

# 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.

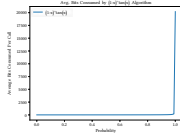
For each algorithm, if a single run was detected to use more than 5000 bits for a given  $\lambda$ , the entire data point for that  $\lambda$  was suppressed in the charts below.

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)[^1]. 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)			

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

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

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

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

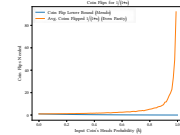
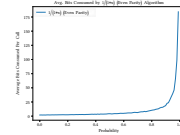
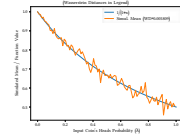
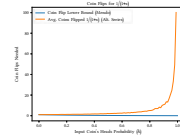
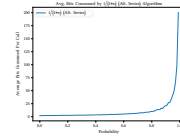
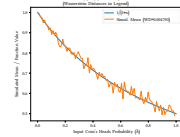
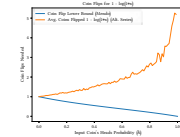
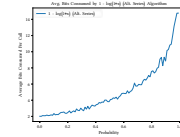
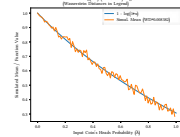
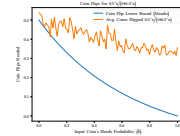
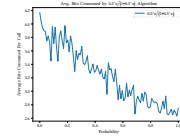
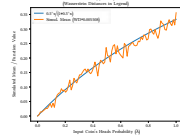
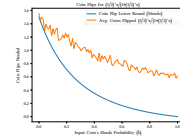
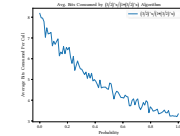
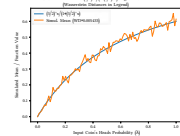
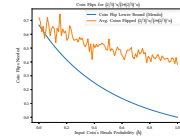
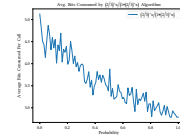
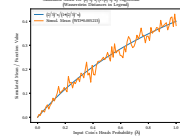
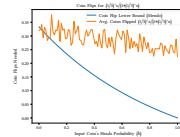
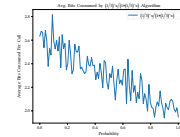
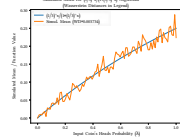
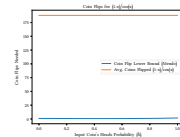
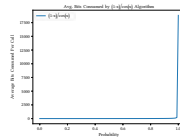
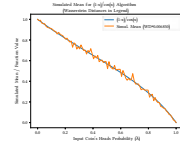
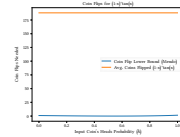
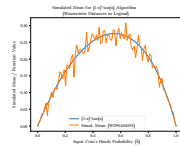
$$0.5^*x/(1+0.5^*x)$$

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

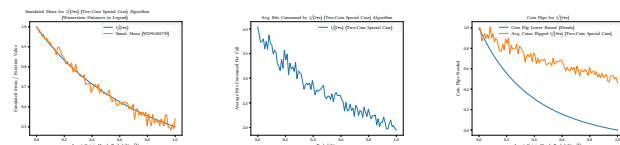
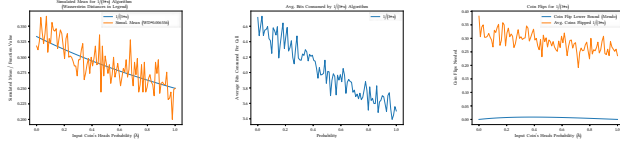
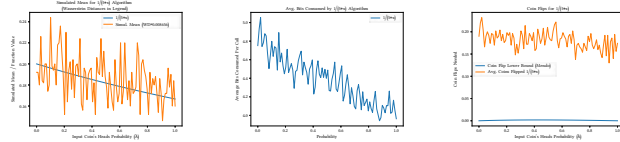
$$1/(1+x) \text{ (Alt. Series)}$$

