Supplemental Notes for Bernoulli Factory Algorithms

This version of the document is dated 2022-08-11.

Peter Occil

1 Contents

2 Definitions

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

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 interval [0, 1], that is, when $0 \le x \le 1$.

- If f is continuous, its *derivative* is, roughly speaking, its "slope" or "velocity" function. The derivative (or *first derivative*) is denoted as $f\prime$. The *second derivative* ("slope-of-slope") of f, denoted $f\prime\prime$, is the derivative of $f\prime$; the *third derivative* is the derivative of $f\prime\prime$; 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 f's domain and no more than *δ* apart. Roughly speaking, the "steepness" of *f* is no greater than that of $M*x^{\alpha}$.

The function also admits a Hölder exponent β such that $0 < \beta < \alpha$.

- A *Lipschitz continuous* function with constant L (the *Lipschitz constant*) is Hölder continuous with Hölder exponent 1 and Hölder constant L.
  Roughly speaking, the "steepness" of f is no greater than that of $M*x$.
  If f has a derivative on its domain, L is the maximum absolute value 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 domain of f.
  Roughly speaking, if f's "slope" never goes down, then it's convex.
- A *concave* function f has the property that if $f((x+y)/2) \ge (f(x)+f(y))/2$ whenever x, y, and $(x+y)/2$ are in the domain of f.
  Roughly speaking, if f's "slope" never goes up, then it's concave.

3 General Factory Functions

As a reminder, the *Bernoulli factory problem* is: We're given 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 non-negative 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 k is an integer in the interval $[0, n]$.

3.1 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 f if f belongs to one of certain classes of factory functions. It would be helpful to plot the desired function f 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* on the interval [0, 1], 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* on the interval [0, 1], then **fabove**(n, k) can equal $f(k/n)$, thanks to Jensen's inequality. One example is $f(\lambda) = \exp(-\lambda/4)$.

Twice differentiable functions. The following method, proved in the appendix, implements **fabove** and **fbelow** if $f(\lambda)$—

- has a second derivative defined on [0, 1] (that is, the function is *twice differentiable* there), and
- in the interval [0, 1]—
  - 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 an upper bound of the highest value of abs($f''(x)$) over the domain [0, 1], where f'' is the second derivative of f. 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, $f(k/n) + m/(7*n)$.

My **GitHub repository** includes SymPy code for a method, `c2params`, to calculate the necessary values for m and the bounds of these polynomials, given f.

> **Note:** For this method, the second derivative need not be continuous (Y. Peres, pers. comm., 2021).
>
> **Example:** Take $f(\lambda) = \exp(-\lambda)$. This is a convex and twice differentiable function, and bounded below by 3321/10000. Then it can be shown that the following scheme for f is valid:
>
> - **fbelow**(n, k) = 3321/10000 if n<4; otherwise, $f(k/n) - 1/(7*n)$. (Observe that $f(k/4) - 1/(7*4) \geq 3321/10000$.)
> - **fabove**(n, k) = $f(k/n)$ (because f is convex).
> - **fbound**(n) = [0, 1].

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 [0, 1], with Hölder constant m and Hölder exponent α (see "**Definitions**"), and
- in the interval [0, 1]—
  - 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.

Finding m and α is non-trivial in general. But assuming m and α are known, then 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, $f(k/n) + D(n)$.

  Note:

  1. Some functions f 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 $-1/(2*\ln(\lambda/2)) + 1/10$ otherwise, which has a slope near 0 that's steeper than every "nth" root.
  2. If f 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.
  3. If f's Hölder exponent is 1/2 or greater, $D(n)$ can be $m*154563/(40000*n^{1/4})$, which is an upper bound.

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 ($h(\lambda)$) that "dominates" f at every point in the interval [0, 1]. Unlike for the original function, there might be a polynomial-building scheme described earlier in this section for the transformed function.

More specifically, $h(\lambda)$ must meet the following requirements:

- $h(\lambda)$ is continuous on the closed interval [0, 1].
- $h(0) = 0$. (This is required to ensure correctness in case λ is 0.)
- $1 \ge h(1) \ge f(1) \ge 0$.
- $1 > h(\lambda) > f(\lambda) > 0$ for every λ in the open interval (0, 1).
- If $f(1) = 0$, then $h(1) = 0$. (This is required to ensure correctness in case λ is 1.)

Also, h should be a function with a simple Bernoulli factory algorithm. For example, h 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**".

The algorithm is now described.

Let $g(\lambda) = \lim_{\nu\to\lambda} f(\nu)/h(\nu)$ (roughly speaking, the value that $f(\nu)/h(\nu)$ approaches as ν approaches λ.) If—

- $f(0) = 0$ and $f(1) < 1$, and
- $g(\lambda)$ is continuous on [0, 1] 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 h. If the call returns 0, return 0. (For example, if $h(\lambda) = \lambda$, 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 h functions that are found based on the current g function. See the second example below.

2. If—

   - f is strictly increasing,
   - $h(\lambda) = \lambda$, and
   - $f'(\lambda)$, the first derivative of f, is continuous on [0, 1], maps (0, 1) to (0, 1), and belongs in one of the classes of functions given earlier,

   then step 2 can be implemented by taking g as f', except: (A) a uniform random variate, greater than 0 and less than 1, is generated at the start of the step; (B) instead of flipping the input coin as normal during that step, a different coin is flipped that does the following: "Flip the input coin, then **sample from the number *u***. Return 1 if both the call and the flip return 1, and return 0 otherwise."
   This is the "**integral method**" of Flajolet et al. (2010)[^2] (the modified step 2 simulates $1/\lambda$ times the *integral* of f.).

Examples:

1. If $f(\lambda) = (\sinh(\lambda)+\cosh(\lambda)-1)/4$, then f is bounded from above by $h(\lambda) = \lambda$, 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 bounded from above by $h(\lambda) = \lambda$, so $g(\lambda)$ is 0 if $\lambda = 0$, and $(\cosh(\lambda)-1)/\lambda$ otherwise. Now, since $g(0) = 0$, find new functions g and h based on the current g. The current g is bounded from above by $H(\lambda) = \lambda*3*(2-\lambda)/5$ (a degree-2 polynomial that in Bernstein form has coefficients [0, 6/10, 6/10]), so $G(\lambda) = 5/12$ if $\lambda = 0$, and $-(5*\cosh(\lambda) - 5)/(3*\lambda^2*(\lambda-2))$ otherwise. G is bounded away from 0 and 1, resulting in the following algorithm:

   1. (Simulate h.) Flip the input coin. If it returns 0, return 0.
   2. (Simulate H.) 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 is twice differentiable) and return the result of that algorithm.

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

- $\omega(\lambda)$ is a function that meets the following requirements:
  - $\omega(\lambda)$ is continuous on the closed interval [0, 1].
  - $\omega(0) = 0$ and $\omega(1) = 1$.
  - $1 > f(\lambda) > \omega(\lambda) > 0$ for every λ in the open interval (0, 1).

- $q(\lambda)$ = $\lim_{\nu\to\lambda} \omega(\nu)/h(\nu)$.
- $r(\lambda)$ = $\lim_{\nu\to\lambda} (1-g(\nu))/(1-q(\nu))$.

Roughly speaking, ω is a function that bounds *f* from below, just as *h* bounds *f* from above. ω should be a function with a simple Bernoulli factory algorithm, such as a polynomial in Bernstein form. If both ω and *h* are polynomials of the same degree, *q* will be a rational function with a relatively simple Bernoulli factory algorithm (see "**Certain Rational Functions**").

Now, if $r(\lambda)$ is continuous on [0, 1], then *f* can be simulated using the following algorithm:

1. Run a Bernoulli factory algorithm for *h*. If the call returns 0, return 0. (For example, if $h(\lambda) = \lambda$, then this step amounts to the following: "Flip the input coin. If it returns 0, return 0.")
2. Run a Bernoulli factory algorithm for *q*(.). If the call returns 1, return 1.
3. Run a Bernoulli factory algorithm for *r*(.), and return 1 minus the result of that call. The Bernoulli factory algorithm can be one of the **general factory function algorithms** 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).

   Note: Quick proof: Rewrite $f=h\cdot(q\cdot1+(1-q)\cdot(1-r))+(1-h)\cdot0$.

