fc*. Then flip the input coin and call the result *bc*.
    2. Output *fc***bc*.
    3. (The following steps pass "unused" randomness to the extractor in a specific way to ensure correctness.) If *fc* is 0, and *bc* is 1, append 0 to extractor 2's input bits.
    4. If *fc* and *bc* are both 0, append 1 then 1 to extractor 2's input bits.
    5. If *fc* is 1 and *bc* is 0, append 1 then 0 to extractor 2's input bits.

Inspired by Peres's result with λ /2, the following algorithm is proposed. It works for every function writable as $D(\lambda)/E(\lambda)$, where—

- D is the polynomial $D(\lambda) = \sum_{i=0}^{k} \lambda^i(1-\lambda)^{k-i}d[i]$,
- E is the polynomial $E(\lambda) = \sum_{i=0}^{k} \lambda^i(1-\lambda)^{k-i}e[i]$,
- every $d[i]$ is less than or equal to the corresponding $e[i]$, and
- each $d[i]$ and each $e[i]$ is a nonnegative integer.

The algorithm is a modified version of the "block simulation" in Mossel and Peres (2005, Proposition 2.5)[101], which also "extracts" residual randomness with the help of six asymptotically optimal randomness extractors. In the algorithm, let r be an integer such that, for every integer i in [0, k], $e[i] <$ choose(k, i)*choose(2*r, r).

1. Set *iter* to 0.
2. Flip the input coin k times. Then build a bitstring *B1* consisting of the coin flip results in the order they occurred. Let i be the number of ones in *B1*.
3. Generate 2*r unbiased random bits (see below). (Rather than flipping the input coin 2*r times, as in the algorithm of Proposition 2.5.) Then build a bitstring *B2* consisting of the coin flip results in the order they occurred.
4. If the number of ones in *B2* is other than r: Translate *B1* + *B2* to an integer under mapping 1, then pass that number to extractor 2 (†), then add 1 to *iter*, then go to step 2.
5. Translate *B1* + *B2* to an integer under mapping 2, call the integer β . If $\beta < d[i]$, pass β to extractor 3, then pass *iter* to extractor 6, then output a 1. Otherwise, if $\beta < e[i]$, pass $\beta - d[i]$ to extractor 4, then pass *iter* to extractor 6, then output a 0. Otherwise, pass $\beta - e[i]$ to extractor 5, then add 1 to *iter*, then go to step 2.

The mappings used in this algorithm are as follows:

1. A one-to-one mapping between—
    - bitstrings of length k + 2*r with fewer or greater than r ones among the last 2*r bits, and
    - the integers in [0, 2^k * ($2^{\ 2}$ − choose(2*r, r))).
2. A one-to-one mapping between—
    - bitstrings of length k + 2*r with exactly i ones among the first k bits and exactly r ones among the remaining bits, and
    - the integers in [0, choose(k, i)*choose(2*r, r)).

In this algorithm, an unbiased random bit is generated as follows. Let m be an even integer that is 32 or greater (in general, the greater m is, the more efficient the overall algorithm is in terms of coin flips).

1. Use extractor 1 to extract outputs from floor(n/m)*m inputs, where n is the number of input bits available to that extractor. Do the same for the remaining extractors.

[101] Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

2. If extractor 2 has at least one unused output bit, take an output and stop. Otherwise, repeat this step for the remaining extractors.
3. Flip the input coin at least m times, append the coin results to extractor 1's inputs, and go to step 1.

Now consider the last paragraph of Proposition 2.5. If the input coin were flipped in step 2, the probability of—

- outputting 1 in the algorithm's last step would be $P1 = \lambda^r * (1-\lambda)^r * D(\lambda)$, and
- outputting either 0 or 1 in that step would be $P01 = \lambda^r * (1-\lambda)^r * E(\lambda)$,

so that the algorithm would simulate $f(\lambda) = P1 \,/\, P01$. Observe that the $\lambda^r * (1-\lambda)^r$ cancels out in the division. Thus, we could replace the input coin with unbiased random bits and still simulate $f(\lambda)$; the $\lambda^r * (1-\lambda)^r$ above would then be $(1/2)^2$.

While this algorithm is coin-flip-efficient, it is not believed to be an optimal factory, at least not without more work. In particular, a bigger savings of input coin flips could occur if $f(\lambda)$ maps each value a or greater and b or less to a small range of values, so that the algorithm could, for example, generate a uniform random variate between 0 and 1 using unbiased random bits and see whether that variate lies outside that range of values — and thus produce a sample from $f(\lambda)$ without flipping the input coin again.

