

# 1 Correctness and Performance Charts

This version of the document is dated 2023-06-13.

The following charts show the correctness of many of the algorithms in “[Bernoulli Factory Algorithms](#)<sup>1</sup>” and show their performance in terms of the number of bits they use on average. For each algorithm, and for each of 100  $\lambda$  values evenly spaced from 0.0001 to 0.9999:

- 500 runs of the algorithm were done. Then...
- The number of bits used by the runs were averaged, as were the return values of the runs (since the return value is either 0 or 1, the mean return value will be in the interval  $[0, 1]$ ). The number of bits used included the number of bits used to produce each coin flip, assuming the coin flip procedure for  $\lambda$  was generated using the `Bernoulli#coin()` method in *bernoulli.py*, which produces that probability in an optimal or near-optimal way.

For each algorithm, if a single run was detected to use more than 5000 bits for a given  $\lambda$ , the entire data point for that  $\lambda$  was suppressed in the charts below.

In addition, for each algorithm, a chart appears showing the minimum number of input coin flips that any fast Bernoulli factory algorithm will need on average to simulate the given function, based on work by Mendo (2019)[<sup>1</sup>]. Note that some functions require a growing number of coin flips as  $\lambda$  approaches 0 or 1. Note that for the 2014, 2016, and 2019 algorithms—

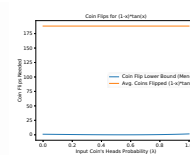
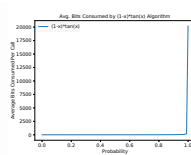
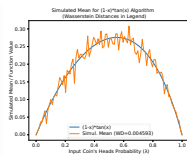
- an  $\epsilon$  of  $1 - (x + c) * 1.001$  was used (or 0.0001 if  $\epsilon$  would be greater than 1), and
- an  $\epsilon$  of  $(x - c) * 0.9995$  for the subtraction variants.

Points with invalid  $\epsilon$  values were suppressed. For the low-mean algorithm, an  $m$  of  $\max(0.49999, xc1.02)$  was used unless noted otherwise.

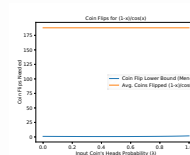
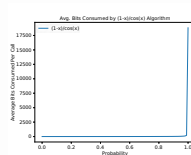
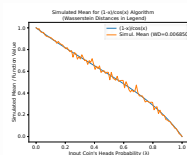
## 1.1 The Charts

Algorithm	Simulated Mean	Average Bits Consumed	Coin Flips
-----------	----------------	-----------------------	------------

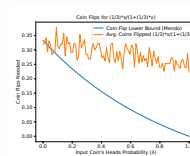
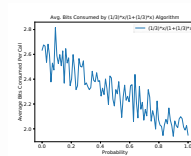
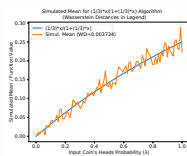
$$(1-x)*\tan(x)$$



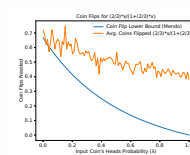
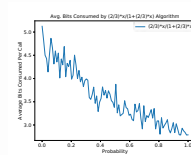
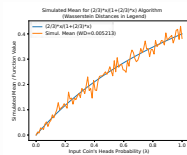
$$(1-x)/\cos(x)$$



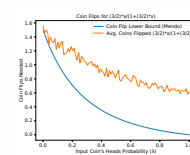
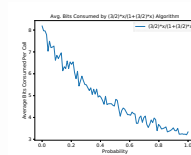
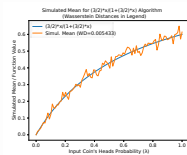
$$(1/3)*x/(1+(1/3)*x)$$



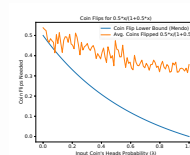
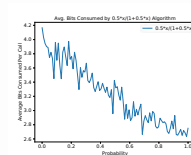
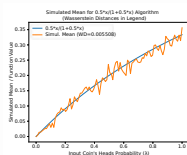
$$(2/3)*x/(1+(2/3)*x)$$



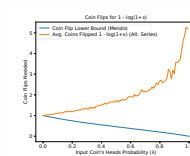
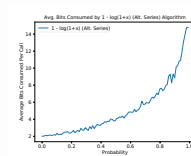
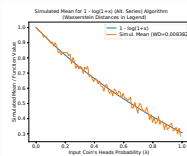
$$(3/2)*x/(1+(3/2)*x)$$