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

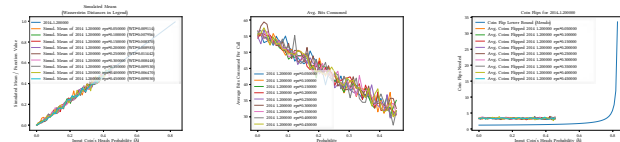
$$1/(1+x) \text{ (Two-Coin Special)}$$



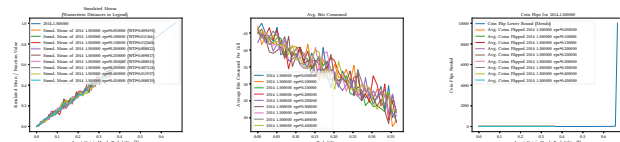
Case)


$$1/(3+x)$$

$$1/(5+x)$$


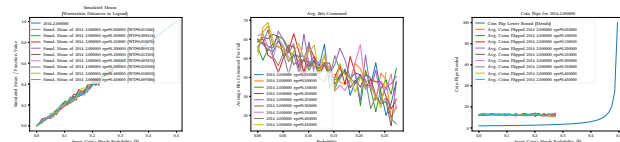
2014 1.200000  
eps=0.050000



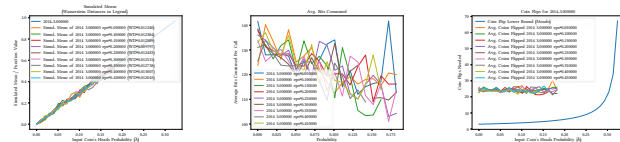
2014 1.500000  
eps=0.050000



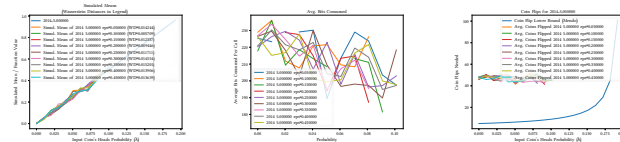
```
2014 2.000000
eps=0.050000
```



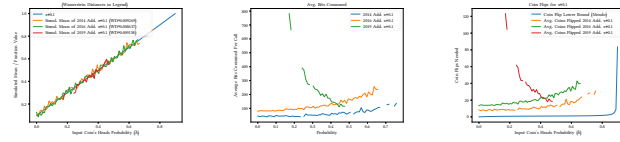
```
2014 3.000000
eps=0.050000
```



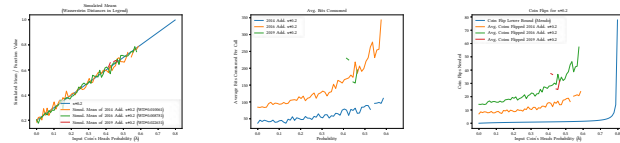
```
2014 5.000000
eps=0.050000
```



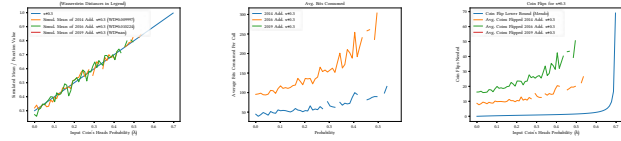
2014 Add. x+0.1



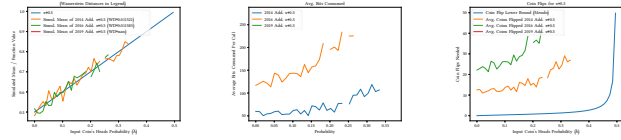
2014 Add. x+0.2



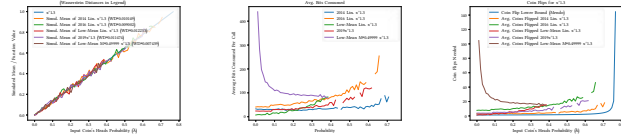
2014 Add.  $x+0.3$



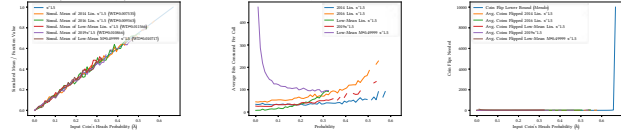
2014 Add.  $x+0.5$



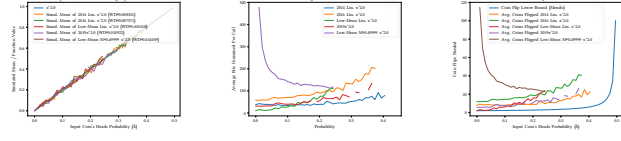
2014 Lin.  $x*1.3$



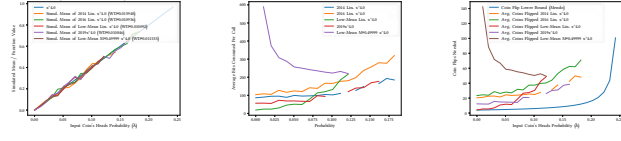
2014 Lin.  $x*1.5$



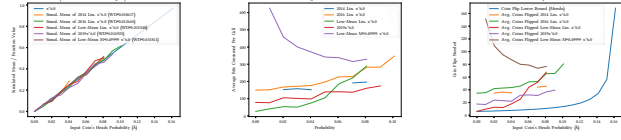
2014 Lin.  $x*2.0$



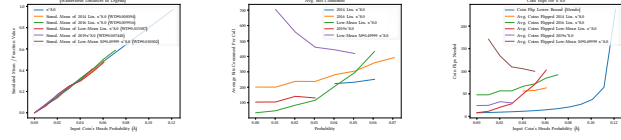
2014 Lin.  $x*4.0$



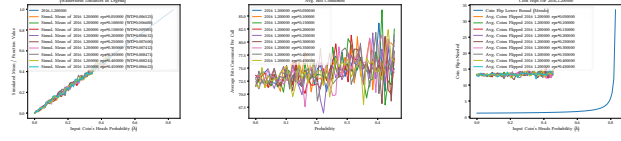
2014 Lin.  $x*6.0$



2014 Lin.  $x*8.0$



2016 1.200000  
eps=0.050000



[illegible]

Figure 1 consists of three subplots labeled (a), (b), and (c).  
 Subplot (a) is titled 'Normalized Error (Relative Error)' and shows 'Normalized Error (Relative Error)' on the y-axis (ranging from 0.0 to 0.4) versus 'Number of Iterations (1e3)' on the x-axis (ranging from 0 to 100). It compares 'Proposed Algorithm' (green line with circles) and 'Reference Algorithm' (blue line with circles). Both algorithms show a decreasing trend in error, with the proposed algorithm reaching a lower error faster.  
