Supplemental Notes for Bernoulli Factory Algorithms

This version of the document is dated 2023-04-08.

Peter Occil

1 Contents

- **Contents**
- **About This Document**
- **Definitions**
- **General Factory Functions**
  - **Building the Lower and Upper Polynomials**
- **Approximate Bernoulli Factories**
  - **Approximate Bernoulli Factories for Certain Functions**
  - **Approximate Bernoulli Factories for Power Series**
  - **Approximate Bernoulli Factories for Other "Smooth" Functions**
  - **Approximate Bernoulli Factories for Linear Functions**
- **Achievable Simulation Rates**
- **Complexity**
- **Notes**
- **Appendix**
  - **Proofs on Cutting Off a Power Series**
  - **Results Used in Approximate Bernoulli Factories**
  - **Failures of the Consistency Requirement**
  - **Which functions admit a Bernoulli factory?**
  - **Which functions don't require outside randomness to simulate?**
  - **Multiple-Output Bernoulli Factory**
  - **Proofs for Polynomial-Building Schemes**
    - **A Conjecture on Polynomial Approximation**
  - **Example of Polynomial-Building Scheme**
  - **Pushdown Automata and Algebraic Functions**
    - **Finite-State and Pushdown Generators**

- **License**

2 About This Document

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

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**".

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**:

- 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 — which 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 $0 \le x \le 1$.

- If $f$ is continuous, its *derivative* is, roughly speaking, its "slope" or "velocity" or "instantaneous-rate-of-change" function. The derivative (or *first derivative*) is denoted as $f'$. The *second derivative* ("slope-of-slope") of $f$, denoted $f''$, is the derivative of $f'$; the *third derivative* is the derivative of $f''$; and so on.
- A ***Hölder continuous*** 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 $0 < \alpha \le 1$.
  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 absolute value of the maximum of that derivative.
- A *convex* function $f$ has the property that $f((x+y)/2) \le (f(x)+f(y))/2$ 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 $f((x+y)/2) \ge (f(x)+f(y))/2$ 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.

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**, 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: The difference—

    - between the degree-($n-1$) upper polynomial and the degree-n upper polynomial, and
    - between the degree-n lower polynomial and the degree-($n-1$) lower polynomial,

    must have nonnegative coefficients, once the polynomials are rewritten in Bernstein form and elevated to degree n.

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. Examples of this fact are shown in the section "**Failures of the Consistency Requirement**" in the appendix.

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

4.1 Building the Lower and Upper Polynomials

A *factory function* $f(\lambda)$ is a function for which the Bernoulli factory problem can be solved (see "**About Bernoulli Factories**"). The following 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 factory function using a computer algebra system to see if it belongs to any of the classes of functions described below.

Concave functions. If f is concave, then **fbelow**(n, k) can equal $f(k/n)$, thanks to Jensen's inequality. One example is $f(\lambda) = 1- \lambda^2$.

Convex functions. If f is convex, then **fabove**(n, k) can equal $f(k/n)$, thanks to Jensen's inequality. One example is $f(\lambda) = \exp(-\lambda/4)$.

Hölder and Lipschitz continuous functions. I have found a way to extend the results of Nacu and Peres (2005)[1] 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*** on the closed unit interval, with Hölder constant m or less and Hölder exponent α (see "**Definitions**" as well as note 2 at the end of this section), 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. **Lipschitz special case**: If the function f has a continuous

> first derivative on its domain, then α is 1 (f is Lipschitz continuous) and m can be the absolute value of the maximum of that derivative.
>
> 2. **Finding Hölder parameters**: The Hölder constant and exponent are found by inspecting $g(\lambda) = \max_c \text{abs}(f(\lambda)-f(c))/((\text{abs}(\lambda-c))^\alpha)$ where the maximum is taken over all points c where f has a "vertical slope" or the "steepest slope exhibited". (Note that Hölder constants are greater than 0 and are usually 1 or less.) 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 equal to or greater than the maximum value of $g(\lambda)$ on its domain.
> 3. If the factory function has a Hölder exponent of 1 (and so is Lipschitz continuous), $D(n)$ can be m*322613/(250000*sqrt(n)), which is an upper bound.
> 4. If the factory function's Hölder exponent is 1/2 or greater, $D(n)$ can be m*154563/(40000*$n^{1/4}$), which is an upper bound.
> 5. Some factory functions are not Hölder continuous for any Hölder exponent greater than 0. These functions have a slope that's steeper than every "nth" root, and can't be handled by this method. One example is $f(\lambda)$ = 1/10 if λ is 0 and −1/(2*ln(λ/2)) + 1/10 otherwise, which has a slope near 0 that's steeper than every "nth" root.
>
> **Examples:**
>
> 1. If $f(\lambda)$ is a *piecewise linear* factory function (made of multiple linear functions defined on a finite number of "pieces", or subintervals, that together make up the closed unit interval), then α is 1 (f is Lipschitz continuous) and m can equal the greatest absolute value of the slope among all pieces' slopes. For example:
>
>   - Suppose $f(\lambda)$ equals 0 at 0, 3/4 at 2/3 and 1/4 at 1, and these points are connected by linear functions. In this example, m is $\max(|(3/4-0)/(2/3)|$, $|(1/4-3/4)/(1/3)|)$ = 1.5. Moreover, f is concave (the first piece's slope is greater than the second piece's).
>   - Suppose $f(\lambda)$ = min(λ*$mult$, 1−ε), with $mult$ > 0 and ε > 0. This has a rising linear piece and a constant piece, and

> equals 0 at 0, $1-\varepsilon$ at $(1-\varepsilon)/mult$, and $1-\varepsilon$ at 1. Functions of this kind were discussed by Thomas and Blanchet (2012)[2] [3] and Nacu & Peres (2005)[4] [5]. In this example, *m* is max(*mult*, 0) = *mult*. Moreover, *f* is concave.
>
> 2. Let $f(\lambda) = 1/2-(1-2\lambda)^{z}/2$ if $\lambda<1/2$ and $1/2+(2\lambda-1)^{z}/2$ otherwise, where $0\lt z\le 1$. This function is Hölder continuous with Hölder exponent $\alpha=z$ and Hölder constant $m=2^z/2$. In this example, $f$ has a "vertical" slope at 1/2, and the maximum value of $g(\lambda)$ (see note 2) is $2^z/2$ when $\alpha=z$.
>
> 3. Let $f(\lambda)=3/4-\sqrt{\lambda(1-\lambda)}$. This function is Hölder continuous with Hölder exponent $\alpha=1/2$ and Hölder constant $m=1$. In this example, $f$ has a "vertical" slope at 0 and 1, and the maximum value of $g(\lambda)$ (see note 2) is 1 when $\alpha=1/2$.
>
> 4. Let $f(\lambda)$ = 3*sin(sqrt(3)*sqrt(sin(2*λ)))/4 + 1/50. This function is Hölder continuous with Hölder exponent $\alpha=1/2$, and its Hölder constant is upper bounded by $m=3\sqrt{6}/4 \lt 1.837118$. In this example, f has a "vertical" slope at 0.

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**"), and
- in 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.

Let *m* be the Lipschitz constant of *f*'s derivative, or a greater number than that constant (if *f* has a second derivative on its domain, then *m* can be the absolute value of the maximum of that second derivative). Then for every integer *n* that's a power of 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, $f(k/n) - m/(7*n)$.
- **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, $f(k/n) + m/(7*n)$.

Examples:

1. Take $f(\lambda)$ = exp($-\lambda$). This is a convex function, and its derivative is Lipschitz continuous with Lipschitz constant 1. Then it can be shown that the following scheme for $f$ is valid (the value 3321/10000 is slightly less than M − 1/(7*4), where M is the minimum of $f$ on its domain):

    - **fbelow**($n$, k) = 3321/10000 if $n$<4; otherwise, $f(k/n)$ − 1/(7*n). (Observe that $f(k/4)$ − 1/(7*4) ≥ 3321/10000.)
    - **fabove**($n$, k) = $f(k/n)$ (because $f$ is convex).

    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.

2. Take $f(\lambda)$ = λ/2 if λ ≤ 1/2; (4*λ − 1)/(8*λ) otherwise. This function is concave, and its derivative is Lipschitz continuous with Lipschitz constant 2. Then it can be shown that the following scheme for $f$ is valid (the value 893/2000 is slightly greater than M + 2/(7*4), where M is the maximum of $f$ on its domain):

    - **fbelow**($n$, k) = $f(k/n)$.
    - **fabove**($n$, k) = 893/2000 if $n$<4; otherwise, $f(k/n)$ + 2/(7*$n$).

3. The following scheme is valid for the functions in the table below:

    - 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, $f(k/n)-\delta(k,n)$. 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, $f(k/n)+\delta(k,n)$. For every other $n$, **fabove**($n$,k) = 1.

Function $f(\lambda)$	**Value of $\delta(k, n)$**	**Not**

cosh(λ) − 3/4.	cosh(1)/(7*n) < 0.220441/n.	Con nam the
$\lambda\cdot\sin(z\lambda)/4+1/2$.	$\frac{z(2+xz)}{4}\frac{1}{7n}$.	Con $\de bour assu $\sir $z>
$\sin(z\lambda)/4+1/2$.	$\frac{z^2}{4}\frac{1}{7n}$.	Con $\de bour nam z^2,
$\lambda^2/2 + 1/10$ if λ ≤ 1/2; λ/2 − 1/40 otherwise.	$1/(7n)$.	Lips with

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 ≥ `High`(1) ≥ *f*(1) ≥ 0.
- 1 > `High`(λ) > *f*(λ) > 0 whenever 0 < λ < 1.
- If *f*(1) = 0, then `High`(1) = 0. (This is required to ensure correctness in case λ is 1.)

Also, `High` should be a function with a simple Bernoulli factory algorithm. For example, `High` can be a polynomial in Bernstein form of degree *n* whose *n* plus one coefficients are [0, 1, 1, ..., 1]. This polynomial is easy to simulate using the algorithms from the section "**Certain Polynomials**".[6]

The algorithm is now described.