(†) For example, by translating the number to input bits via Pae's entropy-preserving binarization (Pae 2018)[102]. But correctness might depend on how this is done; after all, the number of coin flips per sample must equal or exceed the entropy bound for every λ.

8.7 Pushdown Automata and Algebraic Functions

This section has mathematical proofs showing which kinds of algebraic functions (functions that can be a solution of a nonzero polynomial equation) can be simulated with a pushdown automaton (a state machine with a stack).

The following summarizes what can be established about these algebraic functions:

- sqrt(λ) can be simulated.
- Every rational function with rational Bernstein coefficients that maps the open interval (0, 1) to itself can be simulated.
- If $f(\lambda)$ can be simulated, so can any Bernstein-form polynomial in the variable $f(\lambda)$ with Bernstein coefficients that can be simulated.
- If $f(\lambda)$ and $g(\lambda)$ can be simulated, so can $f(\lambda)*g(\lambda)$, $f(g(\lambda))$, and $g(f(\lambda))$.
- If a full-domain pushdown automaton (defined later) can generate words of a given length with a given probability (a *probability distribution* of word lengths), then the probability generating function for that distribution can be simulated, as well as for that distribution conditioned on a finite set or periodic infinite set of word lengths (for example, all odd word lengths only).
- If a stochastic context-free grammar (defined later) can generate a probability distribution of word lengths, and terminates with probability 1, then the probability generating function for that distribution can be simulated.
- Every quadratic irrational number between 0 and 1 can be simulated.

It is not yet known whether the following functions can be simulated:

- $\lambda^{1/}$ for prime numbers p greater than 2. The answer may be no; Banderier and Drmota (2015)[103] proved results that show, among other things, that $\lambda^{1/p}$, where p is not a power of 2, is not a possible solution to $P(\lambda) = 0$, where $P(\lambda)$ is a polynomial whose "power" coefficients are non-negative real numbers.

[102] S. Pae, "**Binarization Trees and Random Number Generation**", arXiv:1602.06058v2 [cs.DS], 2018. https://arxiv.org/abs/1602.06058v2

[103] Banderier, C. And Drmota, M., 2015. Formulae and asymptotics for coefficients of algebraic functions. Combinatorics, Probability and Computing, 24(1), pp.1-53.

- $\min(\lambda\ ,\ 1 - \lambda)$.

The following definitions are used in this section:

1. 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 coin that shows heads with an unknown probability. It starts at 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. (Because each left-hand side has no more than one possible transition, the automaton is *deterministic*.)

2. A *full-domain pushdown automaton* means a pushdown automaton that terminates with probability 1 given a coin with probability of heads λ , for every λ greater than 0 and less than 1.

3. **PDA** is the class of functions $f(\lambda)$ that satisfy $0<f(\lambda)<1$ whenever $0< \lambda <1$, and can be simulated by a full-domain pushdown automaton. **PDA** also includes the constant functions 0 and 1.

4. **ALGRAT** is the class of functions that satisfy $0<f(\lambda)<1$ whenever $0< \lambda <1$, are continuous, and are algebraic over the rational numbers (they satisfy a nonzero polynomial system whose "power" coefficients are rational numbers; specifically, there is a nonzero polynomial $P(x,\ y)$ in two variables and whose "power" coefficients are rational numbers, such that $P(x, f(x)) = 0$ for every x in the domain of f). **ALGRAT** also includes the constant functions 0 and 1.

5. A *probability generating function* has the form $p_0{}^*\ \lambda\ ^0\ +\ p_1{}^*\ \lambda\ ^1\ +$ ..., where p_i is the probability of getting i.

   Notes:

   1. Mossel and Peres (2005)[104] defined pushdown automata to start with a non-empty stack of *arbitrary* size, and to allow each rule to replace the top symbol with an *arbitrary* number of symbols. Both cases can be reduced to the definition in this section.
   2. Pushdown automata, as defined here, are very similar to so-called *probabilistic right-linear indexed grammars* (Icard 2020)[105] and can be translated to those grammars as well as to *probabilistic pushdown systems* (Etessami and Yannakakis 2009)[106], as long as those grammars and systems use only transition probabilities that are rational numbers.

Proposition 0 (Mossel and Peres 2005[107], Theorem 1.2): *A full-domain pushdown automaton can simulate a function that maps (0, 1) to itself only if the function is in class* ***ALGRAT****.*

[104] Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

[105] Icard, Thomas F., "Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy", *Journal of Mathematical Psychology* 95 (2020): 102308.

[106] Icard, Thomas F., "Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy", *Journal of Mathematical Psychology* 95 (2020): 102308.

