

# 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—

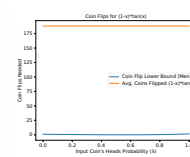
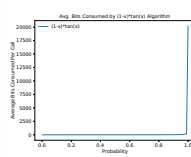
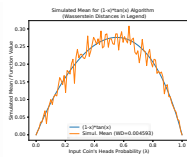
- an  $\epsilon$  of  $1 - (x + c) * 1.001$  was used (or 0.0001 if  $\epsilon$  would be greater than 1), and
- an  $\epsilon$  of  $(x - c) * 0.9995$  for the subtraction variants.

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.

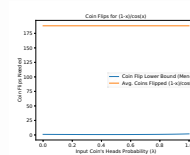
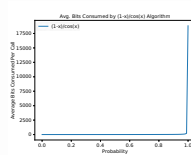
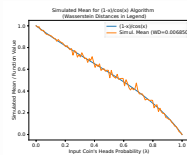
## 1.1 The Charts

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

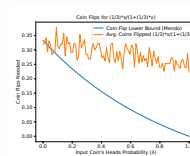
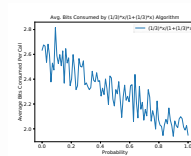
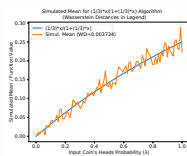
$$(1-x)*\tan(x)$$



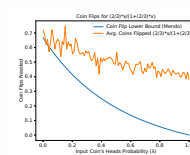
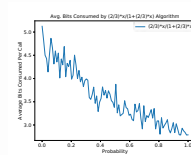
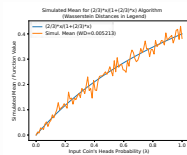
$$(1-x)/\cos(x)$$



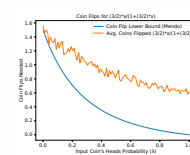
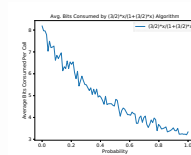
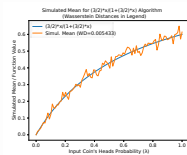
$$(1/3)*x/(1+(1/3)*x)$$



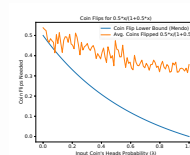
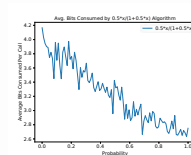
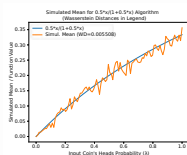
$$(2/3)*x/(1+(2/3)*x)$$



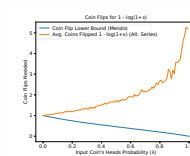
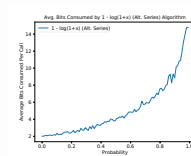
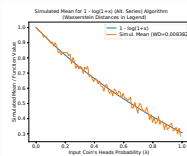
$$(3/2)*x/(1+(3/2)*x)$$



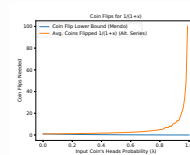
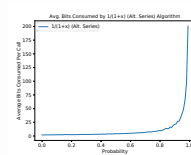
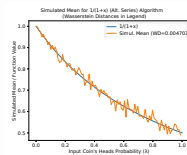
$$0.5*x/(1+0.5*x)$$



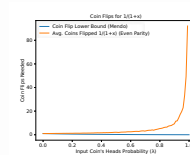
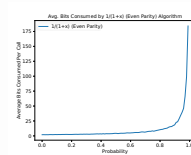
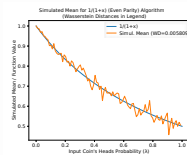
$$1 - \ln(1+x) \text{ (Alt. Series)}$$