   Example: If $f(\lambda) = (1-\exp(\lambda))/(1-\exp(1))$, then *f* is bounded from above by $h(\lambda) = \lambda$, and from below by $\omega(\lambda) = \lambda^2$. As a result, $q(\lambda) = \lambda$, and $r(\lambda) = (2 - \exp(1))/(1 - \exp(1))$ if $\lambda = 0$; $1/(\exp(1)-1)$ if $\lambda = 1$; and $(-\lambda*(1 - \exp(1)) - \exp(\lambda) + 1)/(\lambda*(1 - \exp(1))*(\lambda - 1))$ otherwise. This can be computed using the following code in Python that uses the SymPy computer algebra library: `fx=(1-exp(x))/(1-exp(1)); h=x; omega=x**2; q=(omega/h); r=(1-fx/h)/(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 endpoints 0 and/or 1. If *f* does not fully admit a polynomial-building scheme under the convex, concave, twice differentiable, 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 $f(\lambda) = 1 - f(1-\lambda)$. *Inverted coin*: Instead of the usual input coin, use a coin that does the following: "Flip the input coin and return 1 minus the result." *Inverted result:* If the overall algorithm would return 0, it returns 1 instead, and vice versa.
> 0 and < 1	0	Algorithm for **certain functions that equal 0 at 0**, but with $f(\lambda) = f(1-\lambda)$. (For example, $\cosh(\lambda)-1$ becomes $\cosh(1-\lambda)-1$.) Inverted coin.
1	0	Algorithm for **certain functions that equal 0 at 0 and 1 at 1**, but with $f(\lambda) = 1-f(\lambda)$. Inverted result.
1	> 0 and ≤ 1	Algorithm for **certain functions that equal 0 at 0**, but with $f(\lambda) = 1-f(\lambda)$. Inverted result.

Specific functions. My **GitHub repository** includes SymPy code for a method, `approxscheme2`, to build a polynomial-building scheme for certain factory functions.

4 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 interval [0, 1] to [0, 1], 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 − λ)). Continuous functions on [0, 1] can be approximated arbitrarily well by an approximate Bernoulli factory (as a result of the so-called "Weierstrass approximation theorem"), but this is not the case in general for discontinuous functions.

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 in [0, 1].

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:

   1. 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.
   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.)

4.1 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 converge no faster than the order of $1/n$ in general (Voronovskaya 1932)[^3].

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***(**λ**):	Then the degree-***n*** Bernstein polynomial is close to f with the following error bound:	Where ***n*** is:	Notes
Has Lipschitz continuous derivative (see "Definitions").	$\epsilon = M/(8*n)$.	$n = \text{ceil}(M/(8*\epsilon))$.	Lorentz (1966) [^4]. *M* is the derivative's Lipschitz constant.
Hölder continuous with constant *M* and exponent α.	$\epsilon = M*(1/(4*n))^{\alpha/2}$.	$n = \text{ceil}(1/(4^{\alpha}*\epsilon^2/M^2)^{1/\alpha})$.	Mathé (1999) [^5]. $0 < \alpha \le 1$.
Lipschitz continuous with constant *L*.	$\epsilon = L*\text{sqrt}(1/(4*n))$.	$n = \text{ceil}(L^2/(4*\epsilon^2))$.	Special case of previous entry.

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**.

An example is given by the iterated Bernstein polynomial construction discussed in Micchelli (1973)[^6] and Guan (2009)[^7]. Let $B_n(f(\lambda))$ be the ordinary Bernstein polynomial for $f(\lambda)$. Then—

- the order-2 iterated Bernstein polynomial of degree n is $U_{n,2} = B_n(W_{n,2})$, where $W_{n,2} = 2 f(\lambda) - B_n(f(\lambda))$, and
- the order-3 iterated Bernstein polynomial of degree n is $U_{n,3} = B_n(W_{n,3})$, where $W_{n,3} = B_n(B_n(f(\lambda))) + 3 (f(\lambda) -

B_n(f(\lambda)))$

(Güntürk and Li 2021, sec. 3.3)[^8]. The goal is now to find a degree n such that—

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

By analyzing the proof of Theorem 3.3 of the paper just cited, the following error bounds *appear* to be true. In the table below, M_n is not less than the so-called C^n norm. Unfortunately, the C^n norm is defined differently in different academic works, and the bounds are sensitive to how that norm is defined.[^9]

If ***f*(*λ*)**:	Then the following polynomial:	Is close to ***f*** with the following error bound:	Where ***n*** is:
Has continuous third derivative.	$U_{n,2}$	$\varepsilon = 0.3489*M_3/n^{3/2}$.	n=ceil($(0.3489)^{2/3}*(M_4/\varepsilon)^{2/3}$) < ceil((49561/100000)*$(M/\varepsilon)^{2/3}$).
Has continuous fourth derivative.	$U_{n,2}$	$\varepsilon = 0.275*M_4/n^2$.	n=ceil(sqrt(0.275)*sqrt(M_4/ε)) < ceil((52441/100000)*sqrt(M/ε)).
Has continuous fifth derivative.	$U_{n,3}$	$\varepsilon = 0.7284*M_5/n^{5/2}$.	n=ceil($(0.7284)^{2/5}*(M_5/\varepsilon)^{2/5}$) < ceil((88095/100000)*$(M/\varepsilon)^{2/5}$).
Has continuous sixth derivative.	$U_{n,3}$	$\varepsilon = 0.9961*M_6/n^3$.	n=ceil($(0.9961)^{1/3}*(M_6/\varepsilon)^{1/3}$) < ceil((99870/100000)*$(M/\varepsilon)^{1/3}$).

However, unlike with ordinary Bernstein polynomials, the polynomial W (and thus U) 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 ϵ.

1. Determine whether f is described in the table above. Let A be the minimum of f on [0, 1] and let B be the maximum of f there.
2. If $0 < A \le B < 1$, calculate n as given in the table 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 non-negative. If B is less than 1 and any of those conditions is met, calculate n as given in the table 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 the table above. Then, if $W_{n,i}(j/n)\lt 0$ or $W_{n,i}(j/n)\gt 1$ for some $0\le j\le n$, double the value of n until this condition is no longer true.

Once n is found, simulating the iterated 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 $W_{n,2}(j/n)$ or $W_{n,3}(j/n)$ (as the case may be), return 1. Otherwise, return 0.

    Note: Providing the full proof for the error bounds shown in the table is a bit tedious, so here is a sketch. The proof was found by analyzing Theorem 3.3 of Güntürk and Li (2021)[^8], 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. The most significant estimate in that theorem is denoted $(B_n-I)^{\lceil (r+1)/2 \rceil}(f)$, which in this case is the error when approximating f using an iterated Bernstein polynomial, when f has a continuous $(r+1)$-th derivative.

4.2 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[^10].

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 on the interval [0, 1].

   If f's coefficients are each greater than 0, form a non-increasing sequence, and meet the so-called "ratio test", the algorithms in Carvalho and Moreira (2022)[^11] 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 maximum absolute value of f's (n+1)-th derivative on [0, 1].

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)[^12].) Let $b_0, ..., b_n$ be the Bernstein-form polynomial's coefficients.

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)[^13]; see also (Gevrey 1918)[^14]).

4.3 Approximate Bernoulli Factories for Linear Functions

There are a number of approximate methods to simulate λ*c, where c > 1 and λ lies in [0, 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)[^15] approximates the function λ*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 $\lambda*c$[^16]: "(1.) Set j to 0 and i to 0; (2.) If $i \geq n$, 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 $(j/(i+1)) \geq 1/c$, 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)[^1], which approximates $\lambda*2$.

5 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 unknown probability of heads. In the table below:

- λ, the unknown probability of heads, lies in the interval $[\varepsilon, 1-\varepsilon]$ 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 "bounded from above by $h(n)$ times a constant, for every n 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 [0, 1] (Goyal and Sigman 2012)[^17].
Requires a finite number of flips on average. Also known as "realizable" by Flajolet et al. (2010)[^2].	Only if f is Lipschitz continuous (Nacu and Peres 2005)[^1]. Whenever f admits a fast simulation (Mendo 2019)[^18].
Number of flips required, raised to power of r, is finite on average and has a tail that drops off uniformly for every λ.	Only if f has continuous r-th derivative (Nacu and Peres 2005)[^1].
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)[^19].
Requires more than n flips with probability $\Delta(n, \alpha, \lambda)$, for non-integer $\alpha > 0$ and every λ. (The greater α is, the faster the simulation.)	If and only if f has an r-th derivative that is Hölder continuous with exponent $(\alpha - r)$, where r = floor(α) (Holtz et al. 2011)[^19]. 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)[^2].	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)[^1].
Average number of flips bounded from below by $(f'(\lambda))^2*\lambda*(1-\lambda)/(f(\lambda)*(1-f(\lambda)))$, where f' is the first derivative of f.	Whenever f admits a fast simulation (Mendo 2019)[^18].