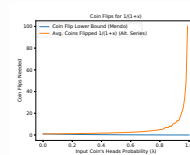
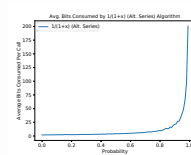
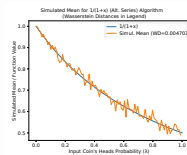
$$0.5*x/(1+0.5*x)$$



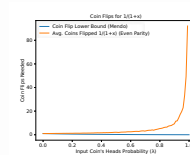
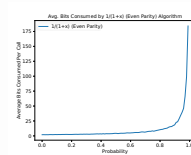
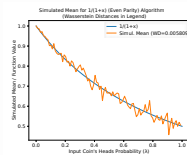
$$1 - \ln(1+x) \text{ (Alt. Series)}$$



$$1/(1+x) \text{ (Alt. Series)}$$



$$1/(1+x) \text{ (Even Parity)}$$



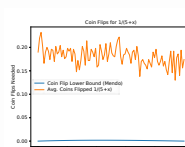
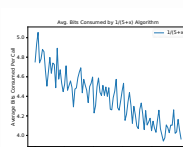
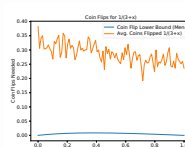
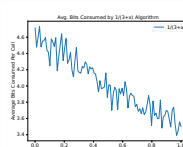
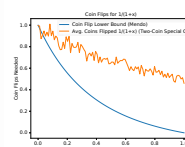
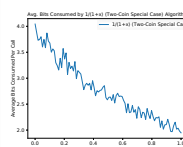
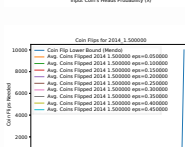
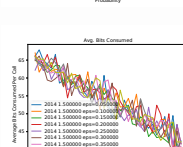
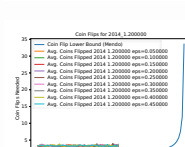
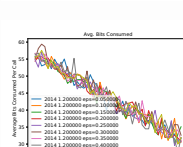
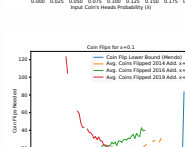
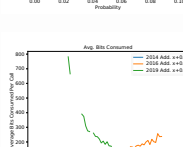
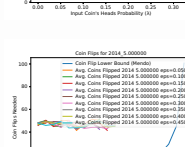
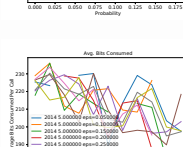
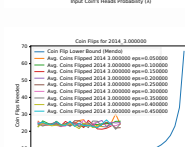
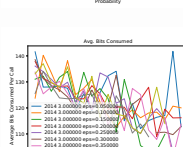
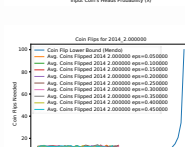
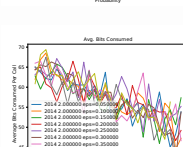
[illegible][illegible]

Figure 1 consists of three subplots showing the performance of the proposed algorithm. The left subplot shows the Displacement Rate / Critical Displacement (Y-axis, 0.0 to 1.0) versus the Avg. Displacement / Average Displacement (X-axis, 0.0 to 1.0). The right subplot shows the CPU Time / Average CPU Time (Y-axis, 0.0 to 80) versus the Avg. Displacement / Average Displacement (X-axis, 0.0 to 1.0). The middle subplot shows the Avg. Displacement / Average Displacement (Y-axis, 0.0 to 1.0) versus the Avg. Displacement / Average Displacement (X-axis, 0.0 to 1.0). The legend for all plots is: 2014 AAS +  $\alpha=0.2$  (blue line), 2014 AAS +  $\alpha=0.3$  (orange line), 2014 AAS +  $\alpha=0.4$  (green line), 2014 AAS +  $\alpha=0.5$  (red line), and 2014 AAS +  $\alpha=0.6$  (black line). The left subplot also includes a legend for the Displacement Rate / Critical Displacement: 0.2 (blue line), 0.3 (orange line), 0.4 (green line), 0.5 (red line), and 0.6 (black line). The middle subplot also includes a legend for the Avg. Displacement / Average Displacement: 0.2 (blue line), 0.3 (orange line), 0.4 (green line), 0.5 (red line), and 0.6 (black line). The right subplot also includes a legend for the CPU Time / Average CPU Time: 0.2 (blue line), 0.3 (orange line), 0.4 (green line), 0.5 (red line), and 0.6 (black line).

