Supplemental Notes for Bernoulli Factory Algorithms

This version of the document is dated 2021-04-01.

Peter Occil

1 Contents

- **Contents**
- **General Factory Functions**
    - **Approximation Schemes**
    - **Schemes That Don't Work**
- **Approximate Bernoulli Factory via a Single Polynomial**
- **Achievable Simulation Rates**
- **Complexity**
- **Examples of Bernoulli Factory Approximation Schemes**
- **Notes**
- **Appendix**
    - **Proofs for Function Approximation Schemes**
    - **Example of Approximation Scheme**
- **License**

2 General Factory Functions

As a reminder, the *Bernoulli factory problem* is: Given a coin with unknown probability of heads of λ, 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 number, 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^{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 all n that are powers 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 discontinuous (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 α-***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 α 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 at all but a countable number of points, and does not tend to a vertical slope anywhere, then f is ***Lipschitz continuous***, α is 1, and m is the highest absolute value of the function's "slope". Otherwise, finding m for a given α is non-trivial and it requires knowing where f's vertical slopes are, among other things.[(2)] But assuming m and α are known, then for all n that are powers of 2:

- $\delta(n) = m*(2/7)^{\alpha/2}/((2^{\alpha/2}-1)*n^{\alpha/2})$.
- **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, $f(k/n) - \delta(n)$.
- **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, $f(k/n) + \delta(n)$.

  Note:

  1. Some functions f are not α-Hölder continuous for any α greater than 0. These functions have an exponentially steep slope (steeper than any "nth" root) and can't be handled by this method. One example is $f(\lambda) = 1/10$ if λ is 0 and $-1/(2*\ln(\lambda/2)) + 1/10$ otherwise, which has an exponentially steep slope near 0.
  2. In the Lipschitz case ($\alpha = 1$), $\delta(n)$ can be $m*322613/(250000*\text{sqrt}(n))$, which is an upper bound.
  3. In the case $\alpha = 1/2$, $\delta(n)$ 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 λ is 0.)
- $1 \geq h(1) \geq f(1) \geq 0$.
- $1 > h(\lambda) > f(\lambda) > 0$ for all λ in the open interval (0, 1).
- If $f(1) = 0$, then $h(1) = 0$. (This is required to ensure correctness in case λ is 1.)

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

The algorithm is now described.

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

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

then f can be simulated using the following algorithm:

1. Run a Bernoulli factory algorithm for h. If the call returns 0, return 0. (For example, if $h(\lambda) = \lambda$, then this step amounts to the following: "Flip the input coin. If it returns 0, return 0.")
2. Run 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 number 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)[(3)] (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 neither flip returns 1, 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 all λ in the open interval (0, 1).
- $q(\lambda) = \lim_{\nu\to\lambda} \omega(\nu)/h(\nu)$.
- $r(\lambda) = \lim_{\nu\to\lambda} (1-g(\nu))/(1-q(\nu))$.

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

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

and ≤ 1	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)**(4)**, let *f* be a C^2 continuous function (a function with continuous "slope" and "slope-of-slope" functions) in [0, 1]. Then for all $n \geq 1$:

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

Where M is an upper bound of 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 at all but a countable number of points, and does not tend to a vertical slope anywhere). Then for all $n \geq 2$:

- **fabove**(n, k) = $f(k/n)$ + L*(5/4) / sqrt(n).
- **fbelow**(n, k) = $f(k/n)$ − L*(5/4) / 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)**(5)**, (Popoviciu 1935)**(6)**.

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(λ, 1−λ)/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*π*λ)/4 + 1/2, a "smooth" function with Lipschitz constant π. 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*sqrt(6))/5832 (Lorentz 1986)**(5)**, (Sikkema 1961)**(7)**, which equals about 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 Factory via a Single Polynomial

Although the schemes in the previous section don't work when building a *family* of polynomials that converge to a factory 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.

More specifically, we approximate f using one polynomial in Bernstein form of degree n. The higher n is, the better this approximation. In fact, since every factory function is continuous, we can choose n high enough that the polynomial differs from f by no more than ε, where $\varepsilon > 0$ is the desired error tolerance.