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. If a function is constant on some non-empty open interval in its domain, but is not constant on the whole domain, then it can't be real analytic.
3. A function in the *Zygmund class*, roughly speaking, has no vertical slope. The Zygmund class includes the smaller class of Lipschitz continuous functions.

6 Complexity

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

Let **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, **E**[N] = 2*φ as shown below.

- Each iteration stops the algorithm with probability p = (1/2) + (1−(1/2)) * (1/φ) (1/2 for the initial bit and 1/φ for the recursive run; (1−(1/2)) because we're subtracting the (1/2) earlier on the right-hand side from 1).
- Thus, the expected number of iterations is **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 + **E**[N]) * (1/2) bits on average, so the whole algorithm uses **E**[N] = (1 * (1/2) + (1 + **E**[N]) * (1/2)) * **E**[T] bits on average (each iteration consumes either 1 bit with probability 1/2, or (1 + **E**[N]) bits with probability 1/2). This equation has the solution **E**[N] = 1 + sqrt(5) = 2*φ.

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.

7 Examples of Bernoulli Factory Polynomial-Building Schemes

The following are polynomial-building schemes and hints to simulate a coin of probability $f(\lambda)$ given an input coin with probability of heads of λ. The schemes were generated automatically using `approxscheme2` and have not been rigorously verified for correctness.

- Let $f(\lambda)$ = **cosh(λ) − 3/4**. Then, for every integer n that's a power of 2, starting from 1:
  - The function was detected to be convex and twice differentiable, leading to:
    - **fbelow**(n, k) = 487/2500 if n<4; otherwise, $f(k/n)$ − 154309/(700000*n).
    - **fabove**(n, k) = $f(k/n)$.
    - **fbound**(n) = [0, 1].
  - Generated using tighter bounds than necessarily proven:
    - **fbelow**(n, k) = 1043/5000 if n<4; otherwise, $f(k/n)$ − 462927/(2800000*n).

        - **fabove**(n, k) = $f(k/n)$.
        - **fbound**(n) = [0, 1].
- Let $f(\lambda)$ = **3/4 − sqrt(−λ*(λ − 1))**. Then, for every integer n that's a power of 2, starting from 1:
    - Detected to be convex and (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(n, k) = $f(k/n)$ − 1545784563/(400000000*$n^{1/4}$).
        - **fabove**(n, k) = $f(k/n)$.
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(n, k) = $f(k/n)$ − 26278337571/(25600000000*$n^{1/4}$).
        - **fabove**(n, k) = $f(k/n)$.
- Let $f(\lambda)$ = **3*sin(sqrt(3)*sqrt(sin(2*λ)))/4 + 1/50**. Then, for every integer n that's a power of 2, starting from 1:
    - Detected to be (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(n, k) = $f(k/n)$ − 709907859/(100000000*$n^{1/4}$).
        - **fabove**(n, k) = $f(k/n)$ + 709907859/(100000000*$n^{1/4}$).
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(n, k) = $f(k/n)$ − 6389170731/(3200000000*$n^{1/4}$).
        - **fabove**(n, k) = $f(k/n)$ + 6389170731/(3200000000*$n^{1/4}$).
- Let $f(\lambda)$ = **3/4 − sqrt(−λ*(λ − 1))**. Then, for every integer n that's a power of 2, starting from 1:
    - Detected to be convex and (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(n, k) = $f(k/n)$ − 1545784563/(400000000*$n^{1/4}$).
        - **fabove**(n, k) = $f(k/n)$.
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(n, k) = $f(k/n)$ − 26278337571/(25600000000*$n^{1/4}$).
        - **fabove**(n, k) = $f(k/n)$.
- Let $f(\lambda)$ = **λ*sin(7*π*λ)/4 + 1/2**. Then, for every integer n that's a power of 2, starting from 1:
    - Detected to be twice differentiable using numerical methods, which may be inaccurate:
        - **fbelow**(n, k) = 523/10000 if n<64; otherwise, $f(k/n)$ − 11346621/(700000*n).
        - **fabove**(n, k) = 1229/1250 if n<64; otherwise, $f(k/n)$ + 11346621/(700000*n).
        - **fbound**(n) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(n, k) = 681/10000 if n<32; otherwise, $f(k/n)$ − 34039863/(4480000*n).
        - **fabove**(n, k) = 4837/5000 if n<32; otherwise, $f(k/n)$ + 34039863/(4480000*n).
        - **fbound**(n) = [0, 1].
- Let $f(\lambda)$ = **sin(4*π*λ)/4 + 1/2**. Then, for every integer n that's a power of 2, starting from 1:
    - The function was detected to be twice differentiable, leading to:
        - **fbelow**(n, k) = 737/10000 if n<32; otherwise, $f(k/n)$ − 1973921/(350000*n).
        - **fabove**(n, k) = 9263/10000 if n<32; otherwise, $f(k/n)$ + 1973921/(350000*n).
        - **fbound**(n) = [0, 1].
    - Generated using tighter bounds than necessarily proven:

        - **fbelow**(n, k) = 1123/10000 if n<32; otherwise, $f(k/n)$ − 1973921/(448000*n).
        - **fabove**(n, k) = 8877/10000 if n<32; otherwise, $f(k/n)$ + 1973921/(448000*n).
        - **fbound**(n) = [0, 1].
- Let $f(\lambda)$ = **sin(6*π*λ)/4 + 1/2**. Then, for every integer n that's a power of 2, starting from 1:
    - The function was detected to be twice differentiable, leading to:
        - **fbelow**(n, k) = 517/10000 if n<64; otherwise, $f(k/n)$ − 2220661/(175000*n).
        - **fabove**(n, k) = 9483/10000 if n<64; otherwise, $f(k/n)$ + 2220661/(175000*n).
        - **fbound**(n) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(n, k) = 1043/10000 if n<64; otherwise, $f(k/n)$ − 104371067/(11200000*n).
        - **fabove**(n, k) = 8957/10000 if n<64; otherwise, $f(k/n)$ + 104371067/(11200000*n).
        - **fbound**(n) = [0, 1].
- Let $f(\lambda)$ = **sin(4*π*λ)/4 + 1/2**. Then, for every integer n that's a power of 2, starting from 1:
    - The function was detected to be twice differentiable, leading to:
        - **fbelow**(n, k) = 737/10000 if n<32; otherwise, $f(k/n)$ − 1973921/(350000*n).
        - **fabove**(n, k) = 9263/10000 if n<32; otherwise, $f(k/n)$ + 1973921/(350000*n).
        - **fbound**(n) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(n, k) = 1123/10000 if n<32; otherwise, $f(k/n)$ − 1973921/(448000*n).
        - **fabove**(n, k) = 8877/10000 if n<32; otherwise, $f(k/n)$ + 1973921/(448000*n).
        - **fbound**(n) = [0, 1].
- Let $f(\lambda)$ = **λ^2/2 + 1/10 if λ ≤ 1/2; λ/2 − 1/40 otherwise**. Then, for every integer n that's a power of 2, starting from 1:
    - Detected to be convex and twice differentiable using numerical methods, which may be inaccurate:
        - **fbelow**(n, k) = 321/5000 if n<4; otherwise, $f(k/n)$ − 1/(7*n).
        - **fabove**(n, k) = $f(k/n)$.
        - **fbound**(n) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(n, k) = 693/10000 if n<4; otherwise, $f(k/n)$ − 55/(448*n).
        - **fabove**(n, k) = $f(k/n)$.
        - **fbound**(n) = [0, 1].
- Let $f(\lambda)$ = **λ/2 if λ ≤ 1/2; (4*λ − 1)/(8*λ) otherwise**. Then, for every integer n that's a power of 2, starting from 1:
    - Detected to be concave and twice differentiable using numerical methods, which may be inaccurate:
        - **fbelow**(n, k) = $f(k/n)$.
        - **fabove**(n, k) = 893/2000 if n<4; otherwise, $f(k/n)$ + 2/(7*n).
        - **fbound**(n) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(n, k) = $f(k/n)$.
        - **fabove**(n, k) = 4197/10000 if n<4; otherwise, $f(k/n)$ + 5/(28*n).

