# Supplemental Notes for Bernoulli Factory Algorithms

This version of the document is dated 2021-08-17.

**Peter Occil**

# 1 Contents



# 2 General Factory Functions

As a reminder, the *Bernoulli factory problem* is: Given a coin with unknown probability of heads of $\lambda$, sample the probability $f(\lambda)$.

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

1. Generate a uniform(0, 1) random variate, $U$.
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: one approaches the function $f(\lambda)$ from above, the other from below, where $f$ is a function for which the Bernoulli factory problem can be solved. (These two sequences form a so-called *approximation scheme* for $f$.) One requirement for these algorithms to work correctly is called the *consistency requirement*:

*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 elevated to degree n and rewritten in Bernstein form.*

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 mean that the scheme will ensure consistency. Examples of this fact are shown in the section "**Schemes That Don't Work**" later in this document.

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

## 2.1 Approximation Schemes

A *factory function* $f(\lambda)$ is a function for which the Bernoulli factory problem can be solved (see "**About Bernoulli Factories**"). The following are approximation schemes for $f$ if it 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 known to be *concave* in the interval [0, 1] (which roughly means that its rate of growth there never goes up), then **fbelow**($n$, $k$) can equal $f(k/n)$, thanks to Jensen's inequality.

**Convex functions.** If $f$ is known to be *convex* in the interval [0, 1] (which roughly means that its rate of growth there never goes down), 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 defined "slope-of-slope" everywhere in [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($f''(x)$) for any $x$ in [0, 1], where $f''$ is the "slope-of-slope" function 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 "slope-of-slope" function need not be continuous (Y. Peres, pers. comm., 2021).

**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 $\alpha$-***Hölder continuous*** in [0, 1], meaning its vertical slopes there, if any, are no "steeper" than that of $m*\lambda^{\alpha}$, for some number $m$ greater than 0 (the Hölder constant) and for some $\alpha$ in the interval (0, 1], 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.

If $f$ in [0, 1] has a defined slope at all points or "almost everywhere"[2], and does not tend to a vertical slope anywhere, then $f$ is ***Lipschitz continuous***, $\alpha$ is 1, and $m$ is the highest absolute value of the function's "slope". Otherwise, finding $m$ for a given $\alpha$ is non-trivial and it requires knowing where $f$'s vertical slopes are, among other things.[3] But assuming $m$ and $\alpha$ 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 $\alpha$-Hölder continuous for any $\alpha$ greater than 0. These functions have a slope that's steeper than any "nth" root, and can't be handled by this method. One example is $f(\lambda) = 1/10$ if $\lambda$ is 0 and $-1/(2*\ln(\lambda/2)) + 1/10$ otherwise, which has a slope near 0 that's steeper than any "nth" root.
  2. In the Lipschitz case ($\alpha = 1$), $D(n)$ can be $m*322613/(250000*\text{sqrt}(n))$, which is an upper bound.
  3. In the case $\alpha = 1/2$, $D(n)$ 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 an approximation 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 $\lambda$ is 0.)
- $1 \ge h(1) \ge f(1) \ge 0$.
- $1 > h(\lambda) > f(\lambda) > 0$ for every $\lambda$ in the open interval (0, 1).
- If $f(1) = 0$, then $h(1) = 0$. (This is required to ensure correctness in case $\lambda$ 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 $g(\lambda) = \lim_{\nu \to \lambda} f(\nu)/h(\nu)$ (in other words, the value that $f(\nu)/h(\nu)$ approaches as $\nu$ approaches $\lambda$.) 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 one of the **general factory function algorithms** for $g(.)$, and return the result of that algorithm. This involves building polynomials that converge to $g(.)$, as described earlier in this section. (Alternatively, if $g$ is easy to simulate, instead run another Bernoulli factory algorithm for $g$ and return the result of that algorithm.)

   **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 monotonically increasing,
      - $h(\lambda) = \lambda$, and
      - $f'(\lambda)$, the "slope" function 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(0, 1) random variate $u$ 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)**(4)** (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 SymPy code 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. Since $g(0) = 0$, we 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, so we have 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 involves 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 $\lambda$ 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, $\omega$ is a function that bounds $f$ from below, just as $h$ bounds $f$ from above. $\omega$ should be a function with a simple Bernoulli factory algorithm, such as a polynomial in Bernstein form. If both $\omega$ 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] 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 $q(.)$. If the call returns 1, return 1.
3. Run one of the **general factory function algorithms** for $r(.)$. If the call returns 0, return 1. Otherwise, return 0. This step involves building polynomials that converge to $r(.)$, as described earlier in this section.

   **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 SymPy code: `fx=(1-exp(x))/(1-exp(1)); h=x; phi=x**2; q=(phi/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 an approximation scheme under the convex, concave, twice differentiable, and Hölder classes:

| If $f(0)$ = | And $f(1)$ = | Method |
|---|---|---|
| $> 0$ and $< 1$ | | Use the algorithm for **certain functions that equal 0 at 0**, but with $f(\lambda) = 1 - f(1-\lambda)$.<br>*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."<br>*Inverted result:* If the overall algorithm would return 0, it returns 1 instead, and vice versa. |
| $> 0$ | and $< 0$ | Algorithm for **certain functions that equal 0 at 0**, but with $f(\lambda) = f(1-\lambda)$. (For example, $\cosh(\lambda)-1$ becomes $\cosh(1-\lambda)-1$.) |

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

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

**Open questions.**

- Are there factory functions used in practice that are not covered by the approximation schemes in this section?
- Are there specific functions (especially those in practical use) for which there are practical and faster formulas for building polynomials that converge to those functions (besides those I list in this section or the main **Bernoulli Factory Algorithms** article)?