Let $g(\lambda) = \lim_{\nu \to \lambda} f(\nu)/\texttt{High}(\nu)$ (roughly speaking, the value that $f(\nu)/\texttt{High}(\nu)$ 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 $\texttt{High}(\lambda) = \lambda$, then this step amounts to the following: "Flip the input coin. If it returns 0, return 0.")
2. Run a Bernoulli factory algorithm for $g(.)$ and return the result of that algorithm. This can be one of the **general factory function algorithms** 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. See the second example below.
> 2. $\texttt{High}(\lambda)$ can also equal 1 instead of be described in this section. That leads to the original Bernoulli factory algorithm for $f(\lambda)$.
>
> **Examples:**
>
> 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 in Bernstein form

> has coefficients [0, 6/10, 6/10]), so $G(\lambda)$ = 5/12 if λ = 0, and −(5*cosh(λ) − 5)/(3*λ^2*(λ−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:

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

Roughly speaking, `Low` is a function that bounds f from below, just as `High` bounds f from above. `Low` should be a function with a simple Bernoulli factory algorithm, such as a polynomial in Bernstein form. 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**").

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 `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 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** 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 $f=\text{High}\cdot(q\cdot1+(1-q)\cdot(1-r))+(1-\text{High})\cdot0$.
> 2. `High`(λ) is allowed to equal 1 if the *r*(.) in step 3 is allowed to equal 0 at 0.
>
> **Example:** If *f*(λ) = (1−exp(λ))/(1−exp(1)), then *f* is less than or equal to `High`(λ) = λ, and greater than or equal to `Low`(λ) = λ^2. As a result, *q*(λ) = λ, and *r*(λ) = (2 − exp(1))/(1 − exp(1)) if λ = 0; 1/(exp(1)−1) if λ = 1; and (−λ*(1 − exp(1)) − exp(λ) + 1)/(λ*(1 − exp(1))*(λ − 1)) otherwise. This can be computed using the following code 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. If *f* does not fully admit a polynomial-building scheme under the convex, concave, Lipschitz derivative, and Hölder classes:

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*(λ) = 1 − *f*(1−λ). *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*(λ) = *f*(1−λ). (For example, cosh(λ)−1 becomes cosh(1−λ)−1.) Inverted coin.
1	0	Algorithm for **certain functions that equal 0 at 0 and 1 at 1**, but with *f*(λ) = 1−*f*(λ).

		Inverted result.
1	> 0 and ≤ 1	Algorithm for **certain functions that equal 0 at 0**, but with $f(\lambda) = 1-f(\lambda)$. Inverted result.

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 to the desired function $f(\lambda)$.

   The simplest choice for this polynomial, known simply as a *Bernstein polynomial*, has n+1 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.

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

2. Then, use one of the algorithms in the section "**Certain Polynomials**" to toss heads with probability equal to that polynomial, given its coefficients.

> **Note:** 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.

5.1 Approximate Bernoulli Factories for Certain Functions

This section first discusses approximating f with a *Bernstein polynomial* (a degree-n polynomial in Bernstein form with coefficients $f(k/n)$ with $0\le k\le n$). The advantage is only one Bernstein coefficient has to be found per run; the disadvantage is that Bernstein polynomials approach f slowly in general, in the order of magnitude of $1/n$ (Voronovskaya 1932)[7].

There are results that give an upper bound on the error on approximating *f* with a degree-*n* Bernstein polynomial. To find a degree *n* such that *f* is approximated with a maximum error of *ε*, solve the error bound's equation for *n*, then take *n* = ceil(*n*) to get the solution if it's an integer, or the nearest integer that's bigger than the solution.

For example:

If $f(\lambda)$:	Then the degree-*n* Bernstein polynomial is close to f with the following error bound:	Where *n* is:
Has Lipschitz continuous derivative (see "Definitions").	$\epsilon = M/(8*n)$.	$n = \text{ceil}(M/(8*\epsilon))$.

Is Hölder continuous.	ε = M*(1/(4*n))$^{\alpha/2}$.	n = ceil(1/(4$^{\alpha}$*ε^2/M^2)$^{1/\alpha}$).
Is Lipschitz continuous.	ε = L*sqrt(1/(4*n)).	n = ceil(L^2/(4*ε^2)).
Is Lipschitz continuous.	ε = $\frac{4306+837\sqrt{6}}{5832} L/n^{1/2}$ < $1.08989 L/n^{1/2}$.	n=ceil((L*1.08989/ε)^2).

Now, if *f* belongs to any of the classes given above, the following algorithm (adapted from "Certain Polynomials") simulates a polynomial that approximates *f* with a maximum error of ε:

1. Calculate *n* as described in the table above for the given class.
2. Flip the input coin *n* times, and let *j* be the number of times the coin returned 1 this way.
3. With probability *f*(*j*/*n*), return 1. Otherwise, return 0. (If *f*(*j*/*n*) can be an irrational number, see "**Algorithms for General Irrational Constants**" for ways to sample this irrational probability exactly.)

Alternatively, polynomials other than Bernstein polynomials, but written in Bernstein form, can be used to approximate f with an error no more than ϵ, as long as an explicit upper bound on the approximation error is available. A ratio of two such polynomials can also approximate f this way. See my **question on MathOverflow**.

Examples of these alternative polynomials (all of degree n) are given in the following table. In the table, $B_n(f(\lambda))$ is the ordinary Bernstein polynomial for $f(\lambda)$.

Name	Polynomial	Its Bernstein coefficients are found as follows:	Notes
Order-2 iterated Bernstein polynomial.	$U_{n,2} = B_n(W_{n,2})$.	Calculate $W_{n,2} = 2 f(\lambda) - B_n(f(\lambda))$.	Micchelli (1973)[11], Guan (2009)[12], (Güntürk and Li 2021, sec. 3.3)[13].
Order-3 iterated Bernstein polynomial.	$U_{n,3} = B_n(W_{n,3})$.	Calculate $W_{n,3} = B_n(B_n(f(\lambda)))$ + $3 (f(\lambda)$ − $B_n(f(\lambda)))$.	Same.
Linear Bernstein polynomial combination.	$L_{2,n/2} = 2 B_{n}(f(\lambda))$ − $B_{n/2}(f(\lambda))$.	Treat the coefficients [$f(0/(n/2))$, $f(1/(n/2))$, ..., $f((n/2)/(n/2))$] as representing a polynomial in Bernstein form of degree $n/2$, elevate that polynomial to one of degree n with Bernstein coefficients $b_0, b_1, ..., b_n$, then set the final Bernstein coefficients to $b_i = 2 f(i/n) - b_i$ for each i.	(Tachev 2022)[14]. $n\ge 6$ must be even.

The goal is now to find a degree n such that—

1. the alternative polynomial is within ϵ of $f(\lambda)$, and
2. the polynomial $W_{n,i}$ is not less than 0 or greater than 1.

Then:

If *f*(*λ*):	Then the following polynomial:	Is close to *f* with the following error bound:	Where
Is concave and has continuous second derivative.	$U_{n,2}$.	ε = 0.75*M/n^2.	$n$ = ceil(sqrt(0.7
Has continuous third derivative.	$L_{2, n/2}$.	ε = (3*sqrt(3−4/$n$)/4)*M/n^2 < (3*sqrt(3)/4)*M/n^2.	$n$=max(6,ceil($\f \sqrt{M/\epsilon max(6,ceil((1139 * sqrt(M/ε))). (If odd, add 1.)

By analyzing the proof of Theorem 2.4 of Güntürk and Li (2021, sec. 3.3)[16], the following error bounds *appear* to be true. In the table below, M_n is not less than the maximum of the absolute value of $f(\lambda)$ and its derivatives up the $n$-th derivative.

If *f*(*λ*):	Then the following polynomial:	Appears to be close to *f* with the following error bound:	Where *n* is:
Has continuous		ε =	

continuous fourth derivative.	$U_{n,2}$	ε = 0.3571*M_4/n^2.	$n$=ceil(sqrt(0.3571)*sqr
Has continuous fifth derivative.	$U_{n,3}$	ε = 4.0421*$M_5/n^{5/2}$.	$n$=ceil((4.0421)$^{2/5}$*(M_5/ε
Has continuous sixth derivative.	$U_{n,3}$	ε = 4.8457*M_6/n^3.	$n$=ceil((4.8457)$^{1/3}$*(M_6/ε

However, unlike with ordinary Bernstein polynomials, the alternative polynomial is not necessarily bounded by 0 and 1. The following process can be used to calculate the required degree $n$, given an error tolerance of $\epsilon$.

1. Determine whether $f$ is described in either of the two tables above. Let *A* be the minimum of $f$ on the closed unit interval and let *B* be the maximum of $f$ there.
2. If 0 < *A* ≤ *B* < 1, calculate $n$ as given in either of the two tables above, but with $\epsilon=\min(\epsilon, A, 1-B)$, and stop.
3. Propositions B1, B2, and B3 in the **appendix** give conditions on $f$ so that $W_{n,2}$ or $W_{n,3}$ (as the case may be) will be nonnegative. If *B* is less than 1 and any of those conditions is met, calculate $n$ as given in either of the two tables above, but with $\epsilon=\min(\epsilon, 1-B)$. (For B3, set $n$ to max($n$, $m$), where $m$ is given in that proposition.) Then stop; $W$ will now be bounded by 0 and 1.
4. Calculate $n$ as given in either of the two tables above. Then, if $P(j/n)\lt 0$ or $P(j/n)\gt 1$ for some $0\le j\le n$, where $P$ is the alternative polynomial, double the value of $n$ until this condition is no longer true.

Once *n* is found, simulating the alternative polynomial is as follows:

1. Flip the input coin *n* times, and let *j* be the number of times the coin returned 1 this way.
2. With probability equal to the alternative polynomial's Bernstein coefficient at position *j* (0≤*j*≤*n*), return 1. Otherwise, return 0.

> **Notes:**

> 1. Providing the full proof for the error bounds shown in the last table is a bit tedious, so here is a sketch. The proof was found by analyzing Theorem 2.2 of Güntürk and Li (2021)[17], finding upper bounds for so-called "central moments" of the binomial distribution (see B4 to B7 in the appendix), then plugging them in to various estimates mentioned in that theorem's proof.
> 2. A polynomial's Bernstein coefficients can be rounded to multiples of δ (where $0 \lt\delta\le 1$) by setting c=floor(c/δ) * δ for each coefficient c. The new polynomial will differ from the old one by at most δ. (Thus, to find a polynomial with multiple-of-δ coefficients that approximates f with error ϵ [which must be greater than δ], first find a polynomial with error $\epsilon - \delta$, then round that polynomial's coefficients as given here.)