        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **1/2 − sqrt(1 − 2*λ)/2 if λ < 1/2; sqrt(2*λ − 1)/2 + 1/2 otherwise**. Then, for every integer *n* that's a power of 2, starting from 1:
    - Detected to be (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*) − 1545784563/(400000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 1545784563/(400000000*$n^{1/4}$).
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*) − 10820491941/(12800000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 10820491941/(12800000000*$n^{1/4}$).
- Let *f*(λ) = **1/2 − sqrt(1 − 2*λ)/4 if λ < 1/2; sqrt(2*λ − 1)/4 + 1/2 otherwise**. Then, for every integer *n* that's a power of 2, starting from 1:
    - Detected to be (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*) − 772969563/(400000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 772969563/(400000000*$n^{1/4}$).
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = 193/5000 if *n*<16; otherwise, *f*(*k*/*n*) − 5410786941/(12800000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = 4807/5000 if *n*<16; otherwise, *f*(*k*/*n*) + 5410786941/(12800000000*$n^{1/4}$).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **λ/2 + (1 − 2*λ)$^{3/2}$/12 − 1/12 if λ < 0; λ/2 + (2*λ − 1)$^{3/2}$/12 − 1/12 if λ ≥ 1/2; λ/2 + (1 − 2*λ)$^{3/2}$/12 − 1/12 otherwise**. Then, for every integer *n* that's a power of 2, starting from 1:
    - Detected to be convex and Lipschitz continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*) − 322613/(500000*sqrt(n)).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*).
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*) − 3548743/(32000000*sqrt(n)).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*).
- Let *f*(λ) = **1/2 − sqrt(1 − 2*λ)/4 if λ < 1/2; sqrt(2*λ − 1)/4 + 1/2 otherwise**. Then, for every integer *n* that's a power of 2, starting from 1:
    - Detected to be (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*) − 772969563/(400000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 772969563/(400000000*$n^{1/4}$).
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = 193/5000 if *n*<16; otherwise, *f*(*k*/*n*) − 5410786941/(12800000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = 4807/5000 if *n*<16; otherwise, *f*(*k*/*n*) + 5410786941/(12800000000*$n^{1/4}$).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **1/2 − sqrt(1 − 2*λ)/8 if λ < 1/2; sqrt(2*λ − 1)/8 + 1/2 otherwise**. Then, for every integer *n* that's a power of 2, starting from 1:
    - Detected to be (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = 333/10000 if *n*<64; otherwise, *f*(*k*/*n*) − 386562063/(400000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = 9667/10000 if *n*<64; otherwise, *f*(*k*/*n*) +

        - 386562063/(400000000*n$^{1/4}$).
    - **fbound**(*n*) = [0, 1].
  - Generated using tighter bounds than necessarily proven:
    - **fbelow**(*n*, *k*) = 451/2000 if *n*<4; otherwise, *f*(*k*/*n*) − 2705934441/(12800000000*n$^{1/4}$).
    - **fabove**(*n*, *k*) = 1549/2000 if *n*<4; otherwise, *f*(*k*/*n*) + 2705934441/(12800000000*n$^{1/4}$).
    - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **λ/4 + (1 − 2*λ)$^{3/2}$/24 + 5/24 if λ < 0; λ/4 + (2*λ − 1)$^{3/2}$/24 + 5/24 if λ ≥ 1/2; λ/4 + (1 − 2*λ)$^{3/2}$/24 + 5/24 otherwise**. Then, for every integer *n* that's a power of 2, starting from 1:
  - Detected to be convex and Lipschitz continuous using numerical methods, which may be inaccurate:
    - **fbelow**(*n*, *k*) = 443/5000 if *n*<4; otherwise, *f*(*k*/*n*) − 322613/(1000000*sqrt(n)).
    - **fabove**(*n*, *k*) = *f*(*k*/*n*).
    - **fbound**(*n*) = [0, 1].
  - Generated using tighter bounds than necessarily proven:
    - **fbelow**(*n*, *k*) = 1111/5000 if *n*<4; otherwise, *f*(*k*/*n*) − 3548743/(64000000*sqrt(n)).
    - **fabove**(*n*, *k*) = *f*(*k*/*n*).
    - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **3*λ/2 if λ ≤ 1 − λ; 3/2 − 3*λ/2 otherwise**. Then, for every integer *n* that's a power of 2, starting from 1:
  - Detected to be concave and Lipschitz continuous using numerical methods, which may be inaccurate:
    - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
    - **fabove**(*n*, *k*) = 124/125 if *n*<64; otherwise, *f*(*k*/*n*) + 967839/(500000*sqrt(n)).
    - **fbound**(*n*) = [0, 1].
  - Generated using tighter bounds than necessarily proven:
    - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
    - **fabove**(*n*, *k*) = 1863/2000 if *n*<64; otherwise, *f*(*k*/*n*) + 2903517/(2000000*sqrt(n)).
    - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **9*λ/5 if λ ≤ 1 − λ; 9/5 − 9*λ/5 otherwise**. Then, for every integer *n* that's a power of 2, starting from 1:
  - Detected to be concave and Lipschitz continuous using numerical methods, which may be inaccurate:
    - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
    - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 2903517/(1250000*sqrt(n)).
  - Generated using tighter bounds than necessarily proven:
    - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
    - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 8710551/(5000000*sqrt(n)).
- Let *f*(λ) = **19*λ/20 if λ ≤ 1 − λ; 19/20 − 19*λ/20 otherwise**. Then, for every integer *n* that's a power of 2, starting from 1:
  - Detected to be concave and Lipschitz continuous using numerical methods, which may be inaccurate:
    - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
    - **fabove**(*n*, *k*) = 1817/2000 if *n*<8; otherwise, *f*(*k*/*n*) + 6129647/(5000000*sqrt(n)).
    - **fbound**(*n*) = [0, 1].
  - Generated using tighter bounds than necessarily proven:

        - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
        - **fabove**(*n*, *k*) = 2337/2500 if *n*<4; otherwise, *f*(*k*/*n*) + 18388941/(20000000*sqrt(n)).
        - **fbound**(*n*) = [0, 1].
- Let *f*(*λ*) = **min(1/8, 3*λ)**. Then, for every integer *n* that's a power of 2, starting from 1:
    - Detected to be concave and Lipschitz continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
        - **fabove**(*n*, *k*) = 4047/5000 if *n*<32; otherwise, *f*(*k*/*n*) + 967839/(250000*sqrt(n)).
        - **fbound**(*n*) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
        - **fabove**(*n*, *k*) = 171/400 if *n*<4; otherwise, *f*(*k*/*n*) + 967839/(1600000*sqrt(n)).
        - **fbound**(*n*) = [0, 1].

8 Notes

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

[^3]: E. Voronovskaya, "Détermination de la forme asymptotique d'approximation des fonctions par les polynômes de M. Bernstein", 1932.

[^4]: G.G. Lorentz, "Approximation of functions", 1966.

[^5]: 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.**

[^6]: Micchelli, Charles. "The saturation class and iterates of the Bernstein polynomials." Journal of Approximation Theory 8, no. 1 (1973): 1-18.

[^7]: Guan, Zhong. "**Iterated Bernstein polynomial approximations**", arXiv:0909.0684 (2009).

[^8]: Güntürk, C.S., Li, W., "**Approximation of functions with one-bit neural networks**", arXiv:2112.09181 [cs.LG], 2021.

[^9]: Güntürk and Li 2021 defines the C^n norm as the maximum absolute value of $f(\lambda)$ and its *n*-th derivative over the domain $[0, 1]$, but the bounds would then be false in general. One counterexample is $2\lambda(1-\lambda)$, and another is $(\sin(\lambda)+2\lambda(1-\lambda))/2$.

[^10]: 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.

[^11]: Carvalho, Luiz Max, and Guido A. Moreira. "**Adaptive truncation of infinite**

sums: applications to Statistics", arXiv:2202.06121 (2022).

[^12]: S. Ray, P.S.V. Nataraj, "A Matrix Method for Efficient Computation of Bernstein Coefficients", Reliable Computing 17(1), 2012.