[107] Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

It is not known whether **ALGRAT** and **PDA** are equal, but the following can be established about **PDA**:

Lemma 1A: *Let g(λ) be a function in the class **PDA**, and suppose a pushdown automaton F has two rules of the form (`state`, HEADS, `stacksymbol`) → RHS1 and (`state`, TAILS, `stacksymbol`) → RHS2, where `state` and `stacksymbol` are a specific state/symbol pair among the left-hand sides of F's rules. Then there is a pushdown automaton that transitions to RHS1 with probability g(λ) and to RHS2 with probability 1 − g(λ) instead.*

Proof: If RHS1 and RHS2 are the same, then the conclusion holds and nothing has to be done. Thus assume RHS1 and RHS2 are different.

Let G be the full-domain pushdown automaton for g. Assume that machines F and G stop when they pop EMPTY from the stack. If this is not the case, transform both machines by renaming the symbol EMPTY to EMPTY ′ ′ , adding a new symbol EMPTY ′ ′ and new starting state X0, and adding rules (X0, *flip*, EMPTY) → (*start*, {EMPTY ′ ′ }) and rule (*state*, *flip*, EMPTY) → (*state*, {}) for all states other than X0, where *start* is the starting state of F or G, as the case may be.

Now, rename each state of G as necessary so that the sets of states of F and of G are disjoint (mutually exclusive). Then, add to F a new stack symbol EMPTY ′ (or a name not found in the stack symbols of G, as the case may be). Then, for the following two pairs of rules in F, namely—

(*state*, HEADS, *stacksymbol*) → (*state2heads*, *stackheads*), and (*state*, TAILS, *stacksymbol*) → (*state2tails*, *stacktails*),

add two new states $state_0$ and $state_1$ that correspond to *state* and have names different from all other states, and replace that rule with the following rules:

(*state*, HEADS, *stacksymbol*) → (*gstart*, {*stacksymbol*, EMPTY ′ }), (*state*, TAILS, *stacksymbol*) → (*gstart*, {*stacksymbol*, EMPTY ′ }), ($state_0$, HEADS, *stacksymbol*) → (*state2heads*, *stackheads*), ($state_0$, TAILS, *stacksymbol*) → (*state2heads*, *stackheads*), ($state_1$, HEADS, *stacksymbol*) → (*state2tails*, *stacktails*), and ($state_1$, TAILS, *stacksymbol*) → (*state2tails*, *stacktails*),

where *gstart* is the starting state for G, and copy the rules of the automaton for G onto F, but with the following modifications:

- Replace the symbol EMPTY in G with EMPTY ′ .
- Replace each state in G with a name distinct from all other states in F.
- Replace each rule in G of the form (*state*, *flip*, EMPTY ′) → (*state2*, {}), where *state2* is a final state of G associated with output 1, with the rule (*state*, *flip*, EMPTY ′) → ($state_1$, {}).
- Replace each rule in G of the form (*state*, *flip*, EMPTY ′) → (*state2*, {}), where *state2* is a final state of G associated with output 0, with the rule (*state*, *flip*, EMPTY ′) → ($state_0$, {}).

Then, the final states of the new machine are the same as those for the original machine F. [ ]

Lemma 1B: *There are pushdown automata that simulate the probabilities 0 and 1.*

Proof: The probability 0 automaton has the rules (START, HEADS, EMPTY) → (START, {}) and (START, TAILS, EMPTY) → (START, {}), and its only state START is associated with output 0. The probability 1 automaton is the same, except START is associated with output 1. Both automata obviously terminate with probability 1. [ ]

Because of Lemma 1A, it's possible to label each left-hand side of a pushdown automaton's rules with not just HEADS or TAILS, but also a rational number or another function in **PDA**, as long as for each state/symbol pair, the probabilities for that pair sum to 1. For example, rules like the following are now allowed:

(START, 1/2, EMPTY) → …, (START, sqrt(λ)/2, EMPTY) → …, (START, (1 − sqrt(λ))/2, EMPTY) → ….

Proposition 1A: *If f(λ) is in the class **PDA**, then so is every polynomial written as—*

$$\binom{n}{0}f(\lambda)^0(1-f(\lambda))^{n-0}a[0] + \binom{n}{1}f(\lambda)^1(1-f(\lambda))^{n-1}a[1] + ... + \binom{n}{n}f(\lambda)^n(1-f(\lambda))^{n-n}a[n],$$

*where n is the polynomial's degree and a[0], a[1], …, a[n] are functions in the class **PDA**.*

Proof Sketch: This corresponds to a two-stage pushdown automaton that follows the algorithm of Goyal and Sigman (2012)[108]: The first stage counts the number of "heads" shown when flipping the f(λ) coin, and the second stage flips another coin that has success probability $a[i]$, where i is the number of "heads". The automaton's transitions take advantage of Lemma 1A. [ ]