Then, one of the algorithms in the section "**Certain Polynomials**" can be used to simulate that polynomial: the polynomial has n+1 coefficients and its j^{th} coefficient (starting at 0) is found as $f(j/n)$.

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 ε, we need to solve the following equation for n:

- ε = L*((4306+837*sqrt(6))/5832) / sqrt(n).

This has the following solution:

- n = L^2*(3604122*sqrt(6) + 11372525)/(17006112*ε^2).

This is generally not an integer, so we use n = 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 = ceil(59393*L^2/(50000*ε^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 ε:

1. Calculate n as ceil(59393*L^2/(50000*ε^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*ε)) (upper bound of the solution to the equation ε = 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.

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, λ, lies in the interval [ε, 1−ε] for some $\varepsilon > 0$. In the table below, $\Delta(n, r, \lambda) = O(\max(\text{sqrt}(\lambda*(1-\lambda)/n),1/n)^r)$, that is, $O((1/n)^r)$ near $\lambda = 0$ or 1, and $O((1/n)^{r/2})$ elsewhere. ($O(h(n))$ roughly means "grows no faster than $h(n)$".)

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)[(8)].
Requires a finite number of flips on average. Also known as "realizable" by Flajolet et al. (2010)[(3)].	Only if *f* is Lipschitz continuous (Nacu and Peres 2005)[(1)].
Number of flips required, raised to power of r, is finite on average and tends to 0 has a tail that drops off uniformly for all λ.	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 all λ. (The greater r is, the faster the simulation.)	Only if *f* is C^r continuous and the r^{th} derivative is in the Zygmund class (has no vertical slope) (Holtz et al. 2011)[(9)].
Requires more than n flips with probability $\Delta(n, \alpha, \lambda)$, for non-integer $\alpha > 0$ and all λ. (The greater α is, the faster the simulation.)	If and only if *f* is C^r continuous and the r^{th} derivative is $(\alpha - r)$-Hölder continuous, where $r = \text{floor}(\alpha)$ (Holtz et al. 2011)[(9)].
"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)[(3)].	If and only if *f* is real analytic (is C^∞ continuous, or has continuous k^{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)[(10)].

5 Complexity

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

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

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

A similar analysis to the one above can be used to find the expected (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 λ. The schemes were generated automatically using `approxscheme2` and have not been rigorously verified for correctness.

- Let $f(\lambda)$ = **min(1/2, λ)**. Then, for all n that are powers 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) = 371/500 if n<4; otherwise, $f(k/n)$ + 967839/(2000000*sqrt(n)).
    - **fbound**(n) = [0, 1].
- Let $f(\lambda)$ = **cosh(λ) − 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 λ = 0; -(cosh(λ) − 1)/(λ^2*(λ − 2)) otherwise. Then, for all n that are powers 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*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 all n that are powers 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, but highly likely to be correct:
        - **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 λ = 0; -(cosh(λ) − 1)/(4*λ^2*(λ − 2)) otherwise. Then, for all n that are powers 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 λ = 0; -(exp(λ) − 1)/(4*λ*(λ − 2)) otherwise. Then, for all n that are powers 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 all n that are powers 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, but highly likely to be correct:
        - **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 all n that are powers 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, but highly likely to be correct:
        - **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*(λ) = **3/4 − sqrt(−λ*(λ − 1))**. Then, for all *n* that are powers 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*(λ) = **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*(λ) (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*(λ) = 1/2 if λ = 0; (1 − sqrt(1 − λ))/λ otherwise. Then, for all *n* that are powers 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*(λ) = **3*sin(sqrt(3)*sqrt(sin(2*λ)))/4 + 1/50**. Then, for all *n* that are powers 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*(λ) = **3/4 − sqrt(−λ*(λ − 1))**. Then, for all *n* that are powers 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*(λ) = **3/4 − sqrt(25 − 4*(5*λ + 2)^2)/10 if λ ≤ 1/10; 3/4 − sqrt(25 − 4*(5*λ − 3)^2)/10 otherwise**. Then, for all *n* that are powers 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*) − 44827752327/(25600000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*).