## 2.2 Schemes That Don't Work

In the academic literature (papers and books), there are many approximation schemes that involve polynomials that converge from above and below to a function. Unfortunately, most of them cannot be used as is to simulate a function $f$ in the Bernoulli Factory setting, because they don't ensure the consistency requirement described earlier.

The following are approximation schemes with counterexamples to consistency.

**First scheme.** In this scheme (Powell 1981)**(5)**, let $f$ be a $C^2$ continuous function (a function with continuous "slope" and "slope-of-slope" functions) in [0, 1]. Then for every $n \geq 1$:

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

Where $M$ is not less than the maximum absolute value of $f$'s slope-of-slope function (second derivative), and where $k$ is an integer in the interval $[0, n]$.

The counterexample involves the $C^2$ continuous function $g(\lambda) = \sin(\pi*\lambda)/4 + 1/2$.

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 we can see, the elevated polynomial's coefficient 0.8208... is less than the corresponding coefficient 0.8271... for the degree-4 polynomial.

*The rest of this section will note counterexamples involving other functions and schemes, without demonstrating them in detail.*

**Second scheme.** In this scheme, let $f$ be a Lipschitz continuous function in [0, 1] (that is, a continuous function in [0, 1] that has a defined slope at all points or "almost everywhere", and does not tend to a vertical slope anywhere). Then for every $n \geq 2$:

- **fabove**($n$, $k$) = $f(k/n) + L^*(5/4) / \text{sqrt}(n)$.
- **fbelow**($n$, $k$) = $f(k/n) - L^*(5/4) / \text{sqrt}(n)$.

Where L is the maximum absolute "slope", also known as the Lipschitz constant, and (5/4) is the so-called Popoviciu constant, and where $k$ is an integer in the interval [0, $n$] (Lorentz 1986)**(6)**, (Popoviciu 1935)**(7)**.

There are two counterexamples here; together they show that this scheme can fail to ensure consistency, even if the set of functions is restricted to "smooth" functions (not just Lipschitz continuous functions):

1. The function $f(\lambda) = \min(\lambda, 1-\lambda)/2$ is Lipschitz continuous with Lipschitz constant 1/2. (In addition, $f$ has a kink at 1/2, so that it's not differentiable, but this is not essential for the counterexample.) The counterexample involves the degree-5 and degree-6 upper polynomials (**fabove**(5, $k$) and **fabove**(6, $k$)).
2. The function $f = \sin(4*\pi*\lambda)/4 + 1/2$, a "smooth" function with Lipschitz constant $\pi$. The counterexample involves the degree-3 and degree-4 lower polynomials (**fbelow**(3, $k$) and **fbelow**(4, $k$)).

It is yet to be seen whether a counterexample exists for this scheme when $n$ is restricted to powers of 2.

**Third scheme.** Same as the second scheme, but replacing (5/4) with the Sikkema constant, $S = (4306+837*\text{sqrt}(6))/5832$ (Lorentz 1986)**(6)**, (Sikkema 1961)**(8)**, which is slightly less than 1.09. In fact, the same counterexamples for the second scheme apply to this one, since this scheme merely multiplies the offset to bring the approximating polynomials closer to $f$.

**Note on "clamping".** For any approximation 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 a counterexample that applies to any approximation 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, if we clamp 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 consistent approximation property.

# 3 Approximate Bernoulli Factories

Although the schemes in the previous section don't work when building a *family* of polynomials that converge to a function $f(\lambda)$, they are still useful for building an *approximation* to that function, in the form of a *single* polynomial, so that we get an **approximate Bernoulli factory** for $f$.

Here, $f(\lambda)$ 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($\lambda$, 1 − $\lambda$)). Continuous functions 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:

1. We first find a polynomial in Bernstein form of degree $n$ that is close to the desired function $f$.

   The simplest choice for this polynomial has $n$+1 coefficients and its $j^{\text{th}}$ coefficient (starting at 0) is found as $f(j/n)$, and this is used in the examples below. 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, we use one of the algorithms in the section "**Certain Polynomials**" to simulate that polynomial, given its coefficients.

**Examples of approximate Bernoulli factories.** The schemes in the previous section give an upper bound on the error on approximating $f$ with a degree-$n$ polynomial in Bernstein form. For example, the third scheme does this when $f$ is a Lipschitz continuous function (with Lipschitz constant $L$). To find a degree $n$ such that $f$ is approximated with a maximum error of $\varepsilon$, we need to solve the following equation for $n$:

- $\varepsilon = L*((4306+837*\text{sqrt}(6))/5832) / \text{sqrt}(n)$.

This has the following solution:

- $n = L^2*(3604122*\text{sqrt}(6) + 11372525)/(17006112*\varepsilon^2)$.

This is generally not an integer, so we use $n = \text{ceil}(n)$ to get the solution if it's an integer, or the nearest integer that's bigger than the solution. This solution can be simplified further to $n = \text{ceil}(59393*L^2/(50000*\varepsilon^2))$, which bounds the previous solution from above.