5.2 Approximate Bernoulli Factories for Power Series

Some functions can be rewritten as a power series, namely: $$f(\lambda) = a_0 \lambda^0 + a_1 \lambda^1 + ... + a_i \lambda^i + ...,$$ where a_i, the *coefficients*, are constant rational numbers[18].

To simulate an approximation of f that comes within ϵ of f:

1. Find the first n+1 coefficients such that the polynomial $P(\lambda) = a_0 \lambda^0 + ... + a_n\lambda^n$ is within ϵ of f wherever $0 \le \lambda \le 1$.

   If f's coefficients are each greater than 0, form a nowhere increasing sequence (example: (1/4, 1/8, 1/8, 1/16, ...)), and meet the so-called "ratio test", the algorithms in Carvalho and Moreira (2022)[19] can be used here (see also "**Proofs on Cutting Off a Power Series**" in the appendix).

   Alternatively, if bounds on the derivatives of f are known, then thanks to Taylor's theorem, $P(\lambda)$ will be close enough if $M/((n+1)!) \le \epsilon$, where M is equal to or greater than the absolute value of the maximum of f's (n+1)-th derivative on the domain of f.

2. Rewrite $P(\lambda)$ as a polynomial in Bernstein form. (One way to transform a polynomial to Bernstein form, given the "power" coefficients $a_0, ..., a_n$, is the so-called "matrix method" from Ray and Nataraj (2012)[20].) Let $b_0, ..., b_n$ be the Bernstein-form polynomial's coefficients. If any of those coefficients is less than 0 or greater than 1, double the value of n and rewrite P to Bernstein form of degree n, until none of the coefficents is less than 0 or greater than 1.
3. Flip the input coin *n* times, then let *j* be the number of times the coin returned 1 this way, then return either 1 with probability b_j, or 0 otherwise.

In fact, if $f(\lambda)$ belongs in *Gevrey's hierarchy* (there are $B\ge 1, l\ge 1, \gamma\ge 1$ such that its n-th derivative's absolute value is not greater than $Bl^n n^{\gamma n}$ for every n), which includes functions equaling power series as a special case ($\gamma=1$), it's possible to bound the derivatives and find the appropriate degree for the approximating polynomial (for details, see (Kawamura et al. 2015)[21]; see also (Gevrey 1918)[22]).

5.3 Approximate Bernoulli Factories for Other "Smooth" Functions

If $f(\lambda)$ is "smooth" enough and ϵ is big enough, then Taylor's theorem shows how to build a polynomial that comes within ϵ of f.

Let $n\ge 0$ be an integer, and let $f^{(i)}$ be the i-th derivative of $f(\lambda)$. Suppose that—

- f is continuous on the closed unit interval, and
- f satisfies $\epsilon\le f(0)\le 1-\epsilon$ and $\epsilon\le f(1)\le 1-\epsilon$, and
- f satisfies $\epsilon\lt f(\lambda)\lt 1-\epsilon$ whenever $0\lt\lambda\lt 1$, and
- f's $(n+1)$-th derivative is continuous and satisfies $\epsilon\ge M/((n+1)!)$, where M is not less than the maximum of the absolute value of that derivative, and

- $f(0)$ is known as well as $f^{(1)}(0), ..., f^{(n)}(0)$.

Then the $n$-th *Taylor polynomial* centered at 0, given below, is within $\epsilon$ of $f$: $$P(\lambda) = a_0 \lambda^0 + a_1 \lambda^1 + ... + a_n \lambda^n,$$

where $a_0 = f(0)$ and $a_i = f^{(i)}(0)/(i!)$ for $i\ge 1$. Given this $P$, the algorithm to simulate this polynomial is the same as steps 2 and 3 of the previous section's algorithm.

5.4 Approximate Bernoulli Factories for Linear Functions

There are a number of approximate methods to simulate $\lambda*c$, where $c > 1$ and $0 \le \lambda < 1/c$. ("Approximate" because this function touches 1 at $1/c$, so it can't be a factory function.) Since the methods use only up to $n$ flips, where $n$ is an integer greater than 0, the approximation will be a polynomial of degree $n$.

- Henderson and Glynn (2003, Remark 4)[23] approximates the function $\lambda*2$ using a polynomial where the j^{th} coefficient (starting at 0) is $\min((j/n)*2, 1-1/n)$. If $g(\lambda)$ is that polynomial, then the error in approximating $f$ is no greater than $1-g(1/2)$. 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))`.

- I found the following approximation for $\lambda*c$[24]: "(1.) Set j to 0 and i to 0; (2.) If $i \ge n$, return 0; (3.) Flip the input coin, and if it returns 1, add 1 to j; (4.) (Estimate the probability and return 1 if it 'went over'.) If $(j/(i+1)) \ge 1/c$, return 1; (5.) Add 1 to i and go to step 2." Here, $\lambda*c$ is approximated by a polynomial where the j^{th} coefficient (starting at 0) is $\min((j/n)*c, 1)$. If $g(\lambda)$ is that polynomial, then the error in approximating $f$ is no greater than $1-g(1/c)$.

- The previous approximation generalizes the one given in section 6 of Nacu and Peres (2005)[25], which approximates $\lambda*2$.

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.
- $\Delta(n, r, \lambda) = O(\max(\text{sqrt}(\lambda*(1-\lambda)/n),1/n)^r)$, that is, $O((1/n)^r)$ near λ = 0 or 1, and $O((1/n)^{r/2})$ elsewhere. ($O(h(n))$ roughly means "less than or equal to $h(n)$ times a constant when n is large enough".)

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 in Bernstein form of degree n with coefficients in the closed unit interval (Goyal and Sigman 2012)[26].
Requires a finite number of flips on average. Also known as "realizable" by Flajolet et al. (2010)[27].	Only if f is Lipschitz continuous (Nacu and Peres 2005)[28]. Whenever f admits a fast simulation (Mendo 2019)[29].
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)[30].
Requires more than n flips with probability $\Delta(n, r + 1, \lambda)$, for integer $r \geq 0$ and every λ. (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 3) (Holtz et al. 2011)[31].
Requires more than n flips with probability $\Delta(n, \alpha, \lambda)$, for non-	If and only if f has an r-th derivative that is Hölder continuous with Hölder exponent

integer $\alpha > 0$ and every λ. (The greater α is, the faster the simulation.)	$(\alpha - r)$ or greater, where r = floor(α) (Holtz et al. 2011)[32]. 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)[33].	If and only if f is real analytic (writable as $f(\lambda)=a_0 \lambda^0 + a_1 \lambda^1 + ...$ for real constants $a_i$) (Nacu and Peres 2005)[34].
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)[35].

> **Notes:**
>
> 1. By the results of Holtz et al., it is suspected that the target function f can't be simulated using a finite number of flips on average for every probability of heads unless f's fourth derivative is Hölder continuous.
> 2. A function $f(\lambda)$ in the *Zygmund class* if there is a constant $D>0$ with the following property for each step size $\epsilon>0$: abs($g(x-h) + g(x+h) - 2g(x)$) $\le D\times\epsilon$ wherever the left-hand side is defined and $0\lt h\le\epsilon$. A function that's Lipschitz continuous is in the Zygmund class, but not necessarily vice versa.

7 Complexity

The following note shows the complexity of the algorithm for $1/\varphi$ in the main article, where φ is the golden ratio.

Let $\mathbf{E}[N]$ be the expected ("long-run average") number of unbiased random bits (fair coin flips) generated by the algorithm.

Then, since each bit is independent, $\mathbf{E}[N] = 2*\varphi$ as shown below.

- Each iteration stops the algorithm with probability p = (1/2) +

$(1-(1/2)) * (1/\varphi)$ (1/2 for the initial bit and $1/\varphi$ for the recursive run; $(1-(1/2))$ because the (1/2) is being subtracted earlier on the right-hand side from 1).

- Thus, the expected number of iterations is $\mathbf{E}[T] = 1/p$ by a well-known rejection sampling argument, since the algorithm doesn't depend on iteration counts.
- Each iteration uses $1 * (1/2) + (1 + \mathbf{E}[N]) * (1/2)$ bits on average, so the whole algorithm uses $\mathbf{E}[N] = (1 * (1/2) + (1 + \mathbf{E}[N]) * (1/2)) * \mathbf{E}[T]$ bits on average (each iteration consumes either 1 bit with probability 1/2, or $(1 + \mathbf{E}[N])$ bits with probability 1/2). This equation has the solution $\mathbf{E}[N] = 1 + \text{sqrt}(5) = 2*\varphi$.

Also, on average, half of these flips (φ) show 1 and half show 0, since the bits are unbiased (the coin is fair).

A similar analysis to the one above can be used to find the expected ("long-run average") time complexity of many Bernoulli factory algorithms.

8 Notes

9 Appendix

9.1 Proofs on Cutting Off a Power Series

Lemma A1: Let— $$f(x)=a_0 x^0 + a_1 x^1 + ...,$$ where the $a_i$ are constants each 0 or greater and sum to a finite value and where $0\le x\le 1$ (the domain is the closed unit interval). Then $f$ is convex and has a maximum at 1.

Proof: By inspection, $f(x)$ is a power series and is nonnegative wherever $x\ge 0$ (and thus wherever $0\le x\le 1$). Each of its terms has a maximum at 1 since—

- for $n=0$, $a_0 x^0=a_0$ is a non-negative constant (which trivially reaches its maximum at 1), and
- for each $n$ where $a_0 = 0$, $a_0 x^n$ is the constant 0 (which

trivially reaches its maximum at 1), and

- for each other n, x^n is a strictly increasing function and multiplying that by a_n (a positive constant) doesn't change whether it's strictly increasing.

Since all of these terms have a maximum at 1 on the domain, so does their sum.

The derivative of f is— $$f'(x) = a_1 x^0 + ... + a_i x^{i-1} + ...,$$ which is still a power series with nonnegative values of a_n, so the proof so far applies to f' instead of f. By induction, the proof so far applies to all derivatives of f, including its second derivative.