[^13]: 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.

[^14]: M. Gevrey, "Sur la nature analytique des solutions des équations aux dérivées partielles", 1918.

[^15]: Henderson, S.G., Glynn, P.W., "Nonexistence of a class of variate generation schemes", *Operations Research Letters* 31 (2003).

[^16]: For this approximation, if *n* were infinity, the method would return 1 with probability 1 and so would not approximate *λ***c*, of course.

[^17]: 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.

[^18]: 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.

[^19]: Holtz, O., Nazarov, F., Peres, Y., "New Coins from Old, Smoothly", *Constructive Approximation* 33 (2011).

[^20]: 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.

[^21]: Li, Zhongkai. "Bernstein polynomials and modulus of continuity." Journal of Approximation Theory 102, no. 1 (2000): 171-174.

[^22]: Skorski, Maciej. "**Handy formulas for binomial moments**", arXiv:2012.06270 (2020).

[^23]: Keane, M. S., and O'Brien, G. L., "A Bernoulli factory", *ACM Transactions on Modeling and Computer Simulation* 4(2), 1994.

[^24]: von Neumann, J., "Various techniques used in connection with random digits", 1951.

[^25]: Peres, Y., "**Iterating von Neumann's procedure for extracting random bits**", Annals of Statistics 1992,20,1, p. 590-597.

[^26]: 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.

[^27]: Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724.

[^28]: S. Pae, "**Binarization Trees and Random Number Generation**", arXiv:1602.06058v2 [cs.DS], 2018.

[^29]: Levy, H., *Stochastic dominance*, 1998.

[^30]: 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** .

[^31]: Gal, S.G., "Calculus of the modulus of continuity for nonconcave functions and applications", *Calcolo* 27 (1990)

[^32]: 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.

[^33]: Anastassiou, G.A., Gal, S.G., *Approximation Theory: Moduli of Continuity and Global Smoothness Preservation*, Birkhäuser, 2012.

[^34]: 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 the second derivative's absolute value, so that formula is useful here.

[^35]: Le Gruyer, Erwan. "Minimal Lipschitz extensions to differentiable functions defined on a Hilbert space." Geometric and Functional Analysis 19, no. 4 (2009): 1101-1118.

[^36]: 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.

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 have a finite sum and where $0\le x\le 1$ (the domain is the interval [0, 1]). Then f is convex and has a maximum at 1.

Proof: By inspection, f is a power series and is non-negative on the positive real line (and thus on the domain [0, 1]). Each of its terms has a maximum at 1 since x^n is an increasing function for every $n\ge 0$, and multiplying that by a non-negative constant doesn't change whether it's increasing. Since all of these terms have a maximum at 1 on its domain, so does their sum.

The derivative of f is— $$f\prime(x) = a_1 x^0 + ... + a_i x^{i-1} + ...,$$ which is still a power series with non-negative coefficients, so the proof so far applies to $f\prime$ 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 non-negative on the positive real line, 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 non-negative 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)[^11] — 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 [0, 1] to [0, 1] and be continuous and concave. Then $W_{n,2}$ and $W_{n,3}$ (as defined in "Approximate Bernoulli Factories for Certain Functions") are non-negative on [0, 1].

Proof: For $W_{n,2}$ it's enough to prove that $B_n(f)\le f$ for every $n\le 1$. 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 non-negative. For this, using only the fact that f maps [0, 1] to [0, 1], $B_n(f)$ will have Bernstein coefficients in [0, 1] (each coefficient is a value of f) and so will likewise map [0, 1] to [0, 1] (Qian et al. 2011)[^20]. Thus, by induction, $B_n(B_n(f(\lambda)))$ is non-negative. The discussion for $W_{n,2}$ also shows that $(f - B_n(f))$ is non-negative as well. Thus, $W_{n,3}$ is non-negative on [0, 1]. □

Proposition B2: Let $f(\lambda)$ map [0, 1] to [0, 1], be continuous, nondecreasing, and subadditive, and equal 0 at 0. Then $W_{n,2}$ is non-negative on [0, 1].

Proof: The assumptions on f imply that $B_n(f)\le 2 f$ (Li 2000)[^21], showing that $W_{n,2}$ is non-negative on [0, 1]. □

> **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 [0, 1] to [0, 1] and have a Lipschitz continuous derivative on [0, 1] 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 non-negative there, for every $n\ge m$.

Proof: Let $E(\lambda, n) = \frac{L \lambda(1-\lambda)}{2n}$. Lorentz (1966)[^4] 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 $W_{n,2}=2 f(\lambda) - B_n(\lambda)= 2 f(\lambda) - f(\lambda) - E(\lambda, m) = f(\lambda) - E(\lambda, m)\ge 0$ 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.

Result B4 (Molteni 2022)[^35]: 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 non-negative whenever r is odd and $0\le p \le 1/2$ (Skorski 2020)[^22], 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 for every $n\ge 1$. □

Corollary B6: For every integer $n_0\ge 1$, $|T(n, 3, p)| \le \frac{\sqrt{3}}{18\sqrt{n_0}} n^{3/2} < (963/1000)\frac{1}{\sqrt{n_0}} n^{3/2}$ 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 $|T(n,5,p)| = n/4+2 {n \choose 2} = n/4+2\frac{n!}{(n-2)!} = n^2 - \frac{3}{4}n \le n^2$ 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. □

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 [0, 1] 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)[^4]. Thus, for every $n\ge 1$:

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

Where k is an integer in the interval [0, 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 [0, 1]. 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 interval [0, 1] 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* interval [0, 1] or a subset of it, and that maps its domain to [0, 1]. 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)[^23].

If $f(\lambda)$ meets these sufficient conditions, it admits a Bernoulli factory:

- $f(\lambda)$ is continuous on the closed interval [0, 1].
- $f(\lambda)$ has a minimum of greater than 0 and a maximum of less than 1.

If $f(\lambda)$ meets these sufficient conditions, it admits a Bernoulli factory and is Hölder continuous (has no slope steeper than an n^{th} root's):

- $f(\lambda)$ is continuous.

- $f(\lambda)$ maps the closed interval [0, 1] to [0, 1].
- $f(\lambda)$ equals neither 0 nor 1 on the open interval (0, 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)[^23]. 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 interval [0, 1] or a subset of it to [0, 1], 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 $f(\lambda)$ 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)[^17].

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)[^24]), 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 $f(\lambda)$ is described in the strong simulability statement, and if either f is constant on its domain or f meets the additional conditions below, then f is strongly simulable.*

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

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

- *There is a polynomial $g(\lambda)$ in Bernstein form whose coefficients are computable and in the interval [0, 1], such that $g(0) = f(0)$ and $g(1) = f(1)$ whenever 0 or 1, respectively, is in the domain of f, and such that $g(\lambda) > f(\lambda)$ for every λ in the domain of f, except at 0 and 1.*

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

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

   By the conditions in the lemma, both f and g will be positive on the open interval (0, 1) (wherever f is defined).

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

3. If h equals 1 everywhere in the interval (0, 1) (wherever f is defined), return 1.

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

1. *There are two polynomials g(λ) and ω(λ) in Bernstein form, such that both polynomials' coefficients are computable and all in the interval [0, 1].*
2. *g(0) = ω(0) = f(0) = 0 (so that 0 is in the domain of f).*
3. *g(1) = ω(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(λ) > f(λ).*
5. *For every λ in the domain of f, except at 0 and 1, ω(λ) < f(λ).*

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 *ω*[*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) = *ω*(0) = 0.
3. If 1 is in the domain of *f* and if *j* is *n*, return *g*(1) = *ω*(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 *ω*[*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*(*λ*) = (*f*(*λ*) − *ω*(*λ*)) / (*g*(*λ*) − *ω*(*λ*)). By the conditions in the lemma, *h* will be positive everywhere in the interval (0, 1), wherever *f* is defined.

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

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

Proof of Proposition 2: The following cases can occur:

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

5. $f(0) = f(1) = 0$: Apply Lemma 1.
6. $f(0) = f(1) = 1$: Apply Lemma 1, but take $f = 1 - f$ and return 1 minus the output of the lemma's algorithm.
7. $f(0) = 0$ and $f(1) = 1$: Apply Lemma 2.
8. $f(0) = 1$ and $f(1) = 0$: Apply Lemma 2, but take $f = 1 - f$ and return 1 minus the output of the lemma's algorithm.