Figure 1 consists of two plots. The left plot shows the standard error of the mean (SEM) of the estimated mean of the input variable (x) versus the input CVD's mean probability (x). The right plot shows the average of the estimated mean of the input variable (x) versus the input CVD's mean probability (x). Both plots compare the proposed method (blue line) with the standard method (orange line) for different values of the input CVD's mean probability (x). The proposed method shows a much lower SEM and a more stable average than the standard method.

**Left Plot: Standard Error of the Mean (SEM) of the Estimated Mean of the Input Variable (x) vs. Input CVD's Mean Probability (x)**

Legend:

- 2016 data,  $\alpha = 0.3$
- 2016 data,  $\alpha = 0.2$
- 2016 data,  $\alpha = 0.1$
- 2019 data,  $\alpha = 0.3$
- 2019 data,  $\alpha = 0.2$
- 2019 data,  $\alpha = 0.1$

**Right Plot: Average of the Estimated Mean of the Input Variable (x) vs. Input CVD's Mean Probability (x)**

Legend:

- Cox's Fine Lasso Regression Method
- Avg. Cox's Fitted 2016 data,  $\alpha = 0.3$
- Avg. Cox's Fitted 2016 data,  $\alpha = 0.2$
- Avg. Cox's Fitted 2016 data,  $\alpha = 0.1$
- Avg. Cox's Fitted 2019 data,  $\alpha = 0.3$
- Avg. Cox's Fitted 2019 data,  $\alpha = 0.2$
- Avg. Cox's Fitted 2019 data,  $\alpha = 0.1$

Figure 1 consists of three subplots illustrating the performance of the proposed algorithm across different input data correlations (0.0 to 0.5).

**Left Subplot: Size of the Feature Space**

The y-axis is 'Size of the Feature Space' (log scale) ranging from  $10^0$  to  $10^4$ . The x-axis is 'Input Data Correlation (0.0 to 0.5)'. The legend indicates:

- Proposed (blue line)
- Baseline (orange line)
- Proposed with  $n=5$  (green line)
- Proposed with  $n=10$  (red line)
- Theoretical Bound (black line)

The proposed algorithm (blue line) shows a linear increase in feature space size with correlation, closely following the theoretical bound (black line). The baseline (orange line) shows a much steeper increase, reaching  $10^4$  at correlation 0.5. The proposed algorithm with  $n=5$  (green line) and  $n=10$  (red line) shows a more gradual increase, reaching approximately  $10^2$  at correlation 0.5.

**Middle Subplot: Average Bits Component**

The y-axis is 'Average Bits Component' ranging from 0.0 to 1.0. The x-axis is 'Input Data Correlation (0.0 to 0.5)'. The legend indicates:

- Proposed (blue line)
- Baseline (orange line)
- Proposed with  $n=5$  (green line)
- Proposed with  $n=10$  (red line)

The proposed algorithm (blue line) shows a steady increase in the average bits component, reaching approximately 0.6 at correlation 0.5. The baseline (orange line) shows a much steeper increase, reaching approximately 0.9 at correlation 0.5. The proposed algorithm with  $n=5$  (green line) and  $n=10$  (red line) shows a more gradual increase, reaching approximately 0.4 at correlation 0.5.

**Right Subplot: Gain Ratio for  $n=5$**

The y-axis is 'Gain Ratio for  $n=5$ ' ranging from 0 to 50. The x-axis is 'Input Data Correlation (0.0 to 0.5)'. The legend indicates:

- Gain Ratio for  $n=5$  (blue line)
- Proposed (orange line)
- Baseline (green line)
- Proposed with  $n=10$  (red line)
- Proposed with  $n=5$  (yellow line)

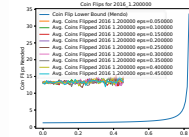
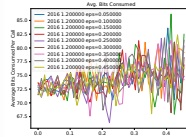
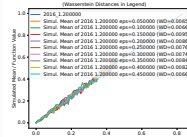
The proposed algorithm (orange line) shows a steady increase in the gain ratio, reaching approximately 25 at correlation 0.5. The baseline (green line) shows a much steeper increase, reaching approximately 45 at correlation 0.5. The proposed algorithm with  $n=10$  (red line) and  $n=5$  (yellow line) shows a more gradual increase, reaching approximately 10 at correlation 0.5.