 Subplot (b) is titled 'Avg. Rel. Error (1e3)' and shows 'Avg. Rel. Error (1e3)' on the y-axis (ranging from 0.0 to 2.0) versus 'Avg. Rel. Error (1e3)' on the x-axis (ranging from 0.0 to 2.0). It compares 'Proposed Algorithm' (green line with circles) and 'Reference Algorithm' (blue line with circles). The proposed algorithm shows a more stable and lower error compared to the reference algorithm.  
 Subplot (c) is titled 'Avg. Rel. Error (1e3)' and shows 'Avg. Rel. Error (1e3)' on the y-axis (ranging from 0.0 to 2.0) versus 'Avg. Rel. Error (1e3)' on the x-axis (ranging from 0.0 to 2.0). It compares 'Proposed Algorithm' (green line with circles) and 'Reference Algorithm' (blue line with circles). The proposed algorithm shows a more stable and lower error compared to the reference algorithm.

[illegible]

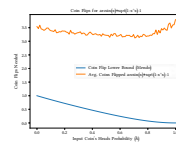
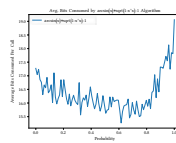
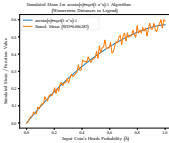
Figure 1 consists of three subplots. Subplot (a) is titled 'Normalized Error' and shows the error on a logarithmic scale from  $10^{-1}$  to  $10^{-5}$  versus iteration from 0 to 1000. It compares the proposed algorithm (blue line) with several other algorithms: ADMM (red), FISTA (green), NIST (yellow), and others. The proposed algorithm shows the fastest convergence. Subplot (b) is titled 'Comparison of the proposed algorithm with other algorithms for the 100000th iteration' and shows the error on a linear scale from 0 to 1000 versus iteration from 0 to 1000. It compares the proposed algorithm (blue line) with several other algorithms: ADMM (red), FISTA (green), NIST (yellow), and others. The proposed algorithm shows the fastest convergence. Subplot (c) is titled 'Comparison of the proposed algorithm with other algorithms for the 100000th iteration' and shows the error on a linear scale from 0 to 1000 versus iteration from 0 to 1000. It compares the proposed algorithm (blue line) with several other algorithms: ADMM (red), FISTA (green), NIST (yellow), and others. The proposed algorithm shows the fastest convergence.