Now, since the second derivative is nonnegative wherever $x\ge 0$, and thus on its domain, f is convex, which completes the proof. □

Proposition A2: For a function $f(x)$ as in Lemma A1, let— $$g_n(x)=a_0 x^0 + ... + a_n x^n,$$ and have the same domain as f. Then for every $n\ge 1$, $g_n(x)$ is within ϵ of $f(x)$, where $\epsilon = f(1) - g_n(1)$.

Proof: g_n, consisting of the first $n+1$ terms of f, is a power series with nonnegative coefficients, so by Lemma A1, it has a maximum at 1. The same is true for $f-g_n$, consisting of the remaining terms of f. Since the latter has a maximum at 1, the maximum error is $\epsilon = f(1)-g_n(1)$. □

For a function f described in Lemma A1, $f(1)=a_0 1^0 + a_1 1^1 + ... = a_0 + a_1+...$, and f's error behavior is described at the point 1, the algorithms given in Carvalho and Moreira (2022)[36] — which apply to infinite sums — can be used to "cut off" f at a certain number of terms and do so with a controlled error.

9.2 Results Used in Approximate Bernoulli Factories

Proposition B1: Let $f(\lambda)$ map the closed unit interval to itself and be continuous and concave. Then $W_{n,2}$ and $W_{n,3}$ (as defined in "Approximate Bernoulli Factories for Certain Functions") are nonnegative on the closed unit interval.

Proof: For $W_{n,2}$ it's enough to prove that $B_n(f)\le f$ for every $n\ge 1$. This is the case because of Jensen's inequality and because f is concave.

For $W_{n,3}$ it must also be shown that $B_n(B_n(f(\lambda)))$ is nonnegative. For this, using only the fact that f maps the closed unit interval to itself, $B_n(f)$ will have Bernstein coefficients in that interval (each coefficient is a value of f) and so will likewise map the closed unit interval to itself (Qian et al. 2011)[37]. Thus, by induction, $B_n(B_n(f(\lambda)))$ is nonnegative. The discussion for $W_{n,2}$ also shows that $(f - B_n(f))$ is nonnegative as well. Thus, $W_{n,3}$ is nonnegative on the closed unit interval. □

Proposition B2: Let $f(\lambda)$ map the closed unit interval to itself, be continuous, nowhere decreasing, and subadditive, and equal 0 at 0. Then $W_{n,2}$ is nonnegative on the closed unit interval.

Proof: The assumptions on f imply that $B_n(f)\le 2 f$ (Li 2000)[38], showing that $W_{n,2}$ is nonnegative on the closed unit interval. □

> **Note:** A subadditive function f has the property that $f(a+b) \le f(a)+f(b)$ whenever a, b, and $a+b$ are in f's domain.

Proposition B3: Let $f(\lambda)$ map the closed unit interval to itself and have a Lipschitz continuous derivative with Lipschitz constant L. If $f(\lambda) \ge \frac{L \lambda(1-\lambda)}{2m}$ on f's domain, for some $m\ge 1$, then $W_{n,2}$ is nonnegative there, for every $n\ge m$.

Proof: Let $E(\lambda, n) = \frac{L \lambda(1-\lambda)}{2n}$. Lorentz (1966)[39] showed that with this Lipschitz derivative assumption on f, B_n differs from $f(\lambda)$ by no more than $E(\lambda, n)$ for every $n\ge 1$. By inspection, $E(\lambda, n)$ is biggest when $n=1$ and decreases as n increases. Assuming the worst case that $B_n(\lambda) = f(\lambda) + E(\lambda, m)$, it follows that $W_{n,2}=2 f(\lambda) - B_n(\lambda)\ge 2 f(\lambda) - f(\lambda) - E(\lambda, m) = f(\lambda) - E(\lambda, m)\ge 0$ whenever $f(\lambda)\ge E(\lambda, m)$. Because $E(\lambda, k+1)\le E(\lambda,k)$ for every $k\ge 1$, the preceding sentence holds true for every $n\ge m$. □

The following results deal with a useful quantity when discussing the error in approximating a function by Bernstein polynomials. Suppose a coin shows heads with probability p, and n independent tosses of the coin are made. Then the total number of heads X follows a *binomial distribution*, and the r-th central moment of that distribution is as follows: $$T(n, r, p) = \mathbb{E}[(X-\mathbb{E}[X])^r] = \sum_{k=0}^n (k-np)^r{n \choose k}p^k (1-p)^{n-k},$$ where $\mathbb{E}[.]$ is the expected value ("long-run average"). The following results bound the absolute value of T.[40]

Result B4 (Molteni 2022)[41]: If r is an even integer such that $0\le r\le 44$, then $|T(n, r, p)| \le \frac{r!}{((r/2)!)8^{r/2}} n^{r/2}$ for every $n\ge 1$.

Proposition B5: For every integer $n\ge 1$, the following is true: $$|T(n, 3, p)| \le \frac{\sqrt{3}}{18\sqrt{n}} n^{3/2} \le \frac{\sqrt{3}}{18} n^{3/2} \lt (963/10000) n^{3/2}.$$

Proof: The critical points of $T(n, 3, p)$ (the points where the maximum might be) are at $p=0$, $p=1$, $p=1/2-\sqrt{3}/6$, and $p=1/2+\sqrt{3}/6$. The moment equals 0 at the points 0 and 1, so that leaves the last two. Since $T(n, r, p)$ is antisymmetric whenever r is odd, and is nonnegative whenever r is odd and $0\le p \le 1/2$ (Skorski 2020)[42], it's enough to take the critical point $0 \le p=1/2-\sqrt{3}/6 \le 1/2$ to bound $|T(n, 3, p)|$ on either side. By inspection, the moment at that critical point is decreasing as n increases, starting with $n=1$. □

Corollary B6: For every integer $n_0\ge 1$, $|T(n, 3, p)| \le \frac{\sqrt{3}}{18\sqrt{n_0}} n^{3/2} < (963/10000)\frac{1}{\sqrt{n_0}} n^{3/2}$ whenever $n\ge n_0$.

Proposition B7: For every integer $n\ge 1$, $|T(n, 5, p)| \le 0.083 n^{5/2}.$ For every integer $n\ge 304$, $|T(n, 5, p)| \le n^2 \le 0.05736 n^{5/2}.$

Proof: Evaluating the moment for each $1\le n \le 303$ at its critical point shows that $|T(n,5,p)| < 0.083 n^{5/2}$ for every such n. An upper bound given in sec. 3.1 of Skorski (2020) leads to $|T(n,5,p)| \le n/4+2 {n \choose 2} = n/4+2\frac{n!}{(n-2)!} = n^2 - \frac{3}{4}n \le n^2$ whenever $n\ge 2$, and $n^2/n^{5/2}$ is decreasing as n increases, starting with $n=2$, because its derivative $\frac{-n}{2n^{5/2}}$ is negative whenever $n\ge 2$. Thus it's enough to take the bound n^2 at 304, namely 92188, so that $|T(n,5,p)|\le 304^2 = 92188 < 0.05736/n^{5/2}$ for every $n\ge 304$. This is still less than $0.083 n^{5/2}$, so that bound stands for the first part. □

Proposition B8: Let $n\geq 1$ be an integer. Suppose $f(\lambda)$ is has a continuous second derivative on the closed unit interval. Suppose further that—

- (R1) f is concave, or
- (R2) $f(0)=0$ and f is nowhere decreasing and subadditive.

Then $U_{n,2}(f)$ (as defined in "Approximate Bernoulli Factories for Continuous Functions") is within $3 M\lambda(1-\lambda)/(n^2)$ and within $0.75 M/(n^2)$ of f, where M is the absolute value of the maximum of f's second derivative.

Proof: Theorem 11 of Bustamante (2008)[43] provides error bounds for certain polynomials that come close to a continuous function. To use this result here, the following conditions are verified:

- $U_{n,2}$ is non-negative (because f satisfies R1 or R2, making $W_{n,2}$, and therefore its Bernstein polynomial $U_{n,2}$, non-negative by Propositions B1 and B2).
- $U_{n,2}$ is a linear operator (Micchelli 1973)[44]. Notably, it equals f whenever f is a linear function.
- $U_{n,2}(\lambda) = \lambda$ and $U_{n,2}(1) = 1$.
- $\phi(\lambda)=1$ which is positive and concave.
- $(U_{n,2}(\lambda^2)-\lambda^2) = \lambda(1-\lambda)/n^2$.

Then by Theorem 11, $U_{n,2}$ is close to f by at most— $$\left(\frac32+\frac{3}{(2h^2\phi(\lambda)^2)}\frac{\lambda(1-\lambda)}{n^2}\right) \omega_{2}^{\phi}(f, h),$$ where $\omega_{2}^{\phi}(f, h)$ is a function[45] that satisfies $\omega_{2}^{\phi}(f, h)\le M h^2$ whenever $h\le 1/2$, for the given choice of ϕ (Timan 1994)[46]. So— $$\left(\frac32+\frac{3}{2h^2\phi(\lambda)^2}\frac{\lambda(1-\lambda)}{n^2}\right) \omega_{2}^{\phi}(f, h) \le \left(\frac32+\frac{3\lambda(1-\lambda)}{2h^2n^2}\right) M h^2 = \epsilon(f, h, n).$$ Finally, the change of variable $h=\sqrt{\lambda(1-\lambda)}/n$ is made (and h is at most 1/2 given $n\ge 1$). This change of variable leads to— $$\epsilon(f,h,n)=\left(\frac32+\frac{3}{2}\right) M (\sqrt{\lambda(1-\lambda)}/n)^2 = \frac{3M\lambda(1-\lambda)}{n^2} \le \frac{3M}{4n^2} = 0.75M/(n^2).$$ □

> **Note:** Requirements R1 and R2 are needed here to ensure $U_{n,2}(f)$ is non-negative. It is open whether this proposition's conclusion remains true if these requirements are

> dropped.

9.3 Failures of the Consistency Requirement

In the academic literature (papers and books), there are many results showing that a polynomial comes within a given error bound of a function $f(\lambda)$, when f meets certain conditions. Unfortunately, these error bounds don't necessarily mean that a sequence of polynomials far from these bounds will obey the consistency requirement, a requirement for simulating f in the Bernoulli factory setting.