□

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

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

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

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

To build g, take its degree as ceil(M)+1 or greater (so that g's Lipschitz constant is greater than M and g has ceil(M) + 2 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(λ) is described in the strong simulability statement, is Lipschitz continuous, and is such that f(0) = 0 and f(1) = 1 (so that 0 and 1 are in the domain of f), then f is strongly simulable.*

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

To build g and ω, take their degree as ceil(M)+1 or greater (so that their Lipschitz constant is greater than M and each polynomial has ceil(M) + 2 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. □

It is suspected that the conditions in Proposition 2 are necessary and sufficient for $f(\lambda)$ to be strongly simulable.

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 J be a closed interval on (0, 1), such as [1/100, 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 can be any value in J, is there a multiple-output Bernoulli factory for f(λ) 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)[^1].)

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

- The functions λ and $1 - \lambda$.
- Constants in [0, 1]. As Nacu and Peres (2005)[^1] already showed, any such constant c admits an optimal factory: generate unbiased random bits using Peres's iterated von Neumann extractor (Peres 1992)[^25], then build a binary tree that generates 1 with probability c and 0 otherwise (Knuth and Yao 1976)[^26].

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

Also, as Yuval Peres (Jun. 24, 2021) told me, there is an efficient multiple-output Bernoulli factory for $f(\lambda) = \lambda/2$: the key is to flip the input coin enough times to produce unbiased random bits using his extractor (Peres 1992)[^18], then multiply each unbiased bit with another input coin flip to get a sample from $\lambda/2$. Given that the sample is equal to 0, there are three possibilities that can "be extracted to produce more fair bits": either the unbiased bit is 0, or the coin flip is 0, or both are 0.

This algorithm, though, doesn't count as an *optimal factory*, and Peres described this algorithm only incompletely. By simulation and trial and error I found an improved version of the algorithm. It uses two randomness extractors (extractor 1 and extractor 2) that produce unbiased random bits from biased data (which is done using a method given later in this section). The extractors must be asymptotically optimal (they must approach the entropy limit as closely as desired); one example is the iterated von Neumann construction in Peres (1992)[^25]. 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 rational function of the form $D(\lambda)/E(\lambda)$, where—

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

The algorithm is a modified version of the "block simulation" in Mossel and Peres (2005, Proposition 2.5)[^27], 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 $P1 = \lambda^r*(1-\lambda)^r*D(\lambda)$.
- outputting either 0 or 1 in that step would be $P01 = \lambda^r*(1-\lambda)^r*E(\lambda)$,

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

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 the interval J to a small range of values, so that the algorithm could, for example, generate a uniform random variate in [0, 1] using unbiased random bits and see whether it 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)[^28]. 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 interval [0, 1], 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 bounded from below by min(λ^n, $(1-\lambda)^n$) for some integer n (Keane and O'Brien 1994)[^23].
- A *modulus of continuity* of a function f means a non-negative and nondecreasing function ω on the interval [0, 1], for which $\omega(0) = 0$, and for which abs(f(x) − f(y)) ≤ ω(abs($x-y$)) for every x in [0, 1] and every y in [0, 1]. Loosely speaking, a modulus of continuity $\omega(\delta)$ is bounded below by f's maximum range in a window of size δ.

Lemma 1. *Let f(λ) be a continuous and nondecreasing 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) is nondecreasing 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)[^29]. This means that the probability that ($X_{m+1} \le j$) is less than or equal to that for X_m for each j in the interval [0, n]. A proof of this was given by the user "Henry" of the *Mathematics Stack Exchange* community[^30]. □

Lemma 6(i) of Nacu and Peres (2005)[^1] 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 [0, 1] to [0, 1], 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 on [0, 1], then the expression—*
   *abs(**E**[f(X/n)] − f(k/(2*n))),   (1)*
   is bounded from above by—
    - *ω(sqrt(1/(8*n−4))), for every integer n≥1 that's a power of 2,*
    - *ω(sqrt(1/(7*n))), for every integer n≥4 that's a power of 2,*
    - *ω(sqrt(1/(2*n))), for every integer n≥1 that's a power of 2, and*
    - *ω(sqrt((k/(2*n)) * (1−k/(2*n)) / (2*n−1))), for every n≥1 that's a power of 2.*
2. *If f is Hölder continuous with Hölder constant M and with Hölder exponent α in the interval (0, 1], then the expression (1) is bounded from above by—*

- *$M*(1/(2*n))^{\alpha/2}$, for every integer n≥1 that's a power of 2,*
- *$M*(1/(7*n))^{\alpha/2}$, for every integer n≥4 that's a power of 2, and*
- *$M*(1/(8*n-4))^{\alpha/2}$, for every integer n≥1 that's a power of 2.*

3. *If f has a second derivative whose absolute value is defined in all of [0, 1] and bounded from above by M, then the expression (1) is bounded from above by—*
   - *(M/2)*(1/(7*n)), for every integer n≥4 that's a power of 2, and*
   - *(M/2)*(1/(8*n−4)), for every integer n≥1 that's a power of 2.*
4. *If f is convex, nondecreasing, and bounded from below by 0, then the expression (1) is bounded from above by **E**[f(Y/n)] for every integer n≥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 non-negative because absolute values are non-negative. To prove the first and second bounds: abs(**E**[$f(X/n)$] − $f(k/(2 * n))$) ≤ **E**[abs($f(X/n) - f(k/(2 * n))$] ≤ **E**[ω(abs($X/n - k/(2 * n)$)] ≤ ω(**E**[abs($X/n - k/(2 * n)$)]) (by Jensen's inequality and because ω is concave) ≤ ω(sqrt(**E**[abs($X/n - k/(2 * n)$)]^2)) = ω(sqrt(**Var**[X/n])) = ω(sqrt((k*(2 * $n-k$)/(4*(2 * $n-1$)*n^2)))) ≤ ω(sqrt((n^2/(4*(2 * $n-1$)*n^2)))) = ω(sqrt((1/(8*$n-4$)))) = ρ, and for every n≥4 that's an integer power of 2, ρ ≤ ω(sqrt(1/(7*n))). To prove the third bound: abs(**E**[$f(X/n)$] − $f(k/(2 * n))$) ≤ ω(sqrt(**Var**[X/n])) ≤ ω(sqrt(1/(2*n))). To prove the fourth bound: abs(**E**[$f(X/n)$] − $f(k/(2 * n))$) ≤ ω(sqrt((n^2/(4*(2 * $n-1$)*n^2)))) = ω(sqrt((k/(2*n)) * (1−k/(2*n)) / (2*$n-1$))).
2. By the definition of Hölder continuous functions, take $\omega(x) = M*x^\alpha$. Because ω is a concave modulus of continuity on [0,1], the result follows from part 1.
3. abs(**E**[$f(X/n)$] − $f(k/(2 * n))$) ≤ (M/2)***Var**[X/n] = (M/2)*(k*(2 * $n-k$)/(4*(2 * $n-1$)*n^2)) ≤ (M/2)*(n^2/(4*(2 * $n-1$)*n^2)) = (M/2)*(1/(8*$n-4$)) = ρ. For every integer n≥4 that's a power of 2, ρ ≤ (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 nondecreasing, **E**[$f(X_k/n)$] is nondecreasing as k increases, so take **E**[$f(X_n/n)$] = **E**[$f(Y/n)$] as the upper bound. Then, abs(**E**[$f(X/n)$] − $f(k/(2 * n))$) = abs(**E**[$f(X/n)$] − f(**E**[X/n])) = **E**[$f(X/n)$] − f(**E**[X/n]) (by Jensen's inequality, because f is convex and bounded by 0) = **E**[$f(X/n)$] − $f(k/(2 * n))$ ≤ **E**[$f(X/n)$] (because f is bounded by 0) ≤ **E**[$f(Y/n)$]. □

   Notes:

   1. **E**[.] means expected value ("long-run average"), and **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 and 2 exploit a tighter bound on **Var**[X/n] than the bound given in Nacu and Peres (2005, Lemma 6(i) and 6(ii), respectively)[^1]. However, for technical reasons, different bounds are proved for different ranges of integers n.
   3. For part 3, as in Lemma 6(ii) of Nacu and Peres 2005, the second derivative need not be continuous (Y. Peres, pers. comm., 2021).
   4. All continuous functions that map the closed interval [0, 1] to [0, 1], 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)[^31]; (Gal 1995)[^32].
2. If f is strictly decreasing and convex, $\omega(x)$ can equal $f(0) - f(x)$ (Gal 1990)[^31]; (Gal 1995)[^32].
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)[^33].