Figure 1 consists of three subplots showing the performance of the proposed model over 10,000 iterations. Subplot (a) shows the Normalized Error (Y-axis, 0.00 to 0.05) versus the Number of Iterations (X-axis, 0 to 10,000). The error decreases for all models, with the proposed model (red line) showing the fastest convergence. Subplot (b) shows the Average Accuracy (Y-axis, 0.00 to 1.00) versus the Number of Iterations (X-axis, 0 to 10,000). The proposed model (red line) achieves the highest accuracy, reaching approximately 0.95. Subplot (c) shows the Classification Accuracy (Y-axis, 0.00 to 1.00) versus the Number of Iterations (X-axis, 0 to 10,000). The proposed model (red line) achieves the highest accuracy, reaching approximately 0.95.

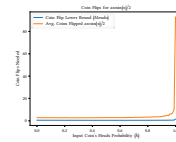
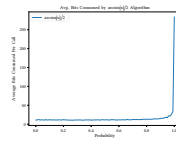
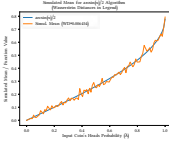
[illegible][illegible][illegible]

Figure 1 consists of three subplots. Subplot (a) is titled 'Normalized MSE for Bayesian C-VAR2 Algorithm' and shows 'Normalized MSE / Frequency (Hz)' on the y-axis (0.00 to 0.04) versus 'Frequency (Hz)' on the x-axis (0 to 1000). It compares 'Exact Bayesian C-VAR2' (orange line with dots) and 'Approx. using C-VAR2.0.0.0' (blue line with dots). Subplot (b) is titled 'Normalized MSE for Bayesian C-VAR2.0.0.0 Algorithm' and shows 'Normalized MSE / Frequency (Hz)' on the y-axis (0.00 to 0.04) versus 'Frequency (Hz)' on the x-axis (0 to 1000). It compares 'Exact Bayesian C-VAR2.0.0.0' (orange line with dots) and 'Approx. using C-VAR2.0.0.0' (blue line with dots). Subplot (c) is titled 'Normalized MSE for Bayesian C-VAR2.0.0.0 Algorithm' and shows 'Normalized MSE / Frequency (Hz)' on the y-axis (0.00 to 0.04) versus 'Frequency (Hz)' on the x-axis (0 to 1000). It compares 'Exact Bayesian C-VAR2.0.0.0' (orange line with dots) and 'Approx. using C-VAR2.0.0.0' (blue line with dots).

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

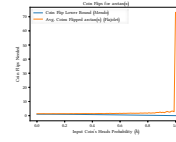
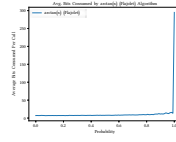
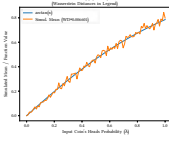


$$\arcsin(x)/2$$



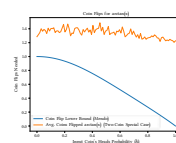
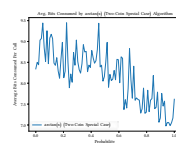
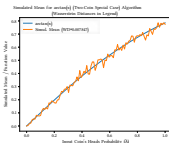
$$\arctan(x)$$

(Flajolet)

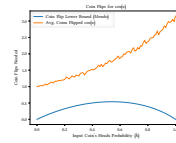
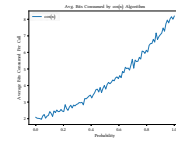
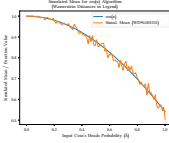


$$\arctan(x)$$

(Two-Coin Special Case)