Now, if $f$ is a Lipschitz continuous factory function with Lipschitz constant $L$, the following algorithm (adapted from "Certain Polynomials") simulates a polynomial that approximates $f$ with a maximum error of $\varepsilon$:

1. Calculate $n$ as $\text{ceil}(59393*L^2/(50000*\varepsilon^2))$.
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.)

As another example, we use the first scheme in the previous section to get the following approximate algorithm for $C^2$ continuous functions with maximum "slope-of-slope" of $M$:

1. Calculate $n$ as ceil($M$/(8*$\varepsilon$)) (upper bound of the solution to the equation $\varepsilon = M/(8*n)$).
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.

We can proceed similarly with other methods that give an upper bound on the Bernstein-form polynomial approximation error, if they apply to the function $f$ that we seek to approximate.

**Approximate methods for linear functions.** There are a number of approximate methods to simulate $\lambda*c$, where $c > 1$ and $\lambda$ 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)[**(9)** approximates the function $\lambda*2$ using a polynomial where the $j^{\text{th}}$ 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 $\lambda*c$**(10)**: "(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 $j^{\text{th}}$ coefficient (starting at 0) is min(($j/n$)*$c$, 1). If $g(\lambda)$ is that polynomial, then the error in approximating $f$ is 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 $\lambda*2$.

  **Note:** Bias and variance are the two sources of error in a randomized estimation algorithm. Let $g(\lambda)$ be an approximation of $f(\lambda)$. The original Bernoulli factory for $f$, if it exists, has bias 0 and variance $f(\lambda)*(1-f(\lambda))$, but the approximate Bernoulli factory has bias $g(\lambda) - f(\lambda)$ and variance $g(\lambda)*(1-g(\lambda))$. Unlike with bias, there are ways to reduce variance, which are outside the scope of this document. An estimation algorithm's *mean squared error* equals variance plus square of bias.

# 4 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:

- $\lambda$, lies in the interval $[\varepsilon, 1-\varepsilon]$ for some $\varepsilon > 0$.
- $\Delta(n, r, \lambda) = O(\max(\text{sqrt}(\lambda*(1-\lambda)/n), 1/n)^r)$, that is, $O((1/n)^r)$ near $\lambda = 0$ or 1, and $O((1/n)^{r/2})$ elsewhere. ($O(h(n))$ roughly means "bounded from above by $h(n)$ times a constant".)

| **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)**(11)**. |
| Requires a finite number of flips on average. Also known as "realizable" by Flajolet et al. (2010)**(4)**. | Only if $f$ is Lipschitz continuous (Nacu and Peres 2005)**(1)**.<br>Whenever $f$ admits a fast simulation (Mendo 2019)**(12)**. |
| Number of flips required, raised to power of $r$, is finite on average and has a tail that drops off uniformly for every $\lambda$. | Only if $f$ is $C^r$ continuous (has $r$ or more continuous derivatives, or "slope" functions) (Nacu and Peres 2005)**(1)**. |
| Requires more than $n$ flips with probability $\Delta(n, r + 1, \lambda)$, for integer $r \geq 0$ and every $\lambda$. (The greater $r$ is, the faster the simulation.) | Only if $f$ is $C^r$ continuous and the $r^{\text{th}}$ derivative is in the Zygmund class (has no vertical slope) (Holtz et al. 2011)**(13)**. |
| Requires more than $n$ flips with probability $\Delta(n, \alpha, \lambda)$, for non-integer $\alpha > 0$ and every $\lambda$. (The greater $\alpha$ is, the faster the simulation.) | If and only if $f$ is $C^r$ continuous and the $r^{\text{th}}$ derivative is $(\alpha - r)$-Hölder continuous, where $r = \text{floor}(\alpha)$ (Holtz et al. 2011)**(13)**. 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)**(4)**. | If and only if $f$ is real analytic (is $C^\infty$ continuous, or has continuous $k^{\text{th}}$ derivative for every $k$, and agrees with its Taylor series "near" every point) (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)**(12)**. |

**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 unless $f$ is $C^2$ continuous.
2. If a function is constant on some open interval in its domain, but is not constant on the whole domain, then it can't be real analytic.

# 5 Complexity

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

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

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

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

Also, on average, half of these flips ($\varphi$) 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 (average) time complexity of many Bernoulli factory algorithms.

# 6 Examples of Bernoulli Factory Approximation Schemes

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

- Let $f(\lambda) = \mathbf{min(1/2, \lambda)}$. 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**$(n, k) = 9563/10000$ if $n<8$; otherwise, $f(k/n) + 322613/(250000*\text{sqrt(n)})$.
    - **fbound**$(n) = [0, 1]$.
  - Generated using tighter bounds than necessarily proven:
    - **fbelow**$(n, k) = f(k/n)$.
    - **fabove**$(n, k) = 371/500$ if $n<4$; otherwise, $f(k/n) + 967839/(2000000*\text{sqrt(n)})$.
    - **fbound**$(n) = [0, 1]$.
- Let $f(\lambda) = \mathbf{\cosh(\lambda) - 1}$. Then simulate $f$ by first flipping the input coin twice. If both flips return 0, return 0. Otherwise, flip the input coin. If it returns 0, return 0. Otherwise, simulate $g(\lambda)$ (a function described below) and return the result.