Theorem 1. *Let ω(x) be as described in part 1 of Lemma 2, and let f(λ) be a strictly bounded factory function. Let—*

$\eta(n) = \eta(2^m) = \sum_{k=m, m+1,...} \varphi(2^k)$,

for every integer n≥1 that's a power of 2 (with $n=2^m$), where φ(n) is either—

- *ω(sqrt(1/(8*n−4))), or*
- *ω(sqrt(1/(2*n))).*

*If the infinite series η(n) converges, then the following scheme for f(λ) is valid in the following sense: By forming two sequences of polynomials in Bernstein form with coefficients **fabove**(n, k) for the upper polynomials, and **fbelow**(n, k) for the lower polynomials, then those polynomials meet conditions (i), (iii), and (iv) of Proposition 3 of Nacu and Peres (2005)[^1], for every integer n≥1 that's a power of 2, by defining **fabove** and **fbelow** as follows:*

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

*Except that the following bounding note applies: If **fabove**(n, k) > 1 for a given n and some k, **fabove**(n, k) = 1 instead for that n, and if **fbelow**(n, k) < 0 for a given n and some k, **fbelow**(n, k) = 0 instead for that n.*

Proof. Follows from part 1 of Lemma 2 above as well as Remark B and the proof of Proposition 10 of Nacu and Peres (2005)[^1].

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

Condition (i) says that the coefficients **fbelow** and **fabove** must be bounded by 0 and 1. This is ensured starting with a large enough value of n greater than 0, call it n_0, 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 converges, $\eta(n)$ will eventually stay less than ϵ. And the `f(k/n)` term is bounded by the minimum and maximum of f by construction. This combined means that the coefficients **fbelow** and **fabove** will eventually be bounded by 0 and 1 for every integer n starting with n_0.

For n less than n_0, condition (i) is ensured by setting **fbelow** and **fabove** to 0 or 1, respectively, whenever a coefficient of the degree-n polynomial would otherwise be outside the bounds.

Condition (iii) says that the upper polynomials must converge to f and so must the lower polynomials. This is ensured in a similar way as in Proposition 10, as well as by the assumption that the series converges: as n goes to infinity, $\eta(n)$ goes to 0 so that the coefficients, and thus the polynomials, converge to f. For n less than n_0, the values of **fbelow** and **fabove** can be 0 or 1 without affecting convergence.

Condition (iv) is the *consistency requirement* described earlier in this page. This is ensured as in Proposition 10 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 φ can take on either form given in the theorem. The solution given in 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, 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. For n_0, the consistency requirement is maintained by noting that the degree-n_0 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_0, and those polynomials are the constant 0 and the constant 1, respectively, as are their coefficients. □

> **Note:** 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 $0 = \eta(n) - \eta(4 * n) - \varphi(2*n) - \varphi(n) = \eta(n) - \eta(8 * n) - \varphi(4*n) - \varphi(2*n) - \varphi(n) = ...$

Corollary 1. *Let f(λ) be a strictly bounded factory function. If f is Hölder continuous with Hölder constant M and with Hölder exponent α in the interval (0, 1], then the following scheme determined by* ***fbelow*** *and* ***fabove*** *is valid in the sense of Theorem 1, subject to the bounding note:*

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

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

Proof. Follows from Theorem 1 by using the ω given in part 2 of Lemma 2, and by using $\varphi(n) = \omega(\text{sqrt}(1/(2*n)))$. □

> **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: $(M* \sum_{i = 0,...,(j*2)-1} 2^{i/(2*j)})/n^{1/(2*j)} = M / ((2^{1/(2*j)} -1)*n^{1/(2*j)})$; and generalizing the latter expression led to the term in the theorem.

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

- ***fbelow***$(n, k) = f(k/n) - M/((\text{sqrt}(2)-1)*\text{sqrt}(n))$.
- ***fabove***$(n, k) = f(k/n) + M/((\text{sqrt}(2)-1)*\text{sqrt}(n))$.

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

> **Note:** This special case of Theorem 1 was already found by Nacu and Peres (2005)[^1].

Theorem 2. *Let $f(\lambda)$ be a strictly bounded factory function, and let $\omega(x)$ be as described in Theorem 1. Theorem 1 remains valid with the following versions of $\varphi(n)$,* ***fbelow****, and* ***fabove****, rather than as given in that theorem, subject to the bounding note:*

- $\varphi(n) = \omega(\text{sqrt}(1/(7*n)))$.
- ***fbelow***$(n, k) = \min($***fbelow***$(4,0)$, ***fbelow***$(4,1)$, ..., ***fbelow***$(4,4))$ *if* $n < 4$*; otherwise,* $f(k/n) - \eta(n)$.
- ***fabove***$(n, k) = \max($***fabove***$(4,0)$, ***fabove***$(4,1)$, ..., ***fabove***$(4,4))$ *if* $n < 4$*; otherwise,* $f(k/n) + \eta(n)$.

Proof. Follows from Theorem 1 and part 1 of Lemma 2 above, as well as Remark B and the proof of Proposition 10 of Nacu and Peres, including the observation in Remark B of the paper that we can start the algorithm from $n = 4$; in that case, the upper and lower polynomials of degree 1 through 3 above would be constant functions, so that as polynomials in Bernstein form, the coefficients of each one would be equal. 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. □

Corollary 3. *Let $f(\lambda)$ be a strictly bounded factory function. If f is Hölder continuous with Hölder constant M and with Hölder exponent α in the interval (0, 1], then the following scheme is valid in the sense of Theorem 1, subject to the bounding note:*

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

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

Proof. Follows from Theorem 2 by using the ω given in part 2 of Lemma 2. □

Theorem 3. *Let $f(\lambda)$ be a strictly bounded factory function. If f has a second derivative whose absolute value is defined in all of [0, 1] and bounded from above by M, then the following scheme is valid in the sense of Theorem 1, subject to the bounding note:*

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

Proof. Because $(M/2)*(1/(7*n))$ in part 3 of Lemma 2 is bounded the same way as the statement $\omega(\text{sqrt}(1/(7*n)))$ in Theorem 2 and part 1 of Lemma 2, take $\omega(n)$ as $(M/2)*(1/(7*n))$. Then the result follows from Theorem 2. □

Theorem 4. *Let $f(\lambda)$ be a strictly bounded factory function. If f is convex and nondecreasing, then Theorem 1 remains valid with $\varphi(n) = \mathbf{E}[f(Y/n)]$ (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 any of Theorems 1 through 4, except that f must be concave and polynomially bounded and may have a minimum of 0. Then the scheme of that theorem remains valid if* ***fbelow****(n, k) = f(k/n), rather than as given in that theorem.*

2. *Let f be as given in any of Theorems 1 through 4, except that f must be convex and polynomially bounded and may have a maximum of 1. Then the scheme of that theorem remains valid if* ***fabove****(n, k) = f(k/n), rather than as given in that theorem.*

3. *Theorems 1 through 4 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 Theorems 1 through 4, 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)[^1]. □

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. *Suppose that for each integer $n\ge 1$ and for a strictly bounded factory function $f(\lambda)$:*

- ***fbelow***(n, k) *is the k-th Bernstein coefficient of the polynomial g_n.*
- ***fabove***(n,k) *is the k-th Bernstein coefficient of the polynomial h_n.*
- **fbelow**(n,k) ≤ **fabove**(n,k).
- *The polynomials g_n and h_n satisfy conditions (iii) and (iv) of Nacu and Peres (2005, Proposition 3)[^1] (discussed in the proof of Theorem 1 above).*

Suppose ***fbelow****(n,k) = $W_n(k/n) - \epsilon_n(f)$ and* ***fabove****(n,k) = $W_n(k/n) + \epsilon_n(f)$ for each n in a set of positive integers. Then for each n in that set, $|f(\lambda) - B_n(W_n(\lambda))| \le \epsilon_n(f)$ 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, condition (iv) of Nacu and Peres (2005)[^1] implies that $g_{n}(\lambda)\le g_{n+1}(\lambda)\le f(\lambda)$ (the lower polynomials "increase") and $h_{n}(\lambda)\ge h_{n+1}\ge f(\lambda)$ (the upper polynomials "decrease") for every $n\ge 1$ (Nacu and Peres 2005, Remark A)[^1].