Proposition 1: *If f(λ) and g(λ) are functions in the class **PDA**, then so is their product, namely f(λ)*g(λ).*

Proof: Special case of Proposition 1A with n=1, $f(\lambda)$=$f(\lambda)$, $a[0]$=0 (using Lemma 1B), and $a[1]$=$g(\lambda)$. [ ]

Corollary 1A: *If f(λ), g(λ), and h(λ) are functions in the class **PDA**, then so is f(λ)*g(λ) + (1 − f(λ))*h(λ).*

Proof: Special case of Proposition 1A with n=1, $f(\lambda)$=$f(\lambda)$, $a[0]$=$h(\lambda)$, and $a[1]$=$g(\lambda)$. [ ]

Proposition 2: *If f(λ) and g(λ) are functions in the class **PDA**, then so is their composition, namely f(g(λ)) or (f g)(λ).*

Proof: Let F be the full-domain pushdown automaton for f. For each state/symbol pair among the left-hand sides of F's rules, apply Lemma 1A to the automaton F, using the function g. Then the new machine F terminates with probability 1 because the original F and the original automaton for g do for every λ greater than 0 and less than 1, and because the automaton for g never outputs the same value with probability 0 or 1 for any λ greater than 0 or less than 1. Moreover, f is in class **PDA** by Theorem 1.2 of (Mossel and Peres 2005)[109] because the machine is a full-domain pushdown automaton. [ ]

Proposition 3: *Every rational function with rational Bernstein coefficients that maps the open interval (0, 1) to itself is in class **PDA**.*

Proof: These functions can be simulated by a finite-state machine (Mossel and Peres 2005)[110]. This corresponds to a full-domain pushdown automaton that has no stack symbols other than EMPTY, never pushes symbols onto the stack, and pops the only symbol EMPTY from the stack whenever it transitions to a final state of the finite-state machine. [ ]

> **Note:** An unbounded stack size is necessary for a pushdown automaton to simulate functions that a finite-state machine can't. With a bounded stack size, there is a finite-state machine where each state not only holds the pushdown automaton's original state, but also encodes the contents of the stack (which is possible because the stack's size is bounded); each operation that would push, pop, or change the top symbol transitions to a state with the appropriate encoding of the stack instead.

Proposition 4: *If a full-domain pushdown automaton can generate words with the same letter such that the length of each word follows a probability distribution, then that distribution's probability generating function is in class **PDA**.*

Proof: Let F be a full-domain pushdown automaton. Add one state FAILURE, then augment F with a special "letter-generating" operation as follows. Add the following rule that pops all symbols from the stack:

(FAILURE, *flip*, *stacksymbol*) → (FAILURE, {}),

[108]Goyal, V. and Sigman, K., 2012. On simulating a class of Bernstein polynomials. ACM Transactions on Modeling and Computer Simulation (TOMACS), 22(2), pp.1-5.

[109]Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

[110]Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

and for each rule of the following form that transitions to a letter-generating operation (where S and T are arbitrary states):

(S, *flip*, *stacksymbol*) → (T, *newstack*),

add another state S ′ (with a name that differs from all other states) and replace that rule with the following rules:

(S, *flip*, *stacksymbol*) → (S ′ , {*stacksymbol*}), (S ′ , HEADS, *stacksymbol*) → (T, *newstack*), and (S ′ , TAILS, *stacksymbol*) → (FAILURE, {}).

Then if the stack is empty upon reaching the FAILURE state, the result is 0, and if the stack is empty upon reaching any other state, the result is 1. By Dughmi et al. (2021)[111], the machine now simulates the distribution's probability generating function. Moreover, the function is in class **PDA** by Theorem 1.2 of Mossel and Peres (2005)[112] because the machine is a full-domain pushdown automaton. [ ]

Define a *stochastic context-free grammar* as follows. The grammar consists of a finite set of *nonterminals* and a finite set of *letters*, and rewrites one nonterminal (the starting nonterminal) into a word. The grammar has three kinds of rules (in generalized Chomsky normal form (Etessami and Yannakakis 2009)[113]):

- $X \to a$ (rewrite X to the letter a).
- $X \to_p (a,\ Y)$ (with rational probability p, rewrite X to the letter a followed by the nonterminal Y). For the same left-hand side, all the p values must sum to 1.
- $X \to (Y,\ Z)$ (rewrite X to the nonterminals Y and Z in that order).

Instead of a letter (such as a), a rule can use ε (the empty string). (The grammar is *context-free* because the left-hand side has only a single nonterminal, so that no context from the word is needed to parse it.)