  Let $g(\lambda) = 1/4$ if $\lambda = 0$; $-(\cosh(\lambda) - 1)/(\lambda^2*(\lambda - 2))$ 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**$(n, k) = 503/2500$ if $n<4$; otherwise, $g(k/n) - 136501/(700000*\text{n})$.
    - **fabove**$(n, k) = g(k/n)$.
    - **fbound**$(n) = [0, 1]$.
  - Generated using tighter bounds than necessarily proven:
    - **fbelow**$(n, k) = 2301/10000$ if $n<4$; otherwise, $g(k/n) -$

1774513/(22400000*n).
        - **fabove**($n$, $k$) = $g(k/n)$.
        - **fbound**($n$) = [0, 1].
- 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 $f(\lambda)$ = **cosh(λ)/4 − 1/4**. Then simulate $f$ by first flipping the input coin twice. If both flips return 0, return 0. Otherwise, flip the input coin. If it returns 0, return 0. Otherwise, simulate $g(\lambda)$ (a function described below) and return the result.
  Let $g(\lambda)$ = 1/16 if $\lambda = 0$; -(cosh($\lambda$) − 1)/(4*$\lambda^2$*($\lambda$ − 2)) 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**($n$, $k$) = 503/10000 if $n$<4; otherwise, $g(k/n)$ − 17063/(350000*n).
        - **fabove**($n$, $k$) = $g(k/n)$.
        - **fbound**($n$) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = 23/400 if $n$<4; otherwise, $g(k/n)$ − 221819/(11200000*n).
        - **fabove**($n$, $k$) = $g(k/n)$.
        - **fbound**($n$) = [0, 1].
- Let $f(\lambda)$ = **exp(λ)/4 − 1/4**. Then simulate $f$ by first flipping the input coin twice. If both flips return 0, return 0. Otherwise, simulate $g(\lambda)$ (a function described below) and return the result.
  Let $g(\lambda)$ = 1/8 if $\lambda = 0$; -(exp($\lambda$) − 1)/(4*$\lambda$*($\lambda$ − 2)) 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**($n$, $k$) = 7/100 if $n$<4; otherwise, $g(k/n)$ − 9617/(43750*n).
        - **fabove**($n$, $k$) = $g(k/n)$.
        - **fbound**($n$) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = 207/2000 if $n$<4; otherwise, $g(k/n)$ − 9617/(112000*n).
        - **fabove**($n$, $k$) = $g(k/n)$.
        - **fbound**($n$) = [0, 1].
- Let $f(\lambda)$ = **sin(3*λ)/2**. Then, for every integer $n$ that's a power of 2, starting from 1:
    - The function was detected to be concave and twice differentiable, leading to:
        - **fbelow**($n$, $k$) = $f(k/n)$.
        - **fabove**($n$, $k$) = 1319/2000 if $n$<4; otherwise, $f(k/n)$ + 9/(14*n).
        - **fbound**($n$) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = $f(k/n)$.
        - **fabove**($n$, $k$) = 1299/2000 if $n$<4; otherwise, $f(k/n)$ + 135/(224*n).
        - **fbound**($n$) = [0, 1].
- Let $f(\lambda)$ = **exp(−λ)**. 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**($n$, $k$) = 3321/10000 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**($n$, $k$) = 861/2500 if $n$<4; otherwise, $f(k/n)$ − 3/(32*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 $f(\lambda)$ = **sin(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 concave and twice differentiable, leading to:
        - **fbelow**($n$, $k$) = $f(k/n)$.
        - **fabove**($n$, $k$) = 4191/5000 if $n$<4; otherwise, $f(k/n)$ + 246741/(700000*n).
        - **fbound**($n$) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = $f(k/n)$.
        - **fabove**($n$, $k$) = 8313/10000 if $n$<4; otherwise, $f(k/n)$ + 14557719/(44800000*n).
        - **fbound**($n$) = [0, 1].
- Let $f(\lambda)$ = **sqrt(λ)**. Then simulate $f$ by first simulating a polynomial with the following coefficients: [0, 1]. If it returns 0, return 1. Otherwise, simulate $g(\lambda)$ (a function described below) and return 1 minus the result. During the simulation, instead of flipping the input coin as usual, a different coin is flipped which does the following: "Flip the input coin and return 1 minus the result."
  Let $g(\lambda)$ = 1/2 if $\lambda$ = 0; (1 − sqrt(1 − $\lambda$))/$\lambda$ otherwise. 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**($n$, $k$) = $g(k/n)$ − 147735488601/(80000000*$n^{1/4}$).

        - **fabove**($n$, $k$) = $g(k/n)$.
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = $g(k/n) - 147735488601/(5120000000*n^{1/4})$.
        - **fabove**($n$, $k$) = $g(k/n)$.
- Let $f(\lambda)$ = $\mathbf{1 - \lambda^2}$. Then simulate $f$ by first flipping the input coin. If it returns 0, return 1. Otherwise, flip the input coin twice. If both flips return 0, return 1. Otherwise, simulate $g(\lambda)$ (a function described below) and return 1 minus the result. Let $g(\lambda) = 1/(2 - \lambda)$. 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**($n$, $k$) = 857/2000 if $n$<4; otherwise, $g(k/n) - 2/(7*n)$.
        - **fabove**($n$, $k$) = $g(k/n)$.
        - **fbound**($n$) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = 4709/10000 if $n$<4; otherwise, $g(k/n) - 13/(112*n)$.
        - **fabove**($n$, $k$) = $g(k/n)$.
        - **fbound**($n$) = [0, 1].
- Let $f(\lambda)$ = $\mathbf{sqrt(1 - \lambda^2)}$. Then simulate $f$ by first flipping the input coin. If it returns 0, return 1. Otherwise, flip the input coin. If it returns 0, return 1. Otherwise, simulate $g(\lambda)$ (a function described below) and return 1 minus the result.