- Let $f(\lambda)$ = **sin(π*λ)/4 + 1/2**. Then, for all n that are powers 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, but highly likely to be correct:
    - **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 λ = 0; (1 − sqrt(1 − λ))/λ otherwise. Then, for all n that are powers 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 $f(\lambda)$ = **1 − λ²**. 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 − λ). Then, for all n that are powers 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, but highly likely to be correct:
    - **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 $f(\lambda)$ = **sqrt(1 − λ²)**. 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 λ = 0; (1 − sqrt(1 − λ^2))/λ^2 otherwise. Then, for all n that are powers 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) = $g(k/n)$ − 633185062700624041382 4/(1953125*$n^{1/4}$).
    - **fabove**(n, k) = $g(k/n)$.
- Let $f(\lambda)$ = **cos(π*λ)/4 + 1/2**. Then, for all n that are powers 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, but highly likely to be correct:
        - **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*(*λ*) = **λ*sin(7*π*λ)/4 + 1/2**. Then, for all *n* that are powers 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*) = 111/2500 if *n*<32; otherwise, *f*(*k*/*n*) − 374438493/(44800000*n).
        - **fabove**(*n*, *k*) = 9911/10000 if *n*<32; otherwise, *f*(*k*/*n*) + 374438493/(44800000*n).
        - **fbound**(*n*) = [0, 1].
- Let *f*(*λ*) = **sin(4*π*λ)/4 + 1/2**. Then, for all *n* that are powers 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*) = 219/2000 if *n*<32; otherwise, *f*(*k*/*n*) − 100669971/(22400000*n).
        - **fabove**(*n*, *k*) = 1781/2000 if *n*<32; otherwise, *f*(*k*/*n*) + 100669971/(22400000*n).
        - **fbound**(*n*) = [0, 1].
- Let *f*(*λ*) = **sin(6*π*λ)/4 + 1/2**. Then, for all *n* that are powers 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*) = 23/250 if *n*<64; otherwise, *f*(*k*/*n*) − 113253711/(11200000*n).
        - **fabove**(*n*, *k*) = 227/250 if *n*<64; otherwise, *f*(*k*/*n*) + 113253711/(11200000*n).
        - **fbound**(*n*) = [0, 1].
- Let *f*(*λ*) = **sin(2*λ)/2**. Then, for all *n* that are powers 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, but highly likely to be correct:
        - **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*(λ) = **sin(4*π*λ)/4 + 1/2**. Then, for all *n* that are powers 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*) = 219/2000 if *n*<32; otherwise, *f*(*k*/*n*) − 100669971/(22400000*n).
        - **fabove**(*n*, *k*) = 1781/2000 if *n*<32; otherwise, *f*(*k*/*n*) + 100669971/(22400000*n).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **λ^2/2 + 1/10 if λ ≤ 1/2; λ/2 − 1/40 otherwise**. Then, for all *n* that are powers 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*) = 687/10000 if *n*<4; otherwise, *f*(*k*/*n*) − 1/(8*n).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **λ/2 if λ ≤ 0.500000000000000; (4*λ − 1)/(8*λ) otherwise**. Then, for all *n* that are powers 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*) = 423/1000 if *n*<4; otherwise, *f*(*k*/*n*) + 43/(224*n).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **1/2 − sqrt(1 − 2*λ)/4 if λ < 1/2; sqrt(2*λ − 1)/4 + 1/2 otherwise**. Then, for all *n* that are powers of 2, starting from 1:
    - Detected to be (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*) − 772969563/(400000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 772969563/(400000000*$n^{1/4}$).
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = 193/5000 if *n*<16; otherwise, *f*(*k*/*n*) − 5410786941/(12800000000*$n^{1/4}$).
        - **fabove**(*n*, *k*) = 4807/5000 if *n*<16; otherwise, *f*(*k*/*n*) + 5410786941/(12800000000*$n^{1/4}$).
        - **fbound**(*n*) = [0, 1].