Proposition 5: *Every stochastic context-free grammar can be transformed into a pushdown automaton. If the automaton is a full-domain pushdown automaton and the grammar has a one-letter alphabet, the automaton can generate words such that the length of each word follows the same distribution as the grammar, and that distribution's probability generating function is in class* ***PDA****.*

Proof Sketch: In the equivalent pushdown automaton:

- $X \to a$ becomes the two rules—(START, HEADS, X) → (*letter*, {}), and(START, TAILS, X) → (*letter*, {}).Here, *letter* is either START or a unique state in F that "detours" to a letter-generating operation for a and sets the state back to START when finished (see Proposition 4). If a is ε , *letter* is START and no letter-generating operation is done.
- $X \to p_i\ (a_i,\ Y_i)$ (all rules with the same nonterminal X) are rewritten to enough rules to transition to a letter-generating operation for a_i, and swap the top stack symbol with Y_i, with probability p_i, which is possible with just a finite-state machine (see Proposition 4) because all the probabilities are rational numbers (Mossel and Peres 2005)[114]. If a_i is ε , no letter-generating operation is done.
- $X \to (Y,\ Z)$ becomes the two rules—(START, HEADS, X) → (START, {Z, Y}), and(START, TAILS, X) → (START, {Z, Y}).

Here, X is the stack symbol EMPTY if X is the grammar's starting nonterminal. Now, assuming the automaton is full-domain, the rest of the result follows easily. For a single-letter alphabet, the grammar corresponds to a system of polynomial equations, one for each rule in the grammar, as follows:

- $X \to a$ becomes $X = 1$ if a is the empty string (ε), or $X = \lambda$ otherwise.

[111]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.

[112]Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

[113]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.

[114]Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

- For each nonterminal X, all n rules of the form $X \to p_i\ (a_i,\ Y_i)$ become the equation $X = p_1 * \lambda\ _1 * Y_1 + p_2 * \lambda\ _2 * Y_2 + \ldots + p_n * \lambda\ _n * Y_n$, where $\lambda\ _i$ is either 1 if a_i is ε , or λ otherwise.
- $X \to (Y, Z)$ becomes $X = Y * Z$.

Solving this system for the grammar's starting nonterminal, and applying Proposition 4, leads to the *probability generating function* for the grammar's word distribution. (See also Flajolet et al. 2010[115], Icard 2020[116].) [ ]

Example: The stochastic context-free grammar—$X \to 1/2\ (a,\ X1), X1 \to (X,\ X2), X2 \to (X,\ X), X \to 1/2\ (a,\ X3), X3 \to \varepsilon$,which encodes ternary trees (Flajolet et al. 2010)[117], corresponds to the equation $X = (1/2) * \lambda * X * X * X + (1/2) * \lambda * 1$, and solving this equation for X leads to the probability generating function for such trees, which is a complicated expression.

Notes:

1. A stochastic context-free grammar in which all the probabilities are 1/2 is called a *binary stochastic grammar* (Flajolet et al. 2010)[118]. If every probability is a multiple of $1/n$, then the grammar can be called an "n-ary stochastic grammar". It is even possible for a nonterminal to have two rules of probability λ and $(1 - \lambda)$, which are used when the input coin returns 1 (HEADS) or 0 (TAILS), respectively.

2. If a pushdown automaton simulates the function $f(\lambda)$, then f corresponds to a special system of equations, built as follows (Mossel and Peres 2005)[119]; see also Esparza et al. (2004)[120]. For each state of the automaton (call the state *en*), include the following equations in the system based on the automaton's transition rules:

   - $(st,\ p,\ sy) \to (s2,\ \{\})$ becomes either $\alpha\ st,sy,en = p$ if *s2* is *en*, or $\alpha\ st,sy,en = 0$ otherwise.
   - $(st,\ p,\ sy) \to (s2,\ \{sy1\})$ becomes $\alpha\ st,sy,en = p * \alpha\ s2,sy1,en$.
   - $(st,\ p,\ sy) \to (s2,\ \{sy1,\ sy2\})$ becomes $\alpha\ st,sy,en = p * \alpha\ s2,sy2,\_\ \sigma\ [1] * \alpha\ \ \sigma\ [1]$,sy1,en $+ \ldots + \mathrm{p} * \alpha$ s2,sy2, $\sigma\ [\mathrm{n}] * \alpha\ \ \sigma\ [\mathrm{n}]$,sy1,en, *where* $\sigma\ [\mathrm{i}]\_$ is one of the machine's n states.