Here is one such error bound. Let f have a Lipschitz continuous derivative on the closed unit interval with Lipschitz constant M. Then the *Bernstein polynomial* for f of degree n (which is in Bernstein form with coefficients $f(k/n)$ with $0\le k\le n$) is within $M*x*(1-x)/(2*n)$ of f (and thus within $M/(8*n)$ of f) whenever $0\le x\le 1$ (Lorentz 1966)[47]. Thus, for every $n\geq 1$:

- **fabove**(n, k) = $f(k/n)$ + M / (8*n).
- **fbelow**(n, k) = $f(k/n)$ − M / (8*n).

Where k is an integer and $0 \le k \le n$.

The example against the consistency requirement involves the function $g(\lambda)$ = sin(π*λ)/4 + 1/2, which has a Lipschitz continuous derivative.

For g, the coefficients for—

- the degree-2 upper polynomial in Bernstein form (**fabove**(5, k)) are [0.6542..., 0.9042..., 0.6542...], and
- the degree-4 upper polynomial in Bernstein form (**fabove**(6, k)) are [0.5771..., 0.7538..., 0.8271..., 0.7538..., 0.5771...].

The degree-2 polynomial lies above the degree-4 polynomial everywhere in the closed unit interval. However, to ensure consistency, the degree-2 polynomial, once elevated to degree 4 and rewritten in Bernstein form, must have coefficients that are greater than or equal to those of the degree-4 polynomial.

- Once elevated to degree 4, the degree-2 polynomial's coefficients are [0.6542..., 0.7792..., 0.8208..., 0.7792..., 0.6542...].

As can be seen, the elevated polynomial's coefficient 0.8208... is less than the corresponding coefficient 0.8271... for the degree-4 polynomial.

Note on "clamping". In addition, for a polynomial-building scheme, "clamping" the values of **fbelow** and **fabove** to fit the closed unit interval won't necessarily preserve the consistency requirement, even if the original scheme met that requirement. Here is an example that applies to any scheme.

Let g and h be two polynomials in Bernstein form as follows:

- g has degree 5 and coefficients [10179/10000, 2653/2500, 9387/10000, 5049/5000, 499/500, 9339/10000].
- h has degree 6 and coefficients [10083/10000, 593/625, 9633/10000, 4513/5000, 4947/5000, 9473/10000, 4519/5000].

After elevating g's degree, g's coefficients are no less than h's, as required by the consistency property.

However, by clamping coefficients above 1 to equal 1, so that g is now g' with [1, 1, 9387/10000, 1, 499/500, 9339/10000] and h is now h' with [1, 593/625, 9633/10000, 4513/5000, 4947/5000, 9473/10000, 4519/5000], and elevate g' for coefficients [1, 1, 14387/15000, 19387/20000, 1499/1500, 59239/60000, 9339/10000], some of the coefficients of g' are less than those of h'. Thus, for this pair of polynomials, clamping the coefficients will destroy the consistency property.

9.4 Which functions admit a Bernoulli factory?

Let $f(\lambda)$ be a function whose domain is the closed unit interval or a subset of it, and that maps its domain to the closed unit interval. The domain of f gives the allowable values of λ, which is the input coin's probability of heads.

f admits a Bernoulli factory if and only if f is constant on its domain, or is continuous and *polynomially bounded* on its domain, as defined later in the section "Proofs for Polynomial-Building Schemes" (Keane and O'Brien 1994)[48].

If $f(\lambda)$ meets these sufficient conditions, it admits a Bernoulli factory and is Hölder continuous (see "**Definitions**"):

- $f(\lambda)$ maps the closed unit interval to itself.
- $f(\lambda)$ is continuous.
- $0 < f(\lambda) < 1$ whenever $0 < \lambda < 1$.
- $f(\lambda)$ is algebraic over rational numbers (that is, there is a nonzero polynomial $P(x, y)$ in two variables and whose coefficients are rational numbers, such that $P(x, f(x)) = 0$ for every x in the domain of f).

A **proof by Reid Barton** begins by showing that f is a *semialgebraic function*, so that by a known inequality and the other conditions, it meets the definitions of being Hölder continuous and polynomially bounded.

9.5 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)[49]. See "**Randomized vs. Non-Randomized Algorithms**".

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 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(λ) is described in the strong simulability statement and is a polynomial with computable 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 coefficient of the polynomial in Bernstein form. Consider the following algorithm, modified from (Goyal and Sigman 2012)[50].

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 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)[51]), then using the algorithm in "**Digit Expansions**" 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(λ) 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(λ) in Bernstein form whose 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(λ) > f(λ) 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(λ) in Bernstein form whose 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(λ) < f(λ) for every λ in the domain of f, except at 0 and 1.*

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

- *There is a polynomial g(λ) in Bernstein form whose 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(λ) > f(λ) 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.

2. Otherwise, use the algorithm given in Proposition 1 to simulate *g*(*λ*). 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*(*λ*)>0 and *g*(*λ*)>0 whenever 0 < *&lambda* < 1 and *λ* is in *f*'s domain.

   Now let *h*(*λ*) = *f*(*λ*)/*g*(*λ*). 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 $\omega(\lambda)$ in Bernstein form, such that both polynomials' 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, elevate the degree of the polynomial with lesser degree to have the same degree as the other.

Now, let $g[j]$ and $\omega[j]$ be the j^{th} coefficient of the polynomial g or ω, respectively, in Bernstein form. 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 $f(\lambda)$ is described in the strong simulability statement and is Lipschitz continuous, then f is strongly simulable.*

Lemma 3. *If $f(\lambda)$ 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 absolute value of the maximum 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 coefficients), then set the first coefficient as l, the last coefficient as u, and the remaining coefficients as 1. (As a result, the polynomial g will have computable coefficients.) Then g will meet the additional condition for Lemma 1 and the result follows from that lemma. □

Lemma 4. *If $f(\lambda)$ 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 coefficients), then for each polynomial, set its first coefficient as l and the last coefficient as 1. The remaining coefficients of g are set as 1 and the remaining coefficients of ω are set as 0. (As a result, the polynomial g will have computable 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. □

I suspect that a function f that maps the closed interval or a subset thereof to the closed unit interval is strongly simulable if and only if—

- f is constant on its domain, or is continuous and polynomially bounded on its domain, and

- $f(0)$ and $f(1)$ are each 0, 1, or undefined, and
- 0 is not in the domain of f, or there are $\epsilon>0$ and a Lipschitz continuous function $g(x)$ such that $f(x)=g(x)$ whenever $0\le x\lt \epsilon$ and x is in f's domain, and
- 1 is not in the domain of f, or there are $\epsilon>0$ and a Lipschitz continuous function $h(x)$ such that $f(x)=h(x)$ whenever $1-\epsilon\lt x\le 1$ and x is in f's domain.

This is suspected by observing that for every integer $n>0$ and every Bernoulli factory function $f(x)$, $(1-(1-x)^n) \cdot f(x)$ is Lipschitz continuous at 0 with Lipschitz constant no more than n. ($(1-x)^n$ is the probability of the biased coin showing zero n times in a row.)

9.6 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 $a \le \lambda \le b$, 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)[52].)

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

- The functions λ and $1 - \lambda$.
- Constants *c* satisfying $0 \le c \le 1$. As Nacu and Peres (2005)[53] already showed, any such constant admits an optimal factory: generate unbiased random bits using Peres's iterated von

Neumann extractor (Peres 1992)[54], then build a binary tree that generates 1 with probability *c* and 0 otherwise (Knuth and Yao 1976)[55].

It is easy to see that if an *optimal factory* exists for *f*(*λ*), then one also exists for 1 − *f*(*λ*): simply change all ones returned by the *f*(*λ*) 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*(*λ*) = *λ*/2: the key is to flip the input coin enough times to produce unbiased random bits using his extractor (Peres 1992)[56], then multiply each unbiased bit with another input coin flip to get a sample from *λ*/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)[57]. 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:
    - 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 $\lambda/2$, 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)[58], 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*r} − 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.
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−λ)^r*$D(\lambda)$.
- outputting either 0 or 1 in that step would be $P01 = \lambda^r$*(1−λ)^r*$E(\lambda)$,

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

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)[59]. 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.

9.7 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.
- A function $f(\lambda)$ is *polynomially bounded* if both $f(\lambda)$ and $1-f(\lambda)$ are greater than or equal to $\min(\lambda^n, (1-\lambda)^n)$ for some integer n (Keane and O'Brien 1994)[60].
- A *modulus of continuity* of a function f means a nonnegative and nowhere decreasing function ω on the closed unit interval, for which $\omega(0) = 0$, and for which $\text{abs}(f(x) - f(y)) \le \omega(\text{abs}(x-y))$ whenever $0\le x\le 1$ and $0\le y\le 1$. Loosely speaking, a modulus of continuity $\omega(\delta)$ is greater than or equal to f's maximum range in a window of size δ.