[illegible][illegible]

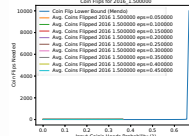
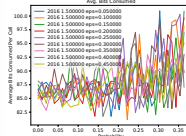
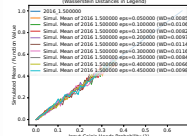
Figure 1 consists of three subplots comparing the proposed method with the baseline. The left subplot, titled "Simulated Results", shows the "Simulated Mean" on the x-axis (ranging from 0 to 100) versus the "Simulated Standard Deviation (x Lognormal)" on the y-axis (ranging from 0 to 100). A diagonal line represents the ideal case where mean equals standard deviation. Data series for various methods are plotted, showing they generally follow the diagonal line. The middle subplot, titled "Avg. Size Comparison", shows the "Average Size (x Lognormal)" on the x-axis (ranging from 0 to 100) versus the "Average Size (x Lognormal)" on the y-axis (ranging from 0 to 100). A diagonal line represents the ideal case. Data series for various methods are plotted, showing they generally follow the diagonal line. The right subplot, titled "Cost Flaps for v2.0", shows the "Cost Flaps (x Lognormal)" on the x-axis (ranging from 0 to 100) versus the "Cost Flaps (x Lognormal)" on the y-axis (ranging from 0 to 100). A diagonal line represents the ideal case. Data series for various methods are plotted, showing they generally follow the diagonal line.

[illegible]

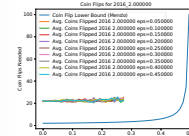
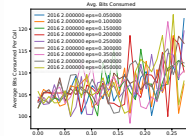
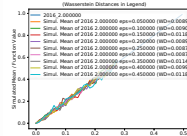
2016 1.200000  
eps=0.050000



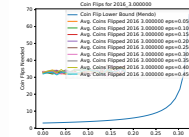
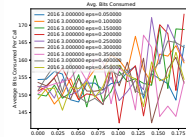
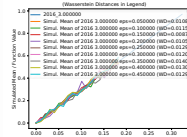
2016 1.500000  
eps=0.050000



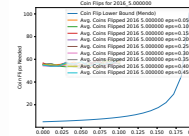
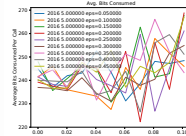
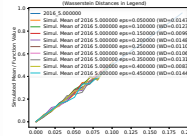
2016 2.000000  
eps=0.050000



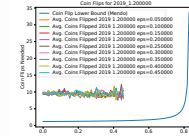
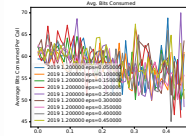
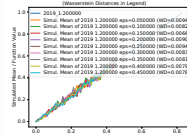
2016 3.000000  
eps=0.050000



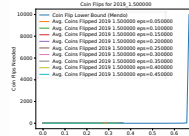
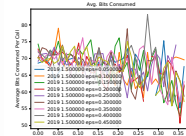
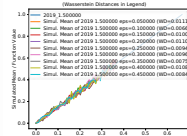
2016 5.000000  
eps=0.050000



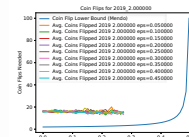
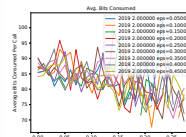
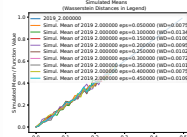
2019 1.200000  
eps=0.050000



2019 1.500000  
eps=0.050000

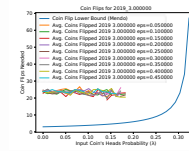
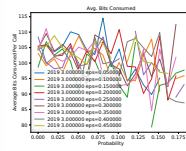
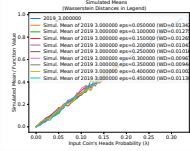


2019 2.000000  
eps=0.050000

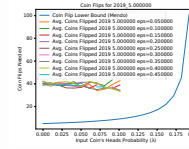
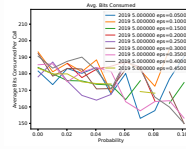
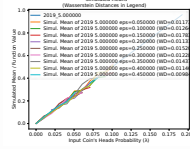