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



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



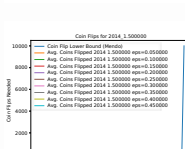
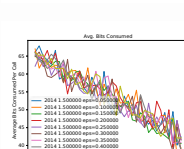
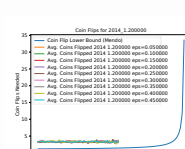
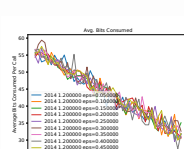
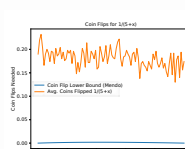
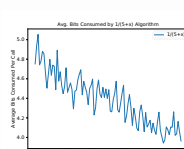
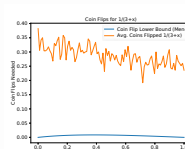
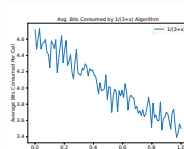
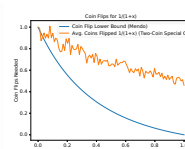
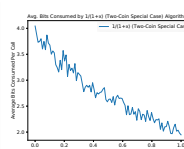
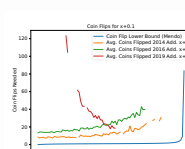
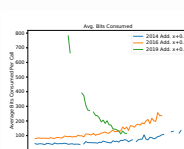
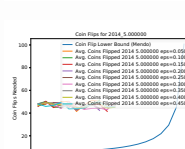
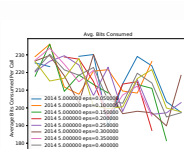
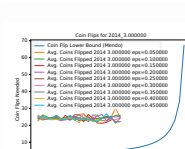
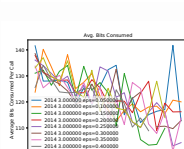
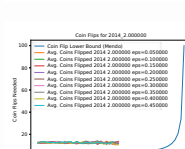
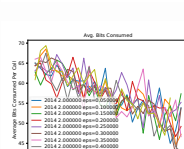
[illegible]

Figure 1 consists of three subplots. The left subplot shows the 'Displacement Rate / Critical Displacement' on the y-axis (ranging from 0.0 to 1.0) versus the 'Avg. Displacement / Average Displacement' on the x-axis (ranging from 0.0 to 1.0). It includes a legend for different values of  $\alpha$ : 0.2 (blue), 0.3 (orange), 0.4 (green), 0.5 (red), and 0.6 (purple). The middle subplot shows the 'Avg. Displacement / Average Displacement' on the y-axis (ranging from 0.0 to 1.0) versus the 'Avg. Displacement / Average Displacement' on the x-axis (ranging from 0.0 to 1.0). It includes a legend for different values of  $\alpha$ : 0.2 (blue), 0.3 (orange), 0.4 (green), 0.5 (red), and 0.6 (purple). The right subplot shows the 'CPU Time / Average CPU Time' on the y-axis (ranging from 0.0 to 1.0) versus the 'Avg. Displacement / Average Displacement' on the x-axis (ranging from 0.0 to 1.0). It includes a legend for different values of  $\alpha$ : 0.2 (blue), 0.3 (orange), 0.4 (green), 0.5 (red), and 0.6 (purple).

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 consistently shows lower SEM and higher average values than the standard method.

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:** Shows the **Size of the Feature Space** (log scale) versus **Input Data Correlation (0.0 to 0.5)**. The proposed algorithm (blue line) maintains a low feature space size, while the baseline methods (orange, green, red lines) show a significant increase in feature space size as correlation increases. The black line represents the theoretical bound.
- Middle Subplot:** Shows the **Average Bits Component** versus **Input Data Correlation (0.0 to 0.5)**. The proposed algorithm (blue line) maintains a low average bits component, while the baseline methods (orange, green, red lines) show a significant increase in average bits component as correlation increases. The black line represents the theoretical bound.
- Right Subplot:** Shows the **Gain Ratio for  $n=5$**  versus **Input Data Correlation (0.0 to 0.5)**. The proposed algorithm (blue line) maintains a high gain ratio, while the baseline methods (orange, green, red lines) show a significant decrease in gain ratio as correlation increases. The black line represents the theoretical bound.

[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.

Figure 1 consists of three subplots. The left subplot, titled "Simulated Means", shows the distribution of simulated means for various parameters (alpha, beta, gamma, delta, epsilon, zeta, eta, theta, iota, kappa, lambda, mu, nu, xi, omicron, pi, rho, sigma, tau, upsilon, phi, chi, psi, omega, and unknown) against the "Observed Statistics in Logspace". The middle subplot, titled "Avg. Bits Consumed", shows the average bits consumed for the same parameters against the "Average of Observed Log-CFs". The right subplot, titled "Coin Flips for alpha=0", shows the number of coin flips required for different parameters against the "Average of Observed Log-CFs".