  Let $g(\lambda) = 1/2$ if $\lambda = 0$; $(1 - \text{sqrt}(1 - \lambda^2))/\lambda^2$ otherwise. 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**($n$, $k$) = $g(k/n) - 4052384401283993864847 36/(1953125*n^{1/4})$.
        - **fabove**($n$, $k$) = $g(k/n)$.
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = $g(k/n) - 6331850627006240413824/(1953125*n^{1/4})$.
        - **fabove**($n$, $k$) = $g(k/n)$.
- Let $f(\lambda)$ = $\mathbf{cos(\pi*\lambda)/4 + 1/2}$. 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**($n$, $k$) = 809/5000 if $n$<4; otherwise, $f(k/n) - 246741/(700000*n)$.
        - **fabove**($n$, $k$) = 4191/5000 if $n$<4; otherwise, $f(k/n) + 246741/(700000*n)$.
        - **fbound**($n$) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = 409/2000 if $n$<4; otherwise, $f(k/n) - 8142453/(44800000*n)$.
        - **fabove**($n$, $k$) = 1591/2000 if $n$<4; otherwise, $f(k/n) + 8142453/(44800000*n)$.
        - **fbound**($n$) = [0, 1].
- Let $f(\lambda)$ = $\mathbf{\lambda*sin(7*\pi*\lambda)/4 + 1/2}$. 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(2*λ)/2**. Then, for every integer $n$ that's a power of 2, starting from 1:
    - The function was detected to be concave and twice differentiable, leading to:
        - **fbelow**($n$, $k$) = $f(k/n)$.
        - **fabove**($n$, $k$) = 2851/5000 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**($n$, $k$) = 5613/10000 if $n$<4; otherwise, $f(k/n)$ + 1/(4*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 $f(\lambda)$ = **$\lambda^2$/2 + 1/10 if $\lambda \le 1/2$; $\lambda$/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 $f(\lambda)$ = **$\lambda$/2 if $\lambda \leq 1/2$; (4*$\lambda$ − 1)/(8*$\lambda$) 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 $f(\lambda)$ = **1/2 − sqrt(1 − 2*$\lambda$)/2 if $\lambda < 1/2$; sqrt(2*$\lambda$ − 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 $f(\lambda)$ = **1/2 − sqrt(1 − 2*$\lambda$)/4 if $\lambda < 1/2$; sqrt(2*$\lambda$ − 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 $f(\lambda)$ = **$\lambda$/2 + (1 − 2*$\lambda$)$^{3/2}$/12 − 1/12 if $\lambda < 0$; $\lambda$/2 + (2*$\lambda$ − 1)$^{3/2}$/12 − 1/12 if $\lambda \geq 1/2$; $\lambda$/2 + (1 − 2*$\lambda$)$^{3/2}$/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 $f(\lambda)$ = **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 $f(\lambda)$ = **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 $f(\lambda)$ = **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 $f(\lambda)$ = **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 $f(\lambda)$ = **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 $f(\lambda)$ = **min(λ, 1 − 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**($n$, $k$) = 9563/10000 if $n$<8; otherwise, $f(k/n)$ + 322613/(250000*sqrt(n)).
        - **fbound**($n$) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = $f(k/n)$.
        - **fabove**($n$, $k$) = 123/125 if $n$<4; otherwise, $f(k/n)$ + 967839/(1000000*sqrt(n)).
        - **fbound**($n$) = [0, 1].
- Let $f(\lambda)$ = **λ/2 if λ ≤ 1 − λ; 1/2 − 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**($n$, $k$) = 5727/10000 if $n$<4; otherwise, $f(k/n)$ + 322613/(500000*sqrt(n)).
        - **fbound**($n$) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**($n$, $k$) = $f(k/n)$.
        - **fabove**($n$, $k$) = 123/250 if $n$<4; otherwise, $f(k/n)$ + 967839/(2000000*sqrt(n)).
        - **fbound**($n$) = [0, 1].
- Let $f(\lambda)$ = **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 $f(\lambda)$ = **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].