- Let *f*(λ) = **λ/2 + (1 − 2*λ)$^{3/2}$/12 − 1/12 if λ < 0; λ/2 + (2*λ − 1)$^{3/2}$/12 − 1/12 if λ ≥ 1/2; λ/2 + (1 − 2*λ)$^{3/2}$/12 − 1/12 otherwise**. Then simulate *f* by first flipping the input coin. If it returns 0, return 0. Otherwise, simulate *g*(*λ*) (a function described below) and return the result.

  Let *g*(*λ*) = 1/4 if *λ* = 0; (6**λ* + (1 − 2**λ*)$^{3/2}$ − 1)/(12**λ*) if *λ* < 0; (6**λ* + (2**λ* − 1)$^{3/2}$ − 1)/(12**λ*) if *λ* ≥ 1/2; (6**λ* + (1 − 2**λ*)$^{3/2}$ − 1)/(12**λ*) otherwise. Then, for all *n* that are powers of 2, starting from 1:
    - Detected to be Lipschitz continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = 627/10000 if *n*<8; otherwise, *g*(*k*/*n*) − 3310977219/(6250000000*sqrt(n)).
        - **fabove**(*n*, *k*) = 6873/10000 if *n*<8; otherwise, *g*(*k*/*n*) + 3310977219/(6250000000*sqrt(n)).
        - **fbound**(*n*) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = 1043/5000 if *n*<4; otherwise, *g*(*k*/*n*) − 3310977219/(40000000000*sqrt(n)).
        - **fabove**(*n*, *k*) = 2707/5000 if *n*<4; otherwise, *g*(*k*/*n*) + 3310977219/(40000000000*sqrt(n)).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **1/2 − sqrt(1 − 2*λ)/4 if λ < 1/2; sqrt(2*λ − 1)/4 + 1/2 otherwise**. Then, for all *n* that are powers of 2, starting from 1:
    - Detected to be (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*) − 772969563/(400000000*n$^{1/4}$).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 772969563/(400000000*n$^{1/4}$).
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = 193/5000 if *n*<16; otherwise, *f*(*k*/*n*) − 5410786941/(12800000000*n$^{1/4}$).
        - **fabove**(*n*, *k*) = 4807/5000 if *n*<16; otherwise, *f*(*k*/*n*) + 5410786941/(12800000000*n$^{1/4}$).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **1/2 − sqrt(1 − 2*λ)/8 if λ < 1/2; sqrt(2*λ − 1)/8 + 1/2 otherwise**. Then, for all *n* that are powers of 2, starting from 1:
    - Detected to be (1/2)-Hölder continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = 333/10000 if *n*<64; otherwise, *f*(*k*/*n*) − 386562063/(400000000*n$^{1/4}$).
        - **fabove**(*n*, *k*) = 9667/10000 if *n*<64; otherwise, *f*(*k*/*n*) + 386562063/(400000000*n$^{1/4}$).
        - **fbound**(*n*) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = 451/2000 if *n*<4; otherwise, *f*(*k*/*n*) − 2705934441/(12800000000*n$^{1/4}$).
        - **fabove**(*n*, *k*) = 1549/2000 if *n*<4; otherwise, *f*(*k*/*n*) + 2705934441/(12800000000*n$^{1/4}$).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **λ/4 + (1 − 2*λ)$^{3/2}$/24 + 5/24 if λ < 0; λ/4 + (2*λ − 1)$^{3/2}$/24 + 5/24 if λ ≥ 1/2; λ/4 + (1 − 2*λ)$^{3/2}$/24 + 5/24 otherwise**. Then, for all *n* that are powers of 2, starting from 1:
    - Detected to be convex and Lipschitz continuous using numerical methods, which may be inaccurate:

        - **fbelow**(*n*, *k*) = 1/125 if *n*<4; otherwise, *f*(*k*/*n*) − 967839/(2000000*sqrt(n)).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*).
        - **fbound**(*n*) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = 2197/10000 if *n*<4; otherwise, *f*(*k*/*n*) − 967839/(16000000*sqrt(n)).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **$\lambda^2/8 - \lambda/24 - (1 - 2*\lambda)^{5/2}/120 + 31/120$ if λ < 0; $\lambda^2/8 - \lambda/24 + (2*\lambda - 1)^{5/2}/120 + 31/120$ if λ ≥ 1/2; $\lambda^2/8 - \lambda/24 - (1 - 2*\lambda)^{5/2}/120 + 31/120$ otherwise**. Then, for all *n* that are powers of 2, starting from 1:
    - Detected to be convex and twice differentiable using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = 1183/5000 if *n*<4; otherwise, *f*(*k*/*n*) − 3/(56*n).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*).
        - **fbound**(*n*) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = 2397/10000 if *n*<4; otherwise, *f*(*k*/*n*) − 21/(512*n).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **3*λ/2 if λ ≤ 1 − λ; 3/2 − 3*λ/2 otherwise**. Then, for all *n* that are powers of 2, starting from 1:
    - Detected to be concave and Lipschitz continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
        - **fabove**(*n*, *k*) = 124/125 if *n*<64; otherwise, *f*(*k*/*n*) + 967839/(500000*sqrt(n)).
        - **fbound**(*n*) = [0, 1].
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
        - **fabove**(*n*, *k*) = 1863/2000 if *n*<64; otherwise, *f*(*k*/*n*) + 2903517/(2000000*sqrt(n)).
        - **fbound**(*n*) = [0, 1].
- Let *f*(λ) = **9*λ/5 if λ ≤ 1 − λ; 9/5 − 9*λ/5 otherwise**. Then, for all *n* that are powers of 2, starting from 1:
    - Detected to be concave and Lipschitz continuous using numerical methods, which may be inaccurate:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 2903517/(1250000*sqrt(n)).
    - Generated using tighter bounds than necessarily proven:
        - **fbelow**(*n*, *k*) = *f*(*k*/*n*).
        - **fabove**(*n*, *k*) = *f*(*k*/*n*) + 8710551/(5000000*sqrt(n)).
- Let *f*(λ) = **min(λ, 1 − λ)**. Then, for all *n* that are powers 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 $\lambda \le 1 - \lambda$; 1/2 − λ/2 otherwise**. Then, for all n that are powers 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 $\lambda \le 1 - \lambda$; 19/20 − 19*λ/20 otherwise**. Then, for all n that are powers 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 all n that are powers 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) Specifically, the constant m is an upper bound of abs($f(x)-f(y)$)/(abs$(x-y)^{\alpha}$) for all x, y pairs, where x and y are each in [0, 1] and x != y. However, this bound can't directly be calculated as it would involve checking an infinite number of x, y pairs.
- (3) Flajolet, P., Pelletier, M., Soria, M., "**On Buffon machines and numbers**", arXiv:0906.5560 [math.PR], 2010.
- (4) Powell, M.J.D., *Approximation Theory and Methods*, 1981
- (5) G. G. Lorentz. Bernstein polynomials. 1986.
- (6) Popoviciu, T., "Sur l'approximation des fonctions convexes d'ordre supérieur", Mathematica (Cluj), 1935.
- (7) Sikkema, P.C., "Der Wert einiger Konstanten in der Theorie der Approximation mit Bernstein-

Polynomen", Numer. Math. 3 (1961).
- (8) 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.
- (9) Holtz, O., Nazarov, F., Peres, Y., "New Coins from Old, Smoothly", *Constructive Approximation* 33 (2011).
- (10) 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.
- (11) Levy, H., *Stochastic dominance*, 1998.
- (12) 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** .
- (13) Gal, S.G., "Calculus of the modulus of continuity for nonconcave functions and applications", *Calcolo* 27 (1990)
- (14) 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.
- (15) Keane, M. S., and O'Brien, G. L., "A Bernoulli factory", *ACM Transactions on Modeling and Computer Simulation* 4(2), 1994.
- (16) Anastassiou, G.A., Gal, S.G., *Approximation Theory: Moduli of Continuity and Global Smoothness Preservation*, Birkhäuser, 2012.

8 Appendix

8.1 Proofs for Function Approximation Schemes

This section shows mathematical proofs for some of the approximation schemes of this page.

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)**[11]**. 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**[12]**. □

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 whose slope doesn't go exponentially fast to a vertical slope.