[illegible]

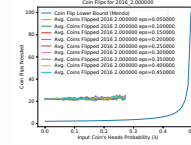
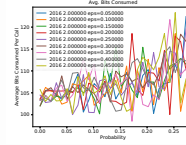
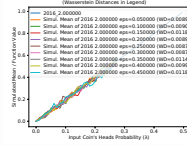
2016 1.200000  
eps=0.050000



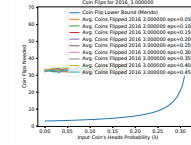
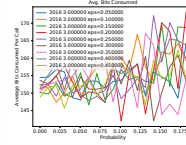
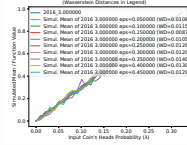
2016 1.500000  
eps=0.050000



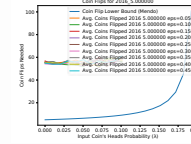
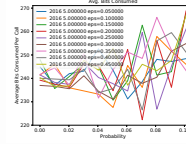
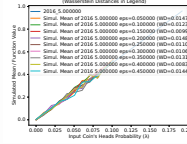
2016 2.000000  
eps=0.050000



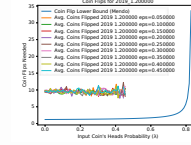
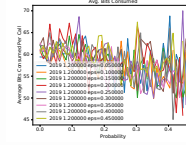
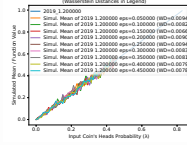
2016 3.000000  
eps=0.050000



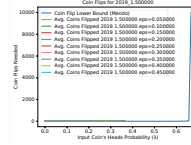
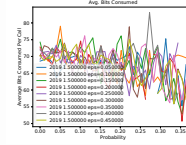
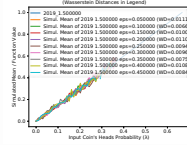
2016 5.000000  
eps=0.050000



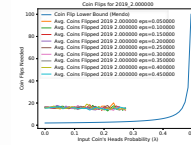
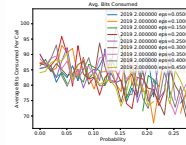
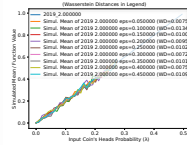
2019 1.200000  
eps=0.050000



2019 1.500000  
eps=0.050000

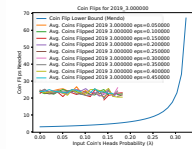


2019 2.000000  
eps=0.050000

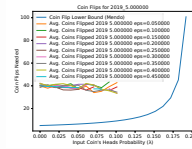


2019 3.000000

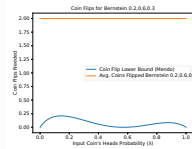
$\epsilon=0.050000$



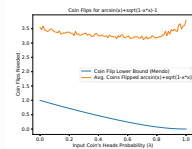
2019 5.000000  
 $\epsilon=0.050000$



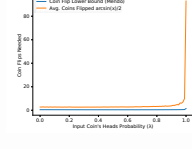
Bernstein  
0.2,0.6,0.3



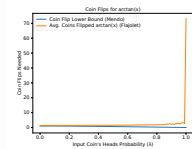
$\arcsin(x)+\sqrt{1-x^2}-1$



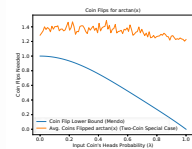
$\arcsin(x)/2$



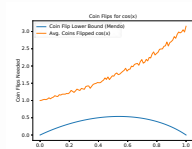
$\arctan(x)$   
(Flajolet)



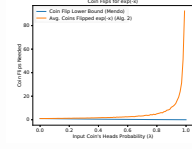
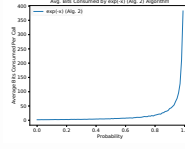
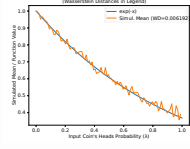
$\arctan(x)$  (Two-Coin Special Case)



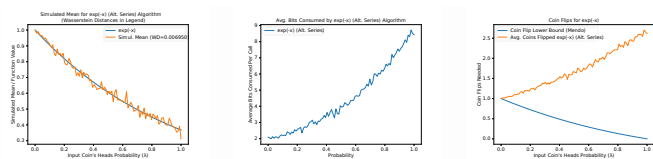
$\cos(x)$



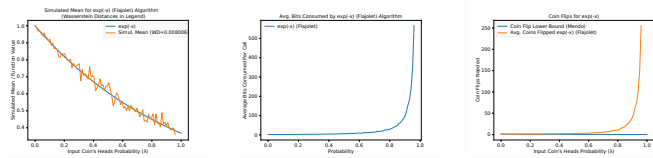
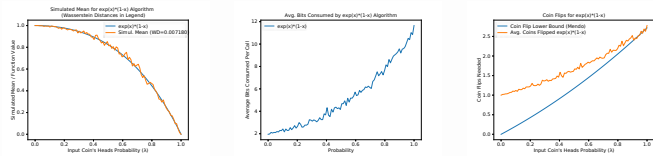
$\exp(-x)$  (Alg. 2)