# 7 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] This means the exceptions make up a zero-volume (Lebesgue measure zero) set of points.
- [3] Specifically, the constant $m$ is a number equal to or greater than abs($f(x)-f(y)$)/(abs($x-y$)$^{\alpha}$) for every $x$ in [0, 1] and every $y$ in [0, 1] such that $x$ != $y$. However, $m$ can't directly be calculated as it would involve checking an infinite number of $x$, $y$ pairs.
- [4] Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010.
- [5] Powell, M.J.D., *Approximation Theory and Methods*, 1981
- [6] G. G. Lorentz. Bernstein polynomials. 1986.
- [7] Popoviciu, T., "Sur l'approximation des fonctions convexes d'ordre supérieur", Mathematica (Cluj), 1935.
- [8] Sikkema, P.C., "Der Wert einiger Konstanten in der Theorie der Approximation mit Bernstein-Polynomen", Numer. Math. 3 (1961).
- [9] Henderson, S.G., Glynn, P.W., "Nonexistence of a class of variate generation schemes", *Operations Research Letters* 31 (2003).
- [10] For this approximation, if $n$ were infinity, the method would return 1 with probability 1 and so would not approximate $\lambda*c$, of course.
- [11] 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.
- [12] 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.
- [13] Holtz, O., Nazarov, F., Peres, Y., "New Coins from Old, Smoothly", *Constructive Approximation* 33 (2011).
- [14] Keane, M. S., and O'Brien, G. L., "A Bernoulli factory", *ACM Transactions on Modeling and Computer Simulation* 4(2), 1994.
- [15] von Neumann, J., "Various techniques used in connection with random digits", 1951.
- [16] Peres, Y., "**Iterating von Neumann's procedure for extracting random bits**", Annals of Statistics 1992,20,1, p. 590-597.
- [17] Knuth, Donald E. and Andrew Chi-Chih Yao. "The complexity of nonuniform random variate generation", in *Algorithms and Complexity: New Directions and Recent Results*, 1976.
- [18] Mossel, Elchanan, and Yuval Peres. New coins from old: computing with unknown bias. Combinatorica, 25(6), pp.707-724.
- [19] S. Pae, "**Binarization Trees and Random Number Generation**", arXiv:1602.06058v2 [cs.DS], 2018.
- [20] Levy, H., *Stochastic dominance*, 1998.
- [21] 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** .
- [22] Gal, S.G., "Calculus of the modulus of continuity for nonconcave functions and applications", *Calcolo* 27 (1990)

- [23] 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.
- [24] Anastassiou, G.A., Gal, S.G., *Approximation Theory: Moduli of Continuity and Global Smoothness Preservation*, Birkhäuser, 2012.


# 8 Appendix

## 8.1 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 maps its domain to [0, 1]. The domain of $f$ gives the allowable values of $\lambda$, which is the input coin's probability of heads.

In general, $f$ 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 in the section "Proofs for Function Approximation Schemes" (Keane and O'Brien 1994)**(14)**.

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 $n^{\text{th}}$ 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.

## 8.2 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)**(14)**. 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 $\lambda$ (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 problem 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.

We can show that $f$ is strongly simulable on its domain by showing 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 $j^{\text{th}}$ coefficient of the polynomial in Bernstein form. Consider the following algorithm, modified from (Goyal and Sigman 2012)**(11)**.

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. Generate a uniform(0, 1) random variate, then return 1 if that variate is less than $a[j]$ ($a[j]$ 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)**(15)**), 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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $f(\lambda)$ is described in the strong simulability statement and meets the additional conditions below, then $f$ is strongly simulable.*

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

*Proof:* First, assume $g$ and $\omega$ have the same degree. If not, elevate the degree of the polynomial with lesser degree to have the same degree as the other.

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

1. Flip the input coin $n$ times, and let $j$ be the number of times the coin returned 1 this way.
2. If 0 is in the domain of $f$ and if $j$ is 0, return $g(0) = \omega(0) = 0$.
3. If 1 is in the domain of $f$ and if $j$ is $n$, return $g(1) = \omega(1) = 0$.
4. Generate a uniform(0, 1) random variate, then return 1 if that variate is less than

$\omega[j]$, or return 0 if that variate is greater than $g[j]$. This step is carried out via the von Neumann method, as in Proposition 1.

If the algorithm didn't return a value, then by now we know that the input coin's probability of heads is neither 0 or 1, since step 2 returned a value (either 0 or 1), which can only happen if the input coin didn't return all zeros or all ones.

Now let $r(\lambda) = (f(\lambda) - \omega(\lambda)) / (g(\lambda) - \omega(\lambda))$. By the conditions in the lemma, $h$ will be positive everywhere in the interval (0, 1), wherever $f$ is defined.

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

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

*Proof of Proposition 2:* The following cases can occur:

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

□

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

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

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

- $M$ be the Lipschitz constant of $f$ (e.g., its "slope" function's maximum absolute value), 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 $\omega$, 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 $\omega$ are set as 0. (As a result, the polynomial $g$ will have computable coefficients.) Then $g$ and $\omega$ 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.

# 8.3 Multiple-Output Bernoulli Factory

A related topic is a Bernoulli factory that takes a coin with unknown probability of heads $\lambda$ 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 $f$, there may be a more efficient algorithm in terms of input coin flips.

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 the Shannon entropy function. The question is:

*When the probability $\lambda$ can be any value in $J$, is there a multiple-output Bernoulli factory for $f(\lambda)$ with an expected 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 $\lambda$ 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)[16], then build a binary tree that generates 1 with probability $c$ and 0 otherwise (Knuth and Yao 1976)[17].

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)[12], 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; one example is the iterated von Neumann construction in Peres (1992)**(16)**. 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 any rational function of the form $D(\lambda)/E(\lambda)$, where—

- $D(\lambda) = \sum_{i = 0, \ldots, k} \lambda^i * (1 - \lambda)^{k - i} * d[i]$,
- $E(\lambda) = \sum_{i = 0, \ldots, k} \lambda^i * (1 - \lambda)^{k - i} * e[i]$,
- 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)**(18)**, 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 $[0, k]$, $e[i] < \text{choose}(k, i)*\text{choose}(2*r, r)$.

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 $\beta$. If $\beta < d[i]$, pass $\beta$ 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 $f(\lambda) = P1 \mathbin{/} P01$. 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) **(19)**. 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 $\lambda$.