   (Here, p is the probability of using the given transition rule; the special value HEADS becomes λ , and the special value TAILS becomes $1 - \lambda$.) Now, each time multiple equations have the same left-hand side, combine them into one equation with the same left-hand side, but with the sum of their right-hand sides. Then, for every variable of the form $\alpha\ a,b,c$ not yet present in the system, include the equation $\alpha\ a,b,c = 0$. Then, for each final state *fs* that returns 1, solve the system for the variable α START,EMPTY,*fs* (where START is the automaton's starting state) to get a solution (a function) that maps the open interval (0, 1) to itself. (Each solve can produce multiple solutions, but only one of them will map that open interval to itself assuming every p is either HEADS or TAILS.) Finally, add all the solutions to get $f(\lambda)$.

3. Assume there is a pushdown automaton (F) that follows Definition 1 except it uses a set of N input letters (and not simply HEADS or TAILS), accepts an input word if the stack is empty, and rejects the word if the machine reaches a configuration without a transition rule. Then a pushdown automaton in the full sense of Definition 1 (G) can be built. In essence:

[115]Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010 https://arxiv.org/abs/0906.5560

[116]Levy, H., *Stochastic dominance*, 1998.

[117]Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010 https://arxiv.org/abs/0906.5560

[118]Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010 https://arxiv.org/abs/0906.5560

[119]Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

[120]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.

1. Add a new FAILURE state, which when reached, pops all symbols from the stack.
2. For each pair (*state*, *stacksymbol*) for F, add a set of rules that generate one of the input letters (each letter i generated with probability $f_i(\lambda)$, which must be a function in **PDA**), then use the generated letter to perform the transition stated in the corresponding rule for F. If there is no such transition, transition to the FAILURE state instead.
3. When the stack is empty, output 0 if G is in the FAILURE state, or 1 otherwise.

Then G returns 1 with the same probability as F accepts an input word with letters randomly generated as in the second step. Also, one of the N letters can be a so-called "end-of-string" symbol, so that a pushdown automaton can be built that accepts "empty strings"; an example is Elder et al. (2015)[121].

Proposition 6: *If a full-domain pushdown automaton can generate a distribution of words with the same letter, there is a full-domain pushdown automaton that can generate a distribution of such words conditioned on—*

1. *a finite set of word lengths, or*
2. *a periodic infinite set of word lengths.*

One example of a finite set of word lengths is {1, 3, 5, 6}, where only words of length 1, 3, 5, or 6 are allowed. A *periodic infinite set* is defined by a finite set of integers such as {1}, as well as an integer modulus such as 2, so that in this example, all integers congruent to 1 modulo 2 (that is, all odd integers) are allowed word lengths and belong to the set.

Proof Sketch:

1. As in Lemma 1A, assume that the automaton stops when it pops EMPTY from the stack. Let S be the finite set (for example, {1, 3, 5, 6}), and let M be the maximum value in the finite set. For each integer i in [0, M], make a copy of the automaton and append the integer i to the name of each of its states. Combine the copies into a new automaton F, and let its start state be the start state for copy 0. Now, whenever F generates a letter, instead of transitioning to the next state after the letter-generating operation (see Proposition 4), transition to the corresponding state for the next copy (for example, if the operation would transition to copy 2's version of "XYZ", namely "2_XYZ", transition to "3_XYZ" instead), or if the last copy is reached, transition to the last copy's FAILURE state. If F would transition to a failure state corresponding to a copy not in S (for example, "0_FAILURE", "2_FAILURE", "3_FAILURE" in this example), first all symbols other than EMPTY are popped from the stack and then F transitions to its start state (this is a so-called "rejection" operation). Now, all the final states (except FAILURE states) for the copies corresponding to the values in S (for example, copies 1, 3, 5, 6 in the example) are treated as returning 1, and all other states are treated as returning 0.

2. Follow (1), except as follows: (A) M is equal to the integer modulus minus 1. (B) For the last copy of the automaton, instead of transitioning to the next state after the letter-generating operation (see Proposition 4), transition to the corresponding state for copy 0 of the automaton. [ ]

Proposition 7: *Every constant function equal to a quadratic irrational number between 0 and 1 is in class **PDA**.*

A *continued fraction* is one way to write a real number. For purposes of the following proof, every real number greater than 0 and less than 1 has the following *continued fraction expansion*: 0 + 1 / (a[1] + 1 / (a[2] + 1 / (a[3] + …))), where each $a[i]$, a *partial denominator*, is an integer greater than 0. A *quadratic irrational number* is an irrational number that can be written as (b+sqrt(c))/d, where b, c, and d are rational numbers.

[121] Elder, Murray, Geoffrey Lee, and Andrew Rechnitzer. "Permutations generated by a depth 2 stack and an infinite stack in series are algebraic." *Electronic Journal of Combinatorics* 22(1), 2015.