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 (a non-negative and nondecreasing function on the interval [0, 1], for which ω(0) = 0, and for which abs(f(x) − f(y)) ≤ ω(abs(x−y)) for all x in [0, 1] and all y in [0, 1]). If ω is concave on [0, 1], then the expression—*
   *abs(**E**[f(X/n)] − f(k/(2*n))),   (1)*
   is bounded from above by—

- $\omega(\text{sqrt}(1/(8*n-4)))$, *for all* $n \ge 1$ *that are integer powers of 2,*
- $\omega(\text{sqrt}(1/(7*n)))$, *for all* $n \ge 4$ *that are integer powers of 2,*
- $\omega(\text{sqrt}(1/(2*n)))$, *for all* $n \ge 1$ *that are integer powers of 2, and*
- $\omega(\text{sqrt}((k/(2*n)) * (1-k/(2*n)) / (2*n-1)))$, *for all* $n \ge 1$ *that are integer powers 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 all* $n \ge 1$ *that are integer powers of 2,*
   - $M*(1/(7*n))^{\alpha/2}$, *for all* $n \ge 4$ *that are integer powers of 2, and*
   - $M*(1/(8*n-4))^{\alpha/2}$, *for all integers* $n \ge 1$ *that are integer powers 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 all* $n \ge 4$ *that are integer powers of 2, and*
   - $(M/2)*(1/(8*n-4))$, *for all* $n \ge 1$ *that are integer powers 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 all n≥1 that are integer powers of 2, where Y is a hypergeometric(2*n, n, n) random variable.*

Proof.

1. ω is assumed to be non-negative because absolute values are non-negative. To prove the first and second bounds: abs($\mathbf{E}[f(X/n)] - f(k/(2*n))$) ≤ $\mathbf{E}[\text{abs}(f(X/n) - f(k/(2*n)))]$ ≤ $\mathbf{E}[\omega(\text{abs}(X/n - k/(2*n)))]$ ≤ $\omega(\mathbf{E}[\text{abs}(X/n - k/(2*n))])$ (by Jensen's inequality and because ω is concave) ≤ $\omega(\text{sqrt}(\mathbf{E}[\text{abs}(X/n - k/(2*n))]^2)) = \omega(\text{sqrt}(\mathbf{Var}[X/n])) = \omega(\text{sqrt}((k*(2*n-k)/(4*(2*n-1)*n^2)))) \le \omega(\text{sqrt}((n^2/(4*(2*n-1)*n^2)))) = \omega(\text{sqrt}((1/(8*n-4)))) = \rho$, and for all $n \ge 4$ that are integer powers of 2, $\rho \le \omega(\text{sqrt}(1/(7*n)))$. To prove the third bound: abs($\mathbf{E}[f(X/n)] - f(k/(2*n))$) ≤ $\omega(\text{sqrt}(\mathbf{Var}[X/n])) \le \omega(\text{sqrt}(1/(2*n)))$. To prove the fourth bound: abs($\mathbf{E}[f(X/n)] - f(k/(2*n))$) ≤ $\omega(\text{sqrt}((n^2/(4*(2*n-1)*n^2)))) = \omega(\text{sqrt}((k/(2*n)) * (1-k/(2*n)) / (2*n-1)))$.
2. By the definition of Hölder continuous functions, take $\omega(x) = M*x^\alpha$. Because ω is non-negative, nondecreasing, and concave in [0,1], the result follows from part 1.
3. abs($\mathbf{E}[f(X/n)] - f(k/(2*n))$) ≤ $(M/2)*\mathbf{Var}[X/n] = (M/2)*(k*(2*n-k)/(4*(2*n-1)*n^2)) \le (M/2)*(n^2/(4*(2*n-1)*n^2)) = (M/2)*(1/(8*n-4)) = \rho$. For all $n \ge 4$ that are integer powers 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, abs($\mathbf{E}[f(X/n)] - f(k/(2*n))$) = abs($\mathbf{E}[f(X/n)] - f(\mathbf{E}[X/n])$) = $\mathbf{E}[f(X/n)] - f(\mathbf{E}[X/n])$ (by Jensen's inequality, because f is convex and bounded by 0) = $\mathbf{E}[f(X/n)] - f(k/(2*n)) \le \mathbf{E}[f(X/n)]$ (because f is bounded by 0) ≤ $\mathbf{E}[f(Y/n)]$. □