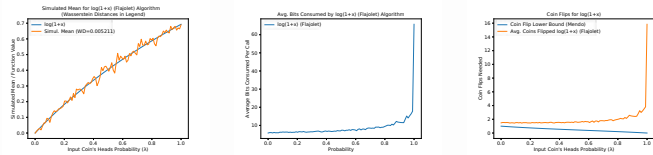
exp(-x) (Alt.  
Series)



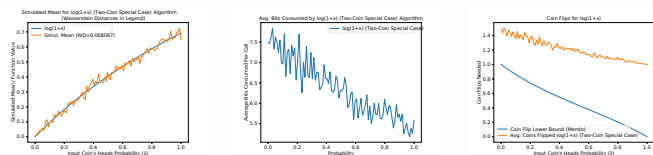
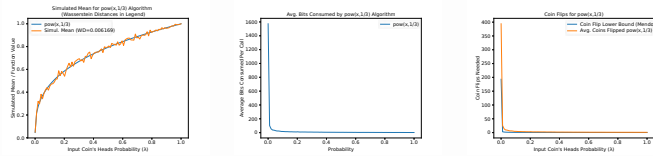
exp(-x) (Flajolet)


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


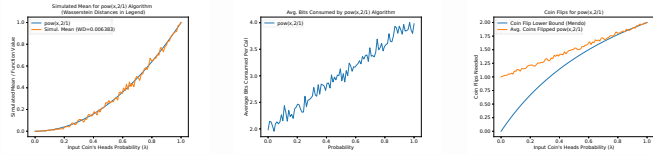
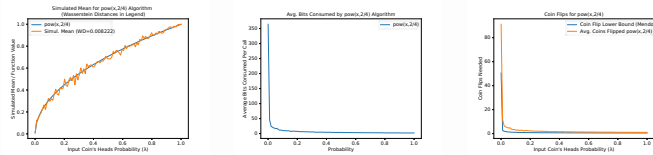
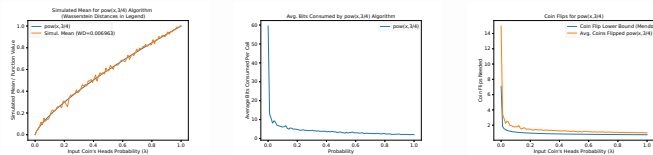
$\ln(1+x)$  (Flajolet)



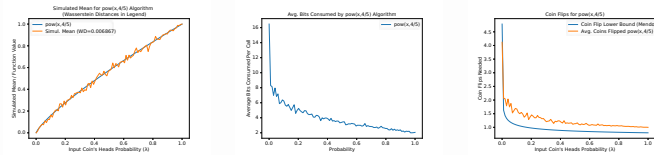
### ln(1+x) (Two-Coin Special Case)

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

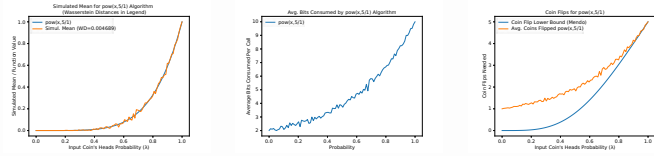
pow(x,2/1)

 $\text{pow}(x, 2/4)$  $\text{pow}(x, 3/4)$ 

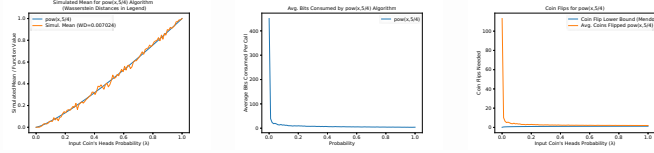
$\text{pow}(x, 4/5)$



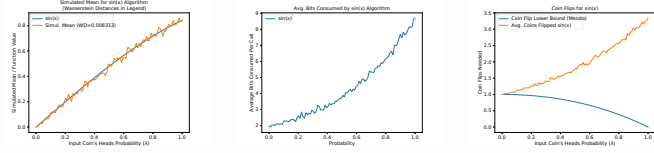
$\text{pow}(x, 5/1)$



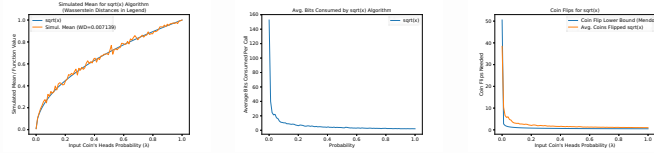
$\text{pow}(x, 5/4)$



$\sin(x)$



$\text{sqrt}(x)$



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