Proof: By Lagrange's continued fraction theorem, every quadratic irrational number has a continued fraction expansion that is eventually periodic; the expansion can be described using a finite number of partial denominators, the last "few" of which repeat forever. The following example describes a periodic continued fraction expansion: [0; 1, 2, (5, 4, 3)], which is the same as [0; 1, 2, 5, 4, 3, 5, 4, 3, 5, 4, 3, …]. In this example, the partial denominators are the numbers after the semicolon; the size of the period (`(5, 4, 3)`) is 3; and the size of the non-period (`1, 2`) is 2.

Given a periodic expansion, and with the aid of an algorithm for simulating **continued fractions**[122], a recursive Markov chain for the expansion (Etessami and Yannakakis 2009)[123] can be described as follows. The chain's components are all built on the following template. The template component has one entry E, one inner node N, one box, and two exits X0 and X1. The box has one *call port* as well as two *return ports* B0 and B1.

- From E: Go to N with probability x, or to the box's call port with probability $1 - x$.
- From N: Go to X1 with probability y, or to X0 with probability $1 - y$.
- From B0: Go to E with probability 1.
- From B1: Go to X0 with probability 1.

Let p be the period size, and let n be the non-period size. Now the recursive Markov chain to be built has $n+p$ components:

- For each i in [1, n+1], there is a component labeled i. It is the same as the template component, except $x = a[i]/(1 + a[i])$, and $y = 1/a[i]$. The component's single box goes to the component labeled i+1, *except* that for component $n+p$, the component's single box goes to the component labeled n+1.

According to Etessami and Yannakakis (2009)[124], the recursive Markov chain can be translated to a pushdown automaton of the kind used in this section. Now all that's left is to argue that the recursive Markov chain terminates with probability 1. For every component in the chain, it goes from its entry to its box with probability 1/2 or less (because each partial numerator must be 1 or greater). Thus, the component recurses with no greater probability than not, and there are otherwise no probability-1 loops in each component, so the overall chain terminates with probability 1. [ ]

Lemma 1: *The square root function sqrt(λ) is in class **PDA**.*

Proof: See Mossel and Peres (2005)[125]. [ ]

Corollary 1: *The function $f(\lambda) = \lambda\ m/(2^{\,n}\ )$, where $n \geq 1$ is an integer and where $m \geq 1$ is an integer, is in class **PDA**.*

Proof: Start with the case m=1. If n is 1, write f as sqrt(λ); if n is 2, write f as (sqrt sqrt)(λ); and for general n, write f as (sqrt sqrt … sqrt)(λ), with n instances of sqrt. Because this is a composition and sqrt can be simulated by a full-domain pushdown automaton, so can f.

For general m and n, write f as ((sqrt sqrt … sqrt)(λ))^m, with n instances of sqrt. This involves doing m multiplications of sqrt sqrt … sqrt, and because this is an integer power of a function that can be simulated by a full-domain pushdown automaton, so can f.

Moreover, f is in class **PDA** by Theorem 1.2 of (Mossel and Peres 2005)[126] because the machine is a full-domain pushdown automaton. [ ]

[122] https://peteroupc.github.io/bernoulli.html#Continued_Fractions

[123] 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.

[124] 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.

[125] Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

[126] Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724, 2005.

8.7.1 Finite-State and Pushdown Generators

Another interesting class of machines (called *pushdown generators* here) are similar to pushdown automata (see above), with the following exceptions:

1. Each transition rule can also, optionally, output a base-N digit in its right-hand side. An example is: (*state*, *flip*, *sy*) → (*digit*, *state2*, {*sy2*}).
2. The machine must output infinitely many digits if allowed to run forever.
3. Rules that would pop the last symbol from the stack are not allowed.

The "output" of the machine is now a real number X in the form of the base-N digit expansion `0.dddddd...`, where `dddddd...` are the digits produced by the machine from left to right. In the rest of this section:

- `CDF(z)` is the cumulative distribution function of X, or the probability that X is z or less.
- `PDF(z)` is the probability density function of X, or the derivative of `CDF(z)`, or the relative probability of choosing a number "close" to z at random.

A *finite-state generator* (Knuth and Yao 1976)[127] is the special case 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. Then if `PDF(z)` has infinitely many derivatives on the open interval (0, 1), it must be a polynomial with rational Bernstein coefficients and satisfy `PDF(z) > 0` whenever $0 \leq z \leq 1$ is irrational (Vatan 2001)[128], (Kindler and Romik 2004)[129], and it can be shown that the expected value (mean or "long-run average") of X must be a rational number. [130]

Proposition 8. *Suppose a finite-state generator can generate a probability distribution that takes on finitely many values. Then:*

1. *Each value occurs with a rational probability.*
2. *Each value is either rational or transcendental.*

A real number is *transcendental* if it can't be a root of a nonzero polynomial with integer "power" coefficients. Thus, part 2 means, for example, that irrational, non-transcendental numbers such as 1/sqrt(2) and the golden ratio minus 1 can't be generated exactly.