   Notes:

   1. **E**[.] means expected or average value, and **Var**[.] means variance. A hypergeometric($2*n, k, n$) random variable is the number of "good" balls out of n balls taken uniformly at random, all at once, from a bag containing $2*n$ balls, k of which are "good".
   2. f is α-Hölder continuous if its vertical slopes, if any, are no "steeper" than that of $M*\lambda^\alpha$, where α is in the interval (0, 1] and M is greater than 0. An α-Hölder continuous function on the closed interval [0, 1] is also β-Hölder continuous for any β less than α.
   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 ω is overestimated. The following functions have a simple ω that satisfies the lemma:
   1. If f is monotone increasing and convex, $\omega(x)$ can equal $f(1) - f(1-x)$ (Gal 1990)[13]; (Gal 1995)[14].
   2. If f is monotone decreasing and convex, $\omega(x)$ can equal $f(0) - f(x)$ (Gal 1990)[13]; (Gal 1995)[14].
   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).

In the following results:

- A *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 *general factory function* means a continuous function $f(\lambda)$ on the closed interval [0, 1] that is either identically 0, identically 1, or such that 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)[15].

Theorem 1. *Let $\omega(x)$ be as described in part 1 of Lemma 2, and let $f(\lambda)$ be a bounded factory function. Let—*

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

for all $n\ge1$ that are integer powers of 2 (with $n=2^m$), where $\varphi(n)$ is either—

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

Assume that the sum in $\eta(n)$ converges. Then, 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, the result is an approximation scheme that meets conditions (i), (iii), and (iv) of Proposition 3 of Nacu and Peres (2005)[1], for all $n\ge1$ that are integer powers of 2:*

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

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]. The function $\eta(n)$ is found as a solution to the functional equation $\eta(n) = \eta(2 * n) + \varphi(n)$, with either form of φ given in the theorem, and functional equations of this kind were suggested in the proof of Proposition 10, to find the offset by which to shift the approximating polynomials. We note that for the sum stated for $\eta(n)$ in the theorem, each term of the sum is nonnegative making the sum nonnegative and, by the assumption that the sum converges, η(n) decreases with increasing n. □

Corollary 1. *Let f(λ) be a 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** meets conditions (i), (iii), and (iv) of Proposition 3 of Nacu and Peres (2005)***[(1)]**, for all n≥1 that are integer powers of 2:*

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

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

Proof. Follows from Theorem 1 by using the ω given in part 2 of Lemma 2. □

> **Note:** For specific values of α, the functional equation used in Corollary 1 can be solved via linear recurrences; an example for α = 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 α suggested the following formula for Hölder continuous functions with α of $1/j$ or greater: $(M* \sum_{i=0,\dots,(j*2)-1} 2^{i/(2*j)})/n^{1/(2*j)} = M / ((2^{1/(2*j)}-1)*n^{1/(2*j)})$; and generalizing the latter expression led to the term in the theorem. $\delta(n)$ can also be found as follows:
> $\delta(n) = M*(1/(2 * n))^{\alpha/2}$ /
> $(1 - (M*(1/(2*(n*2)))^{\alpha/2}) / (M*(1/(2 * n))^{\alpha/2}))$
> $= M/((2^{\alpha/2}-1)*n^{\alpha/2})$.

Corollary 2. *Let f(λ) be a bounded factory function. If f is Lipschitz continuous with Lipschitz constant M, then the following approximation scheme determined by **fbelow** and **fabove** meets conditions (i), (iii), and (iv) of Proposition 3 of Nacu and Peres (2005)***[(1)]**, for all n≥1 that are integer powers of 2:*

- ***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 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:*

- *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 φ given in this theorem, the sum $\eta(n)$ in Theorem 1 remains nonnegative; also, this theorem adopts Theorem 1's assumption that the sum converges, so that $\eta(n)$ still decreases with increasing n. □