†† Peres described this algorithm only incompletely. By simulation I found that the algorithm has to use two extractors (extractor 1 and extractor 2) as well as the method for generating unbiased bits given in this section; and for the three cases given here, the bits (0), (1, 0), and (1, 1), respectively, are passed as input bits to extractor 2. These measures are needed for correctness.

# 8.4 Proofs for Function Approximation Schemes

This section shows mathematical proofs for some of the approximation 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 $\min(\lambda^n, (1-\lambda)^n)$ for some integer $n$ (Keane and O'Brien 1994)**(14)**.

- A *modulus of continuity* of a function $f$ means a non-negative and nondecreasing function on the interval [0, 1], for which $\omega(0) = 0$, and for which abs(f($x$) − f($y$)) ≤ $\omega$(abs($x$−$y$)) for every $x$ in [0, 1] and every $y$ in [0, 1]. Loosely speaking, the modulus of continuity $\omega(\delta)$ gives $f$'s maximum range in a window of size $\delta$.

**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 **E**[f($X_k$/n)] is nondecreasing as k increases.*

*Proof.* This is equivalent to verifying whether $X_{m+1}/n \succeq X_m/n$ (and, obviously by extension, $X_{m+1} \succeq X_m$) in terms of first-degree stochastic dominance (Levy 1998)**(20)**. 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**(21)**. □

Lemma 6(i) of Nacu and Peres (2005)**(1)** can be applied to continuous functions beyond just Lipschitz continuous functions. This includes *Hölder continuous* functions, namely continuous functions with no slope "steeper" than any "nth" root.

**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 α-Hölder continuous with Hölder constant M and with 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. $\omega$ 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**[$\omega$(abs($X$/$n$ − $k$/(2 * $n$))] ≤ $\omega$(**E**[abs($X$/$n$ − $k$/(2 * $n$))]) (by Jensen's inequality and because $\omega$ is concave) ≤ $\omega$(sqrt(**E**[abs($X$/$n$ − $k$/(2 * $n$))]$^2$)) = $\omega$(sqrt(**Var**[$X$/$n$])) = $\omega$(sqrt(($k$*(2 * $n$−$k$)/(4*(2 * $n$−1)*$n^2$)))) ≤ $\omega$(sqrt(($n^2$/(4*(2 * $n$−1)*$n^2$)))) = $\omega$(sqrt((1/(8*$n$−4)))) = $\rho$, and for every $n$≥4 that are integer powers of 2, $\rho$ ≤ $\omega$(sqrt(1/(7*$n$))). To prove the third bound: abs(**E**[$f$($X$/$n$)] − $f$($k$/(2 * $n$))) ≤ $\omega$(sqrt(**Var**[$X$/$n$])) ≤ $\omega$(sqrt(1/(2*n))). To prove the fourth bound: abs(**E**[$f$($X$/$n$)] −

$f(k/(2 * n))) \le \omega(\text{sqrt}((n^2/(4*(2 * n-1)*n^2)))) = \omega(\text{sqrt}( (k/(2*n)) * (1-k/(2*n)) / (2*n-1) ))$.

2. By the definition of Hölder continuous functions, take $\omega(x) = M*x^{\alpha}$. Because $\omega$ is a concave modulus of continuity on [0,1], the result follows from part 1.
3. $\text{abs}(\mathbf{E}[f(X/n)] - f(k/(2 * n))) \le (M/2)*\mathbf{Var}[X/n] = (M/2)*(k*(2 * n-k)/(4*(2 * n-1)*n^2)) \le (M/2)*(n^2/(4*(2 * n-1)*n^2)) = (M/2)*(1/(8*n-4)) = \rho$. For every integer $n \ge 4$ that's a power of 2, $\rho \le (M/2)*(1/(7*n))$.
4. Let $X_m$ be a hypergeometric$(2 * n, m, n)$ random variable. By Lemma 1 and the assumption that $f$ is nondecreasing, $\mathbf{E}[f(X_k/n)]$ is nondecreasing as $k$ increases, so take $\mathbf{E}[f(X_n/n)] = \mathbf{E}[f(Y/n)]$ as the upper bound. Then, $\text{abs}(\mathbf{E}[f(X/n)] - f(k/(2 * n))) = \text{abs}(\mathbf{E}[f(X/n)] - f(\mathbf{E}[X/n])) = \mathbf{E}[f(X/n)] - f(\mathbf{E}[X/n])$ (by Jensen's inequality, because $f$ is convex and bounded by 0) $= \mathbf{E}[f(X/n)] - f(k/(2 * n)) \le \mathbf{E}[f(X/n)]$ (because $f$ is bounded by 0) $\le \mathbf{E}[f(Y/n)]$. □