2019 3.000000

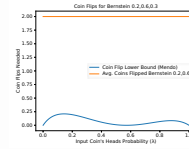
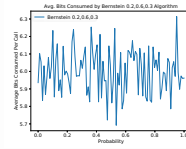
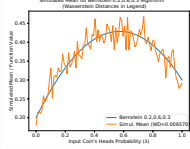
$\epsilon = 0.050000$



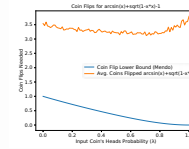
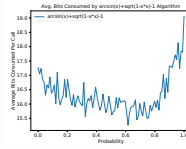
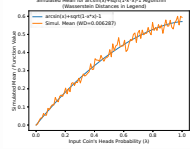
2019 5.000000  
 $\epsilon = 0.050000$



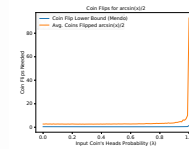
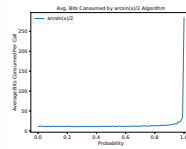
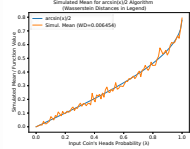
Bernstein  
0.2,0.6,0.3



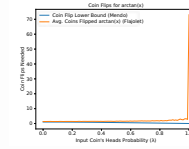
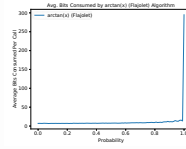
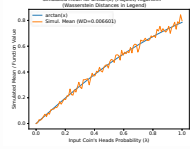
$\arcsin(x) + \sqrt{1 - x^2} - 1$



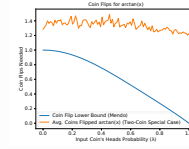
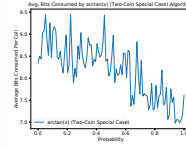
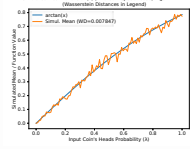
$\arcsin(x)/2$



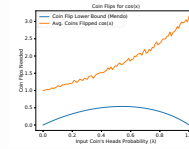
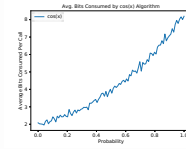
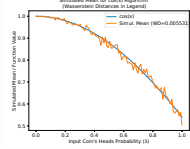
$\arctan(x)$   
(Flajolet)



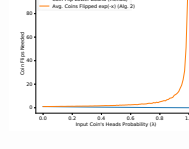
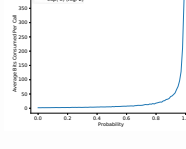
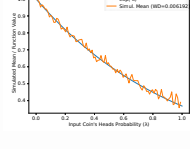
$\arctan(x)$  (Two-Coin Special Case)



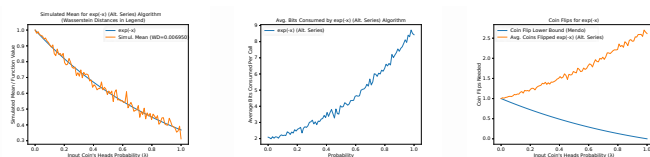
$\cos(x)$



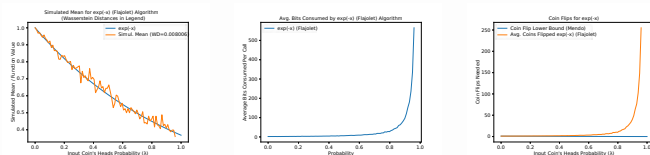
$\exp(-x)$  (Alg. 2)



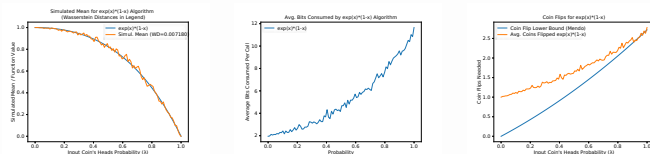
$\exp(-x)$  (Alt. Series)



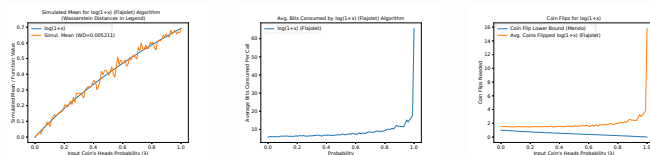
$\exp(-x)$  (Flajolet)