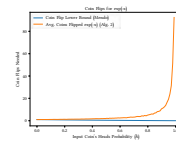
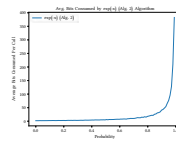
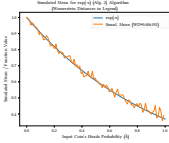


$$\cos(x)$$



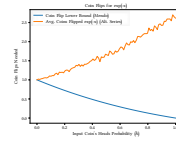
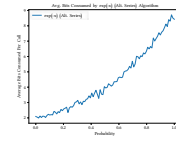
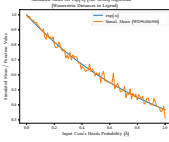
$$\exp(-x)$$

(Alg. 2)



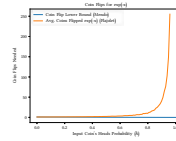
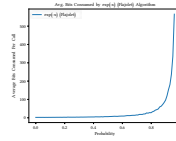
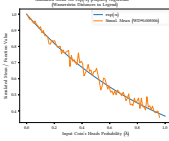
$$\exp(-x)$$

(Alt. Series)

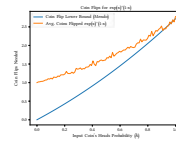
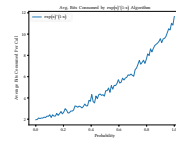
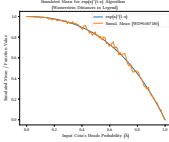


$$\exp(-x)$$

(Flajolet)



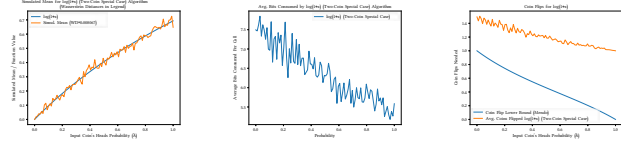
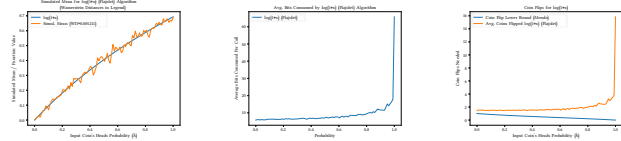
$$\exp(x) * (1-x)$$



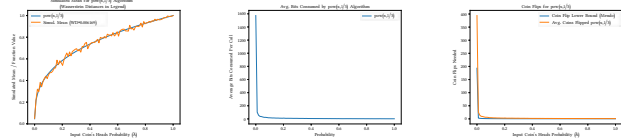
$$\ln(1+x)$$

(Flajolet)

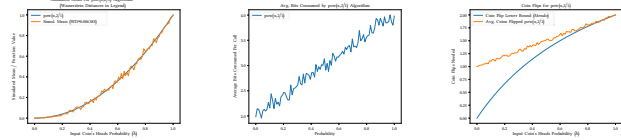
$\ln(1+x)$  (Two-Coin Special Case)



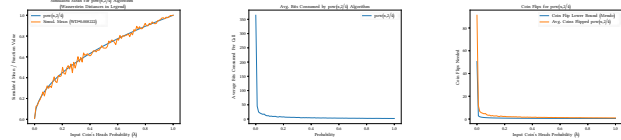
$\text{pow}(x, 1/3)$



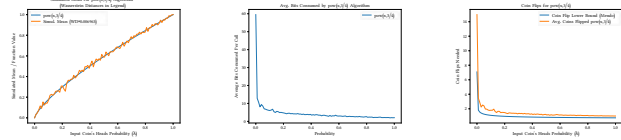
$\text{pow}(x, 2/1)$



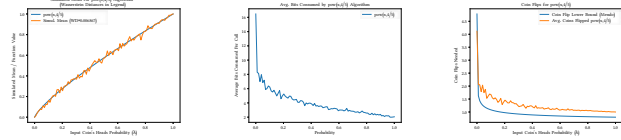
$\text{pow}(x, 2/4)$