Lemma 1. *Let f(λ) be a continuous and nowhere decreasing function, and let X_k be a hypergeometric(2*n, k, n) random variable, where n≥1 is a constant integer and k is an integer in [0, 2*n] . Then the expected value of f(X_k/n) decreases nowhere as k increases.*

Proof. This is equivalent to verifying whether X_{m+1}/n "dominates" X_m/n (and, obviously by extension, X_{m+1} "dominates" X_m) in terms of first-degree stochastic dominance (Levy 1998)[61]. This means that the probability that ($X_{m+1} \le j$) is less than or equal to that for X_m for each j satisfying $0 \le j \le n$. A proof of this was given by the user "Henry" of the *Mathematics Stack Exchange* community[62]. □

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, and let X be a hypergeometric(2*n, k, n) random variable.*

1. *Let ω(x) be a modulus of continuity of f. If ω is continuous and*

concave, then the expression—

$\mathrm{abs}(\mathbf{E}[f(X/n)] - f(k/(2*n))),\quad (1)$

is less than or equal to—

- $\omega(\mathrm{sqrt}(1/(8*n-4)))$*, for every integer* $n\geq 1$ *that's a power of 2,*
- $\omega(\mathrm{sqrt}(1/(7*n)))$*, for every integer* $n\geq 4$ *that's a power of 2,*
- $\omega(\mathrm{sqrt}(1/(2*n)))$*, for every integer* $n\geq 1$ *that's a power of 2, and*
- $\omega(\mathrm{sqrt}(\ (k/(2*n)) * (1-k/(2*n)) / (2*n-1)\ ))$*, for every* $n\geq 1$ *that's a power of 2.*

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—*
   - $M*(1/(2*n))^{\alpha/2}$*, for every integer* $n\geq 1$ *that's a power of 2,*
   - $M*(1/(7*n))^{\alpha/2}$*, for every integer* $n\geq 4$ *that's a power of 2, and*
   - $M*(1/(8*n-4))^{\alpha/2}$*, for every integer* $n\geq 1$ *that's a power of 2.*
3. *If f has a Lipschitz continuous derivative with Lipschitz constant M, then the expression (1) is less than or equal to—*
   - $(M/2)*(1/(7*n))$*, for every integer* $n\geq 4$ *that's a power of 2, and*
   - $(M/2)*(1/(8*n-4))$*, for every integer* $n\geq 1$ *that's a power of 2.*
4. *If f is convex, nowhere decreasing, and greater than or equal to 0, then the expression (1) is less than or equal to* $\mathbf{E}[f(Y/n)]$ *for every integer* $n\geq 1$ *that's a power of 2, where Y is a hypergeometric(2*n, n, n) random variable.*

Proof.

1. ω is assumed to be nonnegative because absolute values are nonnegative. To prove the first and second bounds: $\mathrm{abs}(\mathbf{E}[f(X/n)] - f(k/(2*n))) \leq \mathbf{E}[\mathrm{abs}(f(X/n) - f(k/(2*n))] \leq \mathbf{E}[\omega(\mathrm{abs}(X/n - k/(2*n))] \leq \omega(\mathbf{E}[\mathrm{abs}(X/n - k/(2*n))])$ (by Jensen's inequality and because ω is concave) $\leq \omega(\mathrm{sqrt}(\mathbf{E}[\mathrm{abs}(X/n - k/(2*n))]^2)) = \omega(\mathrm{sqrt}(\mathbf{Var}[X/n])) = \omega(\mathrm{sqrt}((k*(2*n-k)/(4*(2*n-1)*n^2)))) \leq \omega(\mathrm{sqrt}((n^2/(4*(2*n-1)*n^2)))) = \omega(\mathrm{sqrt}((1/(8*n-4)))) = \rho$, and for every $n\geq 4$ that's an integer power of 2, $\rho \leq \omega(\mathrm{sqrt}(1/(7*n)))$. To prove the third bound: $\mathrm{abs}(\mathbf{E}[f(X/n)] - f(k/(2*n))) \leq \omega(\mathrm{sqrt}(\mathbf{Var}[X/n])) \leq \omega(\mathrm{sqrt}(1/(2*n)))$. To prove the fourth bound: $\mathrm{abs}(\mathbf{E}[f(X/n)] - f(k/(2*n))) \leq \omega(\mathrm{sqrt}((n^2/(4*(2*n-1)*n^2)))) = \omega(\mathrm{sqrt}(\ (k/(2*n)) * (1-k/(2*n)) / (2*n-1)\ ))$.
2. By the definition of Hölder continuous functions, take $\omega(x) = M*x^{\alpha}$. 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 $f'(X/n) E(X/n-k/(2n)) = 0$. Moreover, $|f(x)-f(s)-f'(x)(x-s)|\le (M/2)(x-s)^2$ (see Micchelli 1973, Theorem 3.2)[65], so— $$E[|f(X/n)-f(k/(2n))|]=|E[f(X/n)-f(k/(2n))-f'(k/(2n))(X/n-k/(2n))]|$$ $$\le (M/2)(X/n-k/(2n))^2 \le (M/2) Var(X/n).$$ By part 1's proof, it follows that $(M/2)*\mathbf{Var}[X/n] = (M/2)*(k*(2 * n-k)/(4*(2 * n-1)*n^2)) \le (M/2)*(n^2/(4*(2 * n-1)*n^2)) = (M/2)*(1/(8*n-4)) = \rho$. For every integer $n\ge4$ that's a power of 2, $\rho \le (M/2)*(1/(7*n))$.

4. Let X_m be a hypergeometric($2 * n$, m, n) random variable. By Lemma 1 and the assumption that f is nowhere decreasing, $\mathbf{E}[f(X_k/n)]$ is nowhere decreasing as k increases, so take $\mathbf{E}[f(X_n/n)] = \mathbf{E}[f(Y/n)]$ as the upper bound. Then, abs($\mathbf{E}[f(X/n)] - f(k/(2 * n))$) = abs($\mathbf{E}[f(X/n)] - f(\mathbf{E}[X/n])$) = $\mathbf{E}[f(X/n)] - f(\mathbf{E}[X/n])$ (by Jensen's inequality, because f is convex and not less than 0) = $\mathbf{E}[f(X/n)] - f(k/(2 * n)) \le \mathbf{E}[f(X/n)]$ (because f is not less than 0) $\le \mathbf{E}[f(Y/n)]$. □

> **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 ω 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].

Theorem 1. *Let f be a strictly bounded factory function, let $n_0\ge 1$ 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 $n\ge n_0$ is an integer power of 2. Let—*

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

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

*There are polynomials g_n and h_n (where $n\ge 1$ 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 $0\le k\le n$), 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.*

The polynomials g_n and h_n satisfy:

1. *$g_n \le h_n$.*
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 $n\ge n_0$, 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)[72] by bounding, from below, the offset by which to shift the approximating polynomials. This lower bound is $\eta(n)$, a solution to the equation $0 = \eta(n) - \eta(2 * n) - \varphi(n)$ (see note below), where $\varphi(n)$ is a function that takes on a nonnegative value.

$\varphi(n)$ 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 $\varphi(n)$ 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 $n_0 > 1$.

Then we can take advantage of the observation in Remark B of Nacu and Peres (2005)[73] 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 as polynomials in Bernstein form, the coefficients of each one 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 elevated to 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*** and linear recurrences, with a solution that varies depending on the starting value, 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 $0 = \eta(n) - \eta(4 * n) - \varphi(2*n) - \varphi(n) = \eta(n) - \eta(8 * n) - \varphi(4*n) - \varphi(2*n) - \varphi(n) = \ldots$
> 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)*[74] *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 coefficients of h_n (the upper polynomial) are all 1.*
- *If* ***fbelow****(n, k) < 0 for a given n and some k, the coefficients of g_n (the lower polynomial) are all 0.*

Proof: Condition (i) of Proposition 3 says that each 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 $n\ge n_0$ 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 coefficient to 0 or 1, respectively, whenever a 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 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)[75].

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 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 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)[76]. □

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) − η(n).*
- ***fabove****(n, k) = max(****fabove****(4,0),* ***fabove****(4,1), ...,* ***fabove****(4,4)) if n < 4; otherwise, f(k/n) + η(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 $n\ge 1$ 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 $n\ge 4$ 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 $\eta(n)=$ $M*(2/7)^{\alpha/2}/((2^{\alpha/2}-1)*n^{\alpha/2})$.

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 $D(n) = D(2 * n) + \varphi(n)$ 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: `rsolve(Eq(f(n),f(n+1)+z*(1/(2*2**n))**((S(1)/2)/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: ($M*$ $\sum_{i=0}^{2j-1} 2^{i/(2j)})/n^{1/(2*j)}$ = M / $((2^{1/(2*j)}-1)*n^{1/(2*j)})$; 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. □

> **Note:** The first scheme given here is a special case of Theorem 1 that was already found by Nacu and Peres (2005)[77].

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 $n\ge 4$ 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)[78] 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 $\eta(n)$ in Theorem 1 remains nonnegative; also, this theorem adopts Theorem 1's assumption that the series converges, so that $\eta(n)$ 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)[79]. □

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.

Lemma 3. *Let f be a strictly bounded factory function. Let S be an infinite set of positive integers. For each integer $n\ge 1$, 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 ($0\le k\le n$).*

2. *$g_n \le h_n$.*
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, $|f(\lambda) - B_n(W_n(\lambda))| \le \epsilon_n(f)$ whenever $0\le \lambda\le 1$, 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)[80], which implies that, for every $m>n$, $g_{n}(\lambda)\le g_{m}(\lambda)\le f(\lambda)$ (the lower polynomials "increase") and $h_{n}(\lambda)\ge h_{m}(\lambda)\ge f(\lambda)$ (the upper polynomials "decrease") for every $n\ge 1$ (Nacu and Peres 2005, Remark A)[81].

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 $h_n = B_n(W_n(\lambda))+\epsilon_n(f) \ge f(\lambda)$, and rearranging this expression leads to $f(\lambda) - B_n(W_n(\lambda)) \le \epsilon_n(f)$.

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 $g_n = B_n(W_n(\lambda))-\epsilon_n(f) \le f(\lambda)$, and rearranging this expression leads to $B_n(W_n(\lambda)) - f(\lambda) \le \epsilon_n(f)$.

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

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

9.7.1 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 $r\ge 1$, and let f be a strictly bounded factory function whose r-th derivative is continuous. Let M be the absolute value of the maximum 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 $n\ge1$ that's a power of 2, suppose that there is $D>0$ such that— $$|f(\lambda)-B_n(W_n(\lambda))| \le DM/n^{r/2},$$ whenever $0\le \lambda\le 1$, where $B_n(W_n(\lambda))$ is the degree-n Bernstein polynomial of $W_n(\lambda)$.

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

1. g_n and h_n have Bernstein coefficients $W_n(k/n) - CM/n^{r/2}$ and $W_n(k/n) + CM/n^{r/2}$, respectively ($0\le k\le n$), if n is a power of 2, and $g_n=g_{n-1}$ and $h_n=h_{n-1}$ otherwise.
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 $n+1$.

Equivalently (see also Nacu and Peres 2005), there is $C_1>0$ such that, for each integer $n\ge 1$ that's a power of 2— $$\left|\left(\sum_{i=0}^k \left(W_n\left(\frac{i}{n}\right)\right){n\choose i}{n\choose {k-i}}/{2n \choose k}\right)-W_{2n}\left(\frac{k}{2n}\right)\right|\le \frac{C_1 M}{n^{r/2}},$$ whenever $0\le k\le 2n$, so that $C=\frac{C_1}{1-\sqrt{2/2^{r+1}}}$.