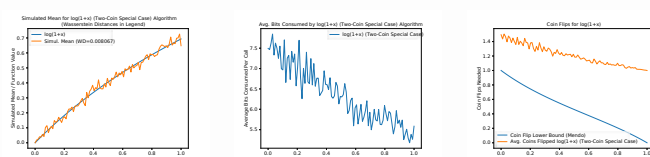
$\exp(x)*(1-x)$



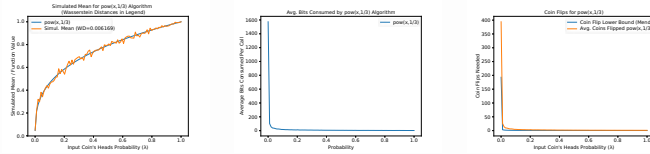
$\ln(1+x)$  (Flajolet)



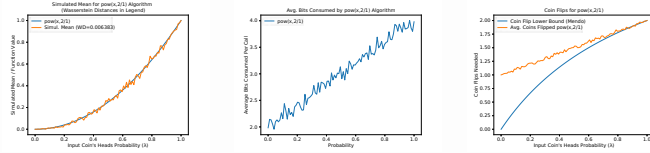
$\ln(1+x)$  (Two-Coin Special Case)



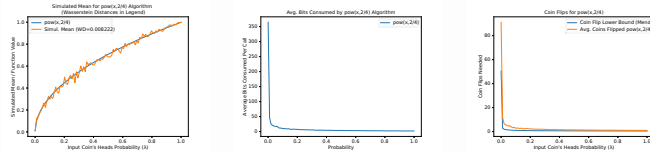
$\text{pow}(x, 1/3)$



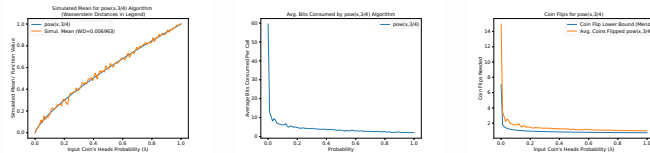
$\text{pow}(x, 2/1)$



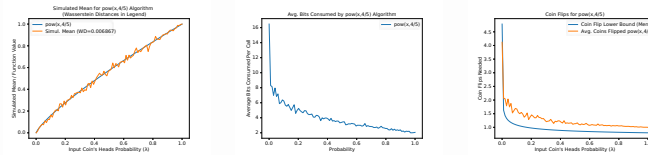
$\text{pow}(x, 2/4)$



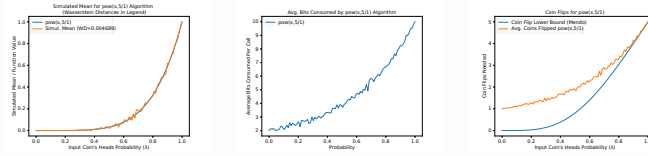
$\text{pow}(x, 3/4)$



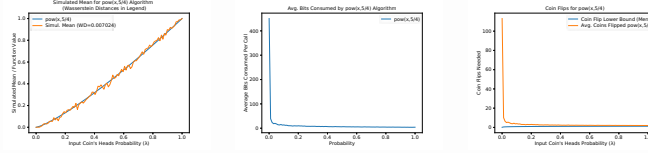
$\text{pow}(x, 4/5)$



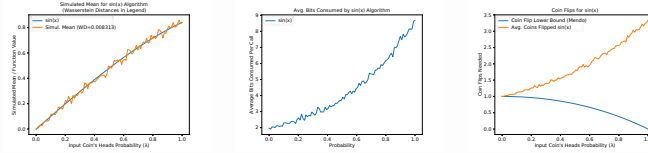
$\text{pow}(x, 5/1)$



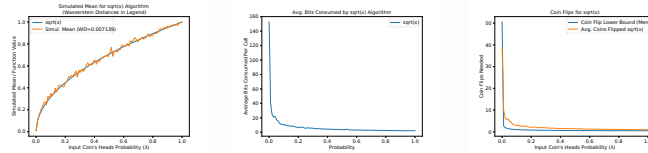
$\text{pow}(x, 5/4)$



$\sin(x)$



$\text{sqrt}(x)$



1. <https://peteroupc.github.io/bernoulli.md>