Corollary 3. *Let $f(\lambda)$ be a 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* ***fabove*** *and* ***fbelow*** *meets conditions (i), (iii), and (iv) of Proposition 3 of Nacu and Peres (2005)*[**(1)**]*, for all n≥1 that are integer powers of 2:*

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

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

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

Theorem 3. *Let $f(\lambda)$ be a 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 determined by* ***fbelow*** *and* ***fabove*** *meets conditions (i), (iii), and (iv) of Proposition 3 of Nacu and Peres (2005)*[**(1)**]*, for all n≥1 that are integer powers of 2:*

- ***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. Follows from part 3 of Lemma 2 above as well as Remark B and the proof of Proposition 10 of Nacu and Peres, noting that the solution to the functional equation $\kappa(n) = \kappa(2 * n) + (M/2)*(1/(7*n))$ is $M/(7*n)$. Notably, this exploits 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. □

Theorem 4. *Let $f(\lambda)$ be a bounded factory function. If f is convex and nondecreasing, then Theorem 1 remains valid with* $\varphi(n) = \mathbf{E}[f(Y/n)]$ *(where Y is a hypergeometric(2*n, n, n) random variable), rather than as given in that theorem.*

Proof. Follows from Theorem 1 and part 4 of Lemma 2 above. With the φ given in this theorem, the sum $\eta(n)$ in Theorem 1 remains nonnegative; also, this theorem adopts Theorem 1's assumption that the sum 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 may be a general factory function. 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 may be a general factory function. 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 all n≥1 other than powers 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, and Jensen's inequality. Part 3 also follows from Remark B of Nacu and Peres (2005)**[1]**. □

8.2 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 λ is 0, and $-\lambda$*ln(λ) otherwise. Then the following approximation scheme is valid for all n≥1 that are integer powers of 2:

- $\eta(k)$ = [Complicated expression].
- **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) + $\eta(n)$.

Notice that this function 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 $\omega(h)$), using the results in Anastassiou and Gal (2012)**[16]**. To do this, we notice that f is concave and increases then decreases. In other words, f is piecewise monotone with two concave pieces. To find where f stops increasing and starts decreasing, we solve f's "slope" function (the "slope" function is `diff()` in SymPy). This solution is σ = exp(−1). This, together with the results in that book, leads to the following for concave and two-piece monotone functions on the interval [0, 1]:

- $F(h)$ = f(min(h, σ))−f(0).
- $G(h)$ = f(1−min(h, 1−σ))−f(1).
- $\omega(h)$ = $F(h)$ + $G(h)$.

And thus $\omega(h)$ = 0 if h = 0; $-\lambda$*ln(λ) + (λ − 1)*ln(1 − λ) if λ ≤ e_λ*p(−1); exp(1)*(*λ_ − 1)*ln(1 − λ) + 1)*exp(−1) if λ ≤ 1 − exp(−1); and 2*exp(−1) otherwise.

By plugging sqrt(1/(7*n)) into ω, we get the following for Theorem 2 (assuming n≥0):

- $\varphi(n)$ = [Complicated expression].

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

- $\eta(k)$ = [Complicated expression].

Which matches the η given in the scheme above. That scheme then follows from Theorems 1 and 2, as well as from part 1 of Proposition 1 because f is concave.

```
px=Piecewise((0,Eq(x,0)),(-x*log(x),True))
<h1>Compute piecewise monotone, two-piece concave</h1>

<h1>modulus of continuity</h1>
```

```
sigma=exp(-1) # breakpoint; solves diff(px)
ff=px.subs(x,Min(x,sigma)) - px.subs(x,0)
gg=px.subs(x,1-Min(x,1-sigma)) - px.subs(x,1)
<h1>omega is modulus of continuity</h1>

omega=ff+gg
omega=piecewise_fold(omega.rewrite(Piecewise))
<h1>compute phi according to Theorem 2</h1>

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

eta=summation(phi.subs(n,2**n),(n,k,oo)).subs(k,log(k,2))
for i in range(20):
  try:
    print("eta(2^%d) ~= %s" % (i,eta.subs(k,i).n()))
  except:
    print("eta(2^%d) ~= [FAILED]" % (i))
```

9 License