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 absolute value of the maximum of f and the Lipschitz constants of f and its derivatives up to the $(r-1)$-th derivative.

> **Note:** 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)[82].

The following lower bounds on C_0 can be shown. In the table:

- M_{r} is the absolute value of the maximum 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 $n\ge 1$.

If r is...	And...	And W_n is...	Then C_0 must be greater than:	And C_0 is conjectured to be:	Be cou
3	$M=M_{3}$	$2 f - B_n(f)$	0.29	$\frac{3}{16-4 \sqrt{2}}$ < 0.29005.**	$2 \ - \la
3	$M=M_{3}$, $n\ge 4$	$2 f - B_n(f)$	0.08	0.09	$2 \ - \la
4	$M=M_{4}$	$2 f - B_n(f)$	0.24	0.25	$2 \ - \la
4	$M=M_{4}$, $n\ge 4$	$2 f - B_n(f)$	0.14	0.15	$2 \ - \la
5	$M=M_{5}$	$B_n(B_n(f))+3(f-B_n(f))$***	0.26	0.27	$2 \ - \la
5	$M=M_{5}$, $n\ge 4$	$B_n(B_n(f))+3(f-B_n(f))$	0.10	0.11	$3 \ \cdo \lam
6	$M=M_{6}$	$B_n(B_n(f))+3(f-B_n(f))$	0.24	0.25	$2 \ - \la
6	$M=M_{6}$,	$B_n(B_n(f))+3(f-	0.22	0.23	$3 \

	$n\ge 4$	B_n(f))$			\cdo \lam

* Corresponds to the iterated Bernstein polynomial of order 2 (Güntürk and Li 2021)[83].

*** Corresponds to the iterated Bernstein polynomial of order 3 (Güntürk and Li 2021)[84].

** The following is evidence for the conjectured bound, at least if $f(0)=f(1)$.

The Bernstein polynomials $B_n(f)$ satisfy the partition-of-unity property: $\sum_{k=0}^n (c+f(k/n)) p_{n,k} = c+\sum_{k=0}^n f(k/n) p_{n,k}$ for every c, where $\sum_{k=0}^n p_{n,k}=1$. So do the hypergeometric probabilities $\sigma_{n,k,i} = {n\choose i}{n\choose {k-i}}/{2n \choose k}$. Thus, rewrite $\sum_{k=0}^n (c+W_n(i/n)) \sigma_{n,k,i}$ = $\sum_{k=0}^n (2 (c+f) - B_n(c+f)) \sigma_{n,k,i}$ = $\sum_{k=0}^n (2c + 2f - c - B_n(f)) \sigma_{n,k,i}$ = $\sum_{k=0}^n (c + 2f - B_n(f)) \sigma_{n,k,i}$ = $c + \sum_{k=0}^n W_n(i/n) \sigma_{n,k,i}$. And rewrite $c + \sum_{k=0}^n W_n(i/n) \sigma_{n,k,i} - (c+W_{2n}(k/(2n))) = \sum_{k=0}^n W_n(i/n) \sigma_{n,k,i} - W_{2n}(k/(2n))$. Thus, when $W_n = 2f-B_n(f)$, $f(0)$ can equal 0 without loss of generality.

Suppose $f(0) = f(1) = 0$, then $W_1(f)$ will equal 0. Let X be a hypergeometric(2,k,1) random variable (for k equal to 0, 1, or 2). Then $E[f(X/1)]$ will likewise equal 0. Thus, since $W_n(f)(0)=f(0)$ and $W_n(f)(1)=f(1)$ for every n, and since $W_2(f)(1/2) = 2f(1/2) - B_2(f)(1/2)$ (and thus considers only the values of f at 0, 1/2, and 1), $|E[f(X/1)] - f(k/2)|$ will take its maximum at $k=1$, namely $|0 - (2f(1/2) - B_2(f)(1/2))| = |- 3f(1/2)/2|$. That right-hand side is the minimum shift needed for the consistency requirement to hold; call it z, and let y be M_{3}. To get the minimum C_0 that works, solve $C_0 y/1^{3/2} - C_0 y/2^{3/2}$ = z for C_0. The solution is — $$C_0=\frac{z}{y} \frac{4}{4-\sqrt{2}} = \frac{|0-f(1/2)|}{y} \frac{12}{2\cdot (4-\sqrt{2})}.$$

The solution shows that if $y = M_{3}$ can come arbitrarily close to 0, then no value for C_0 will work. Which is why the goal is now to find a tight upper bound on the least possible value of M_{3} for $r=3$ such that:

1. $f(\lambda)$ has a continuous third derivative.
2. $f(0)=f(1)=0$ and $0 < f(1/2) < 1$.

Take the function $g(\lambda)=2\lambda(1-\lambda)$, which satisfies (1), (2), and $g(0)=g(1)=0$ and $g(1/2)=1/2$, and has an M_{3} of 4. Given that $\frac{|0-g(1/2)|}{y}=\frac{|0-1/2|}{y}=1/8$, the goal is now to see whether any function f satisfying (1) and (2) has $\max(0, f(1/2)) < M_{3} < 8\cdot |0-f(1/2)|=8 f(1/2)$.

To aid in this goal, there is a formula to find the least possible Lipschitz constant for f's first derivative (see "Definitions")[85], given a finite set of points (0, 1/2, and 1 in the case at hand) and the values of f and f' at those points (Le Gruyer 2009)[86]; see also (Herbert-Voss et al. 2017)[87]. Denote $L(.,.,.)$ as this least possible Lipschitz constant. Then according to that formula— $$L([0, 1/2, 1], [0, t, 0], [z_1, z_2, z_3]) = \max(2 \sqrt{\left|{z_{1} - z_{2}}\right|^{2} + \left|{- 4 t + z_{1} + z_{2}}\right|^{2}} + 2 \left|{- 4 t + z_{1} + z_{2}}\right|,$$ $$\sqrt{\left|{z_{1} - z_{3}}\right|^{2} + \left|{z_{1} + z_{3}}\right|^{2}} + \left|{z_{1} + z_{3}}\right|,$$ $$2 \sqrt{\left|{z_{2} - z_{3}}\right|^{2} + \left|{4 t + z_{2} + z_{3}}\right|^{2}} + 2 \left|{4 t + z_{2} + z_{3}}\right|)$$ (where t is greater than 0 and less than 1), so only f' values in the interval $[-8f(1/2), 8f(1/2)]$ have to be checked.

Let $H = 8\cdot |\beta-f(1/2)|$. In this case, only values of f in the closed unit interval have to be checked and only f' values in $[-H, H]$ have to be checked.

Assuming that no M_{3} less than $8\cdot |0-f(1/2)|$ is found, then — $$C_0=\frac{|0-f(1/2)|}{y} \frac{12}{2\cdot (4-\sqrt{2})}=\frac{|0-f(1/2)|}{H} \frac{12}{2\cdot (4-\sqrt{2})}=(1/8) \frac{12}{2\cdot (4-\sqrt{2})}$$ $$=3/(16-4\sqrt{2}),$$ which is the conjectured lower bound for C_0.

9.8 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, subject to that theorem's bounding note:

- $\eta(k) = \Phi(1, 2, (\ln(k)+\ln(7)+6)/\ln(2))*4/\ln(2)^2$.
- **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 Φ(.) 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 $f(0) = 0$, we can take $\omega(h) = f(h)$.

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

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

Now, by applying Theorem 1, we 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:

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

where Φ(.) 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>we can 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))
```

9.9 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 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 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:

- $\lambda^{1/p}$ for prime numbers p greater than 2. The answer may be no; Banderier and Drmota (2015)[88] 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 coefficients are non-negative real numbers.
- min(λ, 1−λ).

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 coefficients are rational numbers). **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 (a *coefficient*) is the probability of getting i.

> **Notes:**
>
> 1. Mossel and Peres (2005)[89] 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)[90] and can be translated to those grammars as well as to *probabilistic pushdown systems* (Etessami and Yannakakis 2009)[91], as long as those grammars and systems use only transition probabilities that are rational numbers.

Proposition 0 (Mossel and Peres 2005[92], 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**.*

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—*

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

*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)[93]: 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 $f(\lambda)$, $g(\lambda)$, and $h(\lambda)$ are functions in the class **PDA**, then so is $f(\lambda)*g(\lambda) + (1-f(\lambda))*h(\lambda)$.*

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 $f(\lambda)$ and $g(\lambda)$ are functions in the class **PDA**, then so is their composition, namely $f(g(\lambda))$ or $(f\circ g)(\lambda)$.*

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)[94] because the machine is a full-domain pushdown automaton. □

Proposition 3: *Every rational function with rational 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)[95]. 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, {}),

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)[96], 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)[97] 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)[98]):

- $X \to a$ (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 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 → 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.
- $X \rightarrow_{p_i} (a_i, Y_i)$ (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)[99]. If a_i is ε, no letter-generating operation is done.
- X → (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 → a becomes $X = 1$ if a is the empty string (ε), or $X = \lambda$ otherwise.
- For each nonterminal X, all n rules of the form $X \rightarrow_{p_i} (a_i, Y_i)$ become the equation $X = p_1*\lambda_1*Y_1 + p_2*\lambda_2*Y_2 + ... + p_n*\lambda_n*Y_n$, where λ_i is either 1 if a_i is ε, or λ otherwise.
- X → (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[100],

Icard 2020[101].) □

> **Example:** The stochastic context-free grammar—
> $X \rightarrow_{1/2} (a, X1)$,
> $X1 \rightarrow (X, X2)$,
> $X2 \rightarrow (X, X)$,
> $X \rightarrow_{1/2} (a, X3)$,
> $X3 \rightarrow \varepsilon$,
> which encodes ternary trees (Flajolet et al. 2010)[102], 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)[103]. 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)[104]; see also Esparza et al. (2004)[105]. For each state of the automaton (call the state en), include the following equations in the system based on the automaton's transition rules:
>
>   - $(st, p, sy) \rightarrow (s2, \{\})$ becomes either $\alpha_{st,sy,en} = p$ if $s2$ is en, or $\alpha_{st,sy,en} = 0$ otherwise.
>   - $(st, p, sy) \rightarrow (s2, \{sy1\})$ becomes $\alpha_{st,sy,en} = p * \alpha_{s2,sy1,en}$.
>   - $(st, p, sy) \rightarrow (s2, \{sy1, sy2\})$ becomes $\alpha_{st,sy,en} = p*\alpha_{s2,sy2,\sigma[1]}*\alpha_{\sigma[1],sy1,en} + ... + p*\alpha_{s2,sy2,\sigma[n]}*\alpha_{\sigma[n],sy1,en}$, where $\sigma[i]$ 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 $\alpha_{\text{START},\text{EMPTY},fs}$ (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:

    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)[106].

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.

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**, a recursive Markov chain for the expansion (Etessami and Yannakakis 2009)[107] 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)[108], 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)[109]. □

Corollary 1: *The function $f(\lambda) = \lambda^{m/(2^n)}$, where $n \ge 1$ is an integer and where $m \ge 1$ 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)[110] because the machine is a full-domain pushdown automaton. □

9.9.1 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)[111] 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 coefficients and satisfy `PDF(z) > 0` whenever 0 ≤ `z` ≤ 1 is irrational (Vatan 2001)[112], (Kindler and Romik 2004)[113], and it can be shown that the expected value (mean or "long-run average") of X must be a rational number. [114]

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 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)[115] 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.
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)[116]. 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)[117]. The branching process outputs no digit, and part 3 of the proposition follows from Corollary 9 of Icard (2020)[118]. 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)[119], the generator can generate a real number's binary expansion only if that number is rational or transcendental (see also Cobham (1968)[120]; Adamczewski and Bugeaud (2007)[121]). □

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)[122] and Knuth and Yao (1976)[123] with Richman (2012)[124], who proved that a continuous algebraic function on an open interval is piecewise analytic ("analytic" means writable as $c_0 x^0+ ... +c_i x^i + ...$ where $c_i$ are real numbers).

10 License

---

1. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

2. Thomas, A.C., Blanchet, J., "**A Practical Implementation of the Bernoulli Factory**", arXiv:1106.2508v3 [stat.AP], 2012.↩

3. 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 $\lambda$ is near 0, and extremely slowly when $\lambda$ 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.↩

4. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

5. In Nacu and Peres (2005), the following polynomial sequences were suggested to simulate $\min(2\lambda, 1-2\varepsilon)$ (using the algorithms in "General Factory Functions"), provided $\varepsilon \lt 1/8$, 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*)** = $\min(2(k/n), 1-2\varepsilon)$ + $\frac{2\times\max(0, k/n+3\varepsilon - 1/2)}{\varepsilon(2-\sqrt{2})}$ $\sqrt{2/n}$ + $\frac{72\times\max(0,k/n-1/9)}{1-\exp(-2\times\varepsilon^2)} \exp(-2n\times\varepsilon^2)$.↩

6. 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.↩

7. E. Voronovskaya, "Détermination de la forme asymptotique d'approximation des fonctions par les polynômes de M. Bernstein", 1932.↩

8. G.G. Lorentz, "Approximation of functions", 1966.↩

9. Mathé, Peter. "Approximation of Hölder Continuous Functions by Bernstein Polynomials." The American Mathematical Monthly 106, no. 6 (1999): 568–74. **https://doi.org/10.2307/2589469**.↩

10. Sikkema, P.C., "Der Wert einiger Konstanten in der Theorie der Approximation mit Bernstein-Polynomen", 1961.↩

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

12. Guan, Zhong. "**Iterated Bernstein polynomial approximations**", arXiv:0909.0684 (2009).↩

13. Güntürk, C.S., Li, W., "**Approximation of functions with one-bit neural networks**", arXiv:2112.09181 [cs.LG], 2021.↩

14. Tachev, Gancho. "**Linear combinations of two Bernstein polynomials**", *Mathematical Foundations of Computing*, 2022.↩

15. Tachev, Gancho. "**Linear combinations of two Bernstein polynomials**", *Mathematical Foundations of Computing*, 2022.↩

16. Güntürk, C.S., Li, W., "**Approximation of functions with one-bit neural networks**", arXiv:2112.09181 [cs.LG], 2021.↩

17. Güntürk, C.S., Li, W., "**Approximation of functions with one-bit neural networks**", arXiv:2112.09181 [cs.LG], 2021.↩

18. More generally, the coefficients can be real numbers, but there are computational issues. Rational numbers more easily support arbitrary precision than other real numbers, where special measures are required such as so-called constructive/recursive reals.↩

19. Carvalho, Luiz Max, and Guido A. Moreira. "**Adaptive truncation of infinite sums: applications to Statistics**", arXiv:2202.06121 (2022).↩

20. S. Ray, P.S.V. Nataraj, "A Matrix Method for Efficient Computation of Bernstein Coefficients", Reliable Computing 17(1), 2012.↩

21. Kawamura, Akitoshi, Norbert Müller, Carsten Rösnick, and Martin Ziegler. "**Computational benefit of smoothness: Parameterized bit-complexity of numerical operators on analytic functions and Gevrey’s hierarchy**." Journal of Complexity 31, no. 5 (2015): 689-714.↩

22. M. Gevrey, "Sur la nature analytique des solutions des équations aux dérivées partielles", 1918.↩

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

24. For this approximation, if n were infinity, the method would return 1 with probability 1 and so would not approximate $\lambda*c$, of course.↩

25. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

26. 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.↩

27. Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010.↩

28. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

29. 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.↩

30. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

31. Holtz, O., Nazarov, F., Peres, Y., "New Coins from Old, Smoothly", *Constructive Approximation* 33 (2011).↩

32. Holtz, O., Nazarov, F., Peres, Y., "New Coins from Old, Smoothly", *Constructive Approximation* 33 (2011).↩

33. Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010.↩

34. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

35. 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.↩

36. Carvalho, Luiz Max, and Guido A. Moreira. "**Adaptive truncation of infinite sums: applications to Statistics**", arXiv:2202.06121 (2022).↩

37. Qian, Weikang, Marc D. Riedel, and Ivo Rosenberg. "Uniform approximation and Bernstein polynomials with coefficients in the unit interval." European Journal of Combinatorics 32, no. 3 (2011): 448-463.↩

38. Li, Zhongkai. "Bernstein polynomials and modulus of continuity." Journal of Approximation Theory 102, no. 1 (2000): 171-174.↩

39. G.G. Lorentz, "Approximation of functions", 1966.↩

40. *Summation notation*, involving the Greek capital sigma (Σ), is a way to write the sum of one or more terms of similar form. For example, $\sum_{k=0}^n g(k)$ means $g(0)+g(1)+...+g(n)$, and $\sum_{k\ge 0} g(k)$ means $g(0)+g(1)+...$.↩

41. Henry (https://math.stackexchange.com/users/6460/henry), Proving stochastic dominance for hypergeometric random variables, URL (version: 2021-02-20): **https://math.stackexchange.com/q/4033573** .↩

42. Skorski, Maciej. "**Handy formulas for binomial moments**", arXiv:2012.06270 (2020).↩

43. Bustamante, J., "Estimates of positive linear operators in terms of second order moduli", J. Math. Anal. Appl. 345 (2008).↩

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

45. Also known as the Ditzian–Totik second modulus of smoothness (Ditzian, Z., Totik, V., "Moduli of smoothness", Springer, 1987).↩

46. A.F. Timan, Theory of Approximation of Functions of a Real Variable, 1994.↩

47. G.G. Lorentz, "Approximation of functions", 1966.↩

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

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

50. 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.↩

51. von Neumann, J., "Various techniques used in connection with random digits", 1951.↩

52. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

53. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

54. Peres, Y., "**Iterating von Neumann's procedure for extracting random bits**", Annals of Statistics 1992,20,1, p. 590-597.↩

55. 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.↩

56. 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.↩

57. Peres, Y., "**Iterating von Neumann's procedure for extracting random bits**", Annals of Statistics 1992,20,1, p. 590-597.↩

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

59. S. Pae, "**Binarization Trees and Random Number Generation**", arXiv:1602.06058v2 [cs.DS], 2018.↩

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

61. Levy, H., *Stochastic dominance*, 1998.↩

62. Henry (https://math.stackexchange.com/users/6460/henry), Proving stochastic dominance for hypergeometric random variables, URL (version: 2021-02-20): **https://math.stackexchange.com/q/4033573** .↩

63. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

64. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

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.↩

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. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

73. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

74. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

75. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

76. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

77. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

78. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

79. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

80. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

81. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

82. Nacu, Şerban, and Yuval Peres. "**Fast simulation of new coins from old**", The Annals of Applied Probability 15, no. 1A (2005): 93-115.↩

83. Güntürk, C.S., Li, W., "**Approximation of functions with one-bit neural networks**", arXiv:2112.09181 [cs.LG], 2021.↩

84. Güntürk, C.S., Li, W., "**Approximation of functions with one-bit neural networks**", arXiv:2112.09181 [cs.LG], 2021.↩

85. This formula applies to functions with Lipschitz-continuous derivative (a weaker assumption than having three continuous derivatives), but that derivative's Lipschitz constant is a lower

bound on M_{3}, so that formula is useful here.↩

86. Le Gruyer, Erwan. "Minimal Lipschitz extensions to differentiable functions defined on a Hilbert space." Geometric and Functional Analysis 19, no. 4 (2009): 1101-1118.↩

87. Herbert-Voss, Ariel, Matthew J. Hirn, and Frederick McCollum. "Computing minimal interpolants in C1, 1 (Rd)." Rev. Mat. Iberoam 33, no. 1 (2017): 29-66.↩

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

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

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

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

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

93. 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.↩

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

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

96. 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.↩

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

98. 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.↩

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

100. Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010↩

101. Levy, H., *Stochastic dominance*, 1998.↩

102. Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010.↩

103. Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010↩

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

105. 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.↩

106. 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.↩

107. 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.↩

108. 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.↩

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.↩

111. 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.↩

112. 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.↩

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

114. 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.↩

115. 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.↩

116. 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.↩

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

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

119. 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.↩

120. 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.↩

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

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

123. 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.↩

124. Richman, F. (2012). Algebraic functions, calculus style. Communications in Algebra, 40(7), 2671-2683.↩