Proving this proposition involves the following lemma, which shows that a finite-state generator is related to a machine with a one-way read-only input and a one-way write-only output:

Lemma 2. *A finite-state generator can fit the model of a one-way transducer-like k-machine (as defined in Adamczewski et al. (2020)[131] section 5.3), for some k equal to 2 or greater.*

Proof Sketch: There are two cases.

Case 1: If every transition rule of the generator outputs a digit, then k is the number of unique inputs among the generator's transition rules (usually, there are two unique inputs, namely HEADS and TAILS), and the model of a finite-state generator is modified as follows:

1. A *configuration* of the finite-state generator consists of its current state together with either the last coin flip result or, if the coin wasn't flipped yet, the empty string.

[127] 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.

[128] 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.

[129] Kindler, Guy and D. Romik, "On distributions computable by random walks on graphs," *SIAM Journal on Discrete Mathematics* 17 (2004): 624-633.

[130] Vatan (2001) claims that a finite-state generator has a continuous `CDF` (unless it produces a single value with probability 1), but this is not necessarily true if the generator has a state that outputs 0 forever.

[131] 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.

2. The *output function* takes a configuration described above and returns a digit. If the coin wasn't flipped yet, the function returns an arbitrary digit (which is not used in proposition 4.6 of the Adamczewski paper).

Case 2: If at least one transition rule does not output a digit, then the finite-state generator can be transformed to a machine where HEADS/TAILS is replaced with 50% probabilities, then transformed to an equivalent machine whose rules always output one or more digits, as claimed in Lemma 5.2 of Vatan (2001)[132]. In case the resulting generator has rules that output more than one digit, additional states and rules can be added so that the generator's rules output only one digit as desired. Now at this point the generator's probabilities will be rational numbers. Now transform the generator from probabilities to inputs of size k, where k is the product of those probabilities, by adding additional rules as desired. [ ]

Proof of Proposition 8: Let n be an integer greater than 0. Take a finite-state generator that starts at state START and branches to one of n finite-state generators (sub-generators) with some probability, which must be rational because the overall generator is a finite-state machine (Icard 2020, Proposition 13)[133]. The branching process outputs no digit, and part 3 of the proposition follows from Corollary 9 of Icard (2020)[134]. The n sub-generators are special; each of them generates the binary expansion of a single real number in the closed unit interval with probability 1.

To prove part 2 of the proposition, translate an arbitrary finite-state generator to a machine described in Lemma 2. Once that is done, all that must be shown is that there are two different non-empty sequences of coin flips that end up at the same configuration. This is easy using the pigeonhole principle, since the finite-state generator has a finite number of configurations. Thus, by propositions 5.11, 4.6, and AB of Adamczewski et al. (2020)[135], the generator can generate a real number's binary expansion only if that number is rational or transcendental (see also Cobham (1968)[136]; Adamczewski and Bugeaud (2007)[137]). [ ]

Proposition 9. *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 on that interval.*

The proof follows from combining Kindler and Romik (2004, Theorem 2)[138] and Knuth and Yao (1976)[139] with Richman (2012)[140], who proved that a continuous algebraic function on an open interval is piecewise analytic; that is, each piece is analytic at every point except possibly at the endpoints.

9 License

[132]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.

[133]Icard, Thomas F., "Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy", *Journal of Mathematical Psychology* 95 (2020): 102308.

[134]Icard, Thomas F., "Calibrating generative models: The probabilistic Chomsky–Schützenberger hierarchy", *Journal of Mathematical Psychology* 95 (2020): 102308.

[135]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.

[136]Cobham, A., "On the Hartmanis-Stearns problem for a class of tag machines", in *IEEE Conference Record of 1968 Ninth Annual Symposium on Switching and Automata Theory* 1968.

[137]Adamczewski, B., Bugeaud, Y., "On the complexity of algebraic numbers I. Expansions in integer bases", *Annals of Mathematics* 165 (2007).

[138]Kindler, Guy and D. Romik, "On distributions computable by random walks on graphs," *SIAM Journal on Discrete Mathematics* 17 (2004): 624-633.

[139]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.

[140]Richman, F. (2012). Algebraic functions, calculus style. Communications in Algebra, 40(7), 2671-2683.

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