Then if $B_n(W_n(\lambda)) < f(\lambda)$ for some λ in [0, 1], shifting the left-hand side upward by $\epsilon_n(f)$ 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 [0, 1], shifting the left-hand side downward by $\epsilon_n(f)$ means that $g_n = B_n(W_n(\lambda))-\epsilon_n(f) \ge f(\lambda)$, 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 the scheme given in Theorem 1, then $B_n(f(\lambda))$ comes within $\eta(n)$ of f for every integer $n\ge 1$ that's a power of 2; that is, $|B_n(f(\lambda))| \le \eta(n)$ for every such n.*

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 f be a strictly bounded factory function.

For each $n\ge 1$, let g_n and h_n be polynomials in Bernstein form; let **fbelow**(n, k) be the k-th Bernstein coefficient of the polynomial g_n; and let **fabove**(n,k) is the k-th Bernstein coefficient of the polynomial h_n.

Denote the Bernstein polynomial of degree n of a function f as $B_n(f)$.

For each integer $n\ge1$ that's a power of 2, let $W_n(\lambda)$ be a function with a known Bernstein polynomial. (W_n can but need not equal f.)

For each such n, suppose that there is $D>0$ such that— $$|f(\lambda)-B_n(W_n(\lambda))| \le DM/n^{r/2},$$ whenever $0\le \lambda\le 1$, f has $r\ge 1$ continuous derivatives, and M is the maximum absolute value of f and its derivatives up to the r-th derivative.

Then there is C_0 greater than 0 such that for every $C\ge C_0$, if—

- **fbelow**$(n,k)=W_n(k/n) - CM/n^{r/2}$, and
- **fabove**$(n,k)=W_n(k/n) + CM/n^{r/2}$,

for each integer $n\ge1$ that's a power of 2, and if $g_n = g_{2^{floor(log_2(n))}}$ and $h_n = h_{2^{floor(log_2(n))}}$ for each other integer $n\gt 1$, then the polynomials g_n and h_n satisfy conditions (iii) and (iv) of Nacu and Peres (2005, Proposition 3) [^1] (discussed in the proof of Theorem 1 above).

It is further conjectured that the same value of C_0 suffices when f has a Lipschitz continuous $(r-1)$-th derivative and M is the maximum absolute value 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 $C_0 M/n^{r/2}$.

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

- M_0 is the maximum absolute value of $f(\lambda)$ and its r-th derivative (Güntürk and Li 2021)[^8].
- M_1 is the maximum absolute value of $f(\lambda)$ and its derivatives up to the r-th derivative. Thus, $M_1\ge M_0$.
- 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:	Because of this counterexample:
3	$M=M_0$	$2 f - B_n(f)$*	2.62	2.63	$2 \lambda \left(1 - \lambda\right)$
3	$M=M_0$, $n\ge 4$	$2 f - B_n(f)$	0.66	0.67	$2\lambda(1-\lambda)$
4	$M=M_0$	$2 f - B_n(f)$	3.58	3.59	$3 \lambda^{2} \cdot \left(1 - \lambda\right)$
4	$M=M_0$, $n\ge 4$	$2 f - B_n(f)$	3.52	3.53	$\lambda^{2} \cdot \left(1 - \lambda\right)$
3	$M=M_1$	$2 f - B_n(f)$	0.29	$\frac{3}{16-4 \sqrt{2}}$ < 0.29005.**	$2 \lambda \left(1 - \lambda\right)$
3	$M=M_1$, $n\ge 4$	$2 f - B_n(f)$	0.08	0.09	$2 \lambda \left(1 - \lambda\right)$
4	$M=M_1$	$2 f - B_n(f)$	0.24	0.25	$2 \lambda \left(1 - \lambda\right)$
4	$M=M_1$, $n\ge 4$	$2 f - B_n(f)$	0.14	0.15	$2 \lambda \left(1 - \lambda\right)$
5	$M=M_1$	$B_n(B_n(f))+3(f-B_n(f))$***	0.26	0.27	$2 \lambda \left(1 - \lambda\right)$
5	$M=M_1$, $n\ge 4$	$B_n(B_n(f))+3(f-B_n(f))$	0.10	0.11	$3 \lambda^{2} \cdot \left(1 - \lambda\right)$
6	$M=M_1$	$B_n(B_n(f))+3(f-B_n(f))$	0.24	0.25	$2 \lambda \left(1 - \lambda\right)$
6	$M=M_1$, $n\ge 4$	$B_n(B_n(f))+3(f-B_n(f))$	0.22	0.23	$3 \lambda^{2} \cdot \left(1 - \lambda\right)$

* Corresponds to the iterated Bernstein polynomial of order 2.

*** Corresponds to the iterated Bernstein polynomial of order 3.

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

Suppose $f(0) = f(1) = \beta$, then $W_1(f)$ will equal β. 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 β. 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 $|\beta - (2f(1/2) - B_2(f)(1/2))| = |3\beta/2 - 3f(1/2)/2|$. That right-hand side is the minimum shift needed for the consistency requirement to hold; call it z, and let y be M_1 for $r=3$. 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— $$C_0=\frac{z}{y} \frac{4}{4-\sqrt{2}} = \frac{|\beta-f(1/2)|}{y} \frac{12}{2\cdot (4-\sqrt{2})}.$$

The solution shows that if $y = M_1$ 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_1 for $r=3$ such that:

1. $f(\lambda)$ has at least three continuous derivatives.
2. $f(0)=f(1)$ 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_1 of 4. Given that $\frac{|\beta-g(1/2)|}{y}=\frac{|0-1/2|}{y}=1/8$, the goal is now to see whether any function f satisfying (1) and (2) has $\max(\beta, f(1/2)) < M_1 < 8\cdot |\beta-f(1/2)|$.

To aid in this goal, there is a formula to find the least possible Lipschitz constant for f's first derivative (see "Definitions")[^34], given a finite set of points (0, 1/2, and 1 in the case at hand) and the values of f and $f\prime$ at those points (Le Gruyer 2009)[^35]; see also (Herbert-Voss et al. 2017)[^36]. Denote $L(.,.,.)$ as this least possible Lipschitz constant. Then according to that formula— $$L([0, 1/2, 1], [\beta, \beta+t, \beta], [z_1, z_2, z_3]) = \max(2 \sqrt{\left|{z_{1} - z_{2}}\right|^{2} + \left|{- 4 t + z_{1} + z_{2}}\right|^{2}} + 2 \left|{- 4 t + z_{1} + z_{2}}\right|,$$ $$\sqrt{\left|{z_{1} - z_{3}}\right|^{2} + \left|{z_{1} + z_{3}}\right|^{2}} + \left|{z_{1} + z_{3}}\right|,$$ $$2 \sqrt{\left|{z_{2} - z_{3}}\right|^{2} + \left|{4 t + z_{2} + z_{3}}\right|^{2}} + 2 \left|{4 t + z_{2} + z_{3}}\right|)$$ $$=L([0, 1/2, 1], [0, t, 0], [z_1, z_2, z_3])$$ (where t is greater than 0 and less than 1), so L no longer depends on β; thus β can equal 0 without loss of generality. Moreover, only $f\prime$ 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 $[0, 1]$ have to be checked and only $f\prime$ values in $[-H, H]$ have to be checked.

Assuming that no M_1 less than $8\cdot |\beta-f(1/2)|$ is found, then— $$C_0=\frac{|\beta-f(1/2)|}{y} \frac{12}{2\cdot (4-\sqrt{2})}=\frac{|\beta-f(1/2)|}{H} \frac{12}{2\cdot (4-\sqrt{2})}=(1/8) \frac{12}{2\cdot (4-\sqrt{2})}$$ $$=3/(16-4\sqrt{2}),$$ 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 $n \geq 0$):

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

Now, by applying Theorem 1, we compute $\eta(k)$ by substituting n with 2^n, summing over $[k, \infty)$, and substituting k with $\log2(k)$. η converges, resulting in:

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

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

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

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

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

<h1>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()
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,i)*10000000).n()/10000000))
  except:
    print("eta(2^%d) ~= [FAILED]" % (i))
```

10 License