**Notes:**

1. $\mathbf{E}[.]$ means expected or average value, and $\mathbf{Var}[.]$ means variance. A hypergeometric$(2 * n, k, n)$ random variable is the number of "good" balls out of $n$ balls taken uniformly at random, all at once, from a bag containing $2 * n$ balls, $k$ of which are "good".
2. $f$ is $\alpha$-Hölder continuous if its vertical slopes, if any, are no "steeper" than that of $M*\lambda^{\alpha}$, where $\alpha$ is in the interval (0, 1] and $M$ is greater than 0. An $\alpha$-Hölder continuous function on the closed interval [0, 1] is also $\beta$-Hölder continuous for any $\beta$ less than $\alpha$.
3. Parts 1 and 2 exploit a tighter bound on $\mathbf{Var}[X/n]$ than the bound given in Nacu and Peres (2005, Lemma 6(i) and 6(ii), respectively)**(1)**. However, for technical reasons, different bounds are proved for different ranges of integers $n$.
4. 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).
5. 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 $\omega$ is overestimated. The following functions have a simple $\omega$ that satisfies the lemma:
    1. If $f$ is monotone increasing and convex, $\omega(x)$ can equal $f(1) - f(1-x)$ (Gal 1990)**(22)**; (Gal 1995)**(23)**.
    2. If $f$ is monotone decreasing and convex, $\omega(x)$ can equal $f(0) - f(x)$ (Gal 1990)**(22)**; (Gal 1995)**(23)**.
    3. If $f$ is monotone increasing and concave, $\omega(x)$ can equal $f(x) - f(0)$ (by symmetry with 2).
    4. If $f$ is monotone decreasing and concave, $\omega(x)$ can equal $f(1-x) - f(1)$ (by symmetry with 1).
    5. If $f$ is concave and is monotone increasing then monotone decreasing, then $\omega(h)$ can equal $(f(\min(h, \sigma))+(f(1-\min(h, 1-\sigma))-f(1))$, where $\sigma$ is the point where $f$ stops increasing and starts decreasing (Anastassiou and Gal 2012)**(24)**.

**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 approximation 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 $\varepsilon$ be a positive distance between 0 and the minimum or between 1 and the maximum of $f$, whichever is smaller. This $\varepsilon$ exists by the assumption that $f$ is bounded away from 0 and 1. Because the series $\eta$ converges, $\eta(n)$ will eventually stay less than $\varepsilon$. 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 $\varphi$ 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 identically 0 and identically 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 α-Hölder continuous with Hölder constant M and with α in the interval (0, 1], then the following approximation 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) − δ(n).*
- ***fabove****(n, k) = f(k/n) + δ(n).*

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

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

> **Note:** For specific values of $\alpha$, the equation $D(n) = D(2 * n) + \varphi(n)$ can be solved via linear recurrences; an example for $\alpha = 1/2$ is the following SymPy code: `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 $\alpha$ suggested the following formula for Hölder continuous functions with $\alpha$ 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 approximation 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/((sqrt(2)−1)*sqrt(n)).*
- ***fabove****(n, k) = f(k/n) + M/((sqrt(2)−1)*sqrt(n)).*

*Proof.* Because Lipschitz continuous functions are 1-Hölder continuous with Hölder constant $M$, 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(λ) be a strictly bounded factory function, and let ω(x) be as described in Theorem 1. Theorem 1 remains valid with the following versions of φ(n),* ***fbelow****, and* ***fabove****, rather than as given in that theorem, subject to the bounding note:*

- *φ(n) = ω(sqrt(1/(7*n))).*
- ***fbelow****(n, k) = min(****fbelow****(4,0),* ***fbelow****(4,1), ...,* ***fbelow****(4,4)) if n < 4; otherwise, f(k/n) − η(n).*
- ***fabove****(n, k) = max(****fabove****(4,0),* ***fabove****(4,1), ...,* ***fabove****(4,4)) if n < 4; otherwise,*

*f(k/n) + η(n).*

*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 $\varphi$ 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(λ) be a strictly bounded factory function. If f is α-Hölder continuous with Hölder constant M and with α in the interval (0, 1], then the following approximation 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) − η(n).*
- ***fabove**(n, k) = max(**fabove**(4,0), **fabove**(4,1), ..., **fabove**(4,4)) if n < 4; otherwise, f(k/n) + η(n).*

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

*Proof.* Follows from Theorem 2 by using the $\omega$ given in part 2 of Lemma 2. □

**Theorem 3.** *Let f(λ) 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 approximation 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 ω(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(λ) be a strictly bounded factory function. If f is convex and nondecreasing, then Theorem 1 remains valid with φ(n) = **E**[f(Y/n)] (where Y is a hypergeometric(2*n, n, n) random variable), rather than as given in that theorem.*

*Proof.* Follows from Theorem 1 and part 4 of Lemma 2 above. With the $\varphi$ 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 approximation 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 approximation 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 approximations 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)]. □

# 8.5 Example of Approximation Scheme

The following example uses the results above to build an approximation scheme for a factory function.

Let $f(\lambda) = 0$ if $\lambda$ is 0, and $(\ln(\lambda/\exp(3)))^{-2}$ otherwise. Then the following approximation scheme is valid in the sense of Theorem 1:

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

Where Φ is a function called the *Lerch transcendent*, and **fabove** is subject to Theorem 1's bounding note.

Notice that the function $f$ is not Hölder continuous; its slope is exponentially steep at the point 0.

The first step is to find a concave modulus of continuity of $f$ (called $\omega(h)$). Because $f$ is monotone increasing and concave, and because $f(0) = 0$, we can take $\omega(h) = f(h)$.

Now, by plugging sqrt(1/(7*$n$)) into $\omega$, we get the following for Theorem 2 (assuming $n$≥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($k$). $\eta$ converges, resulting in:

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

where Φ is the Lerch transcendent. This $\eta$ matches the $\eta$ 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 SymPy code is an example of finding the parameters for this approximation

scheme.

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

<h1>px is monotone 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))
```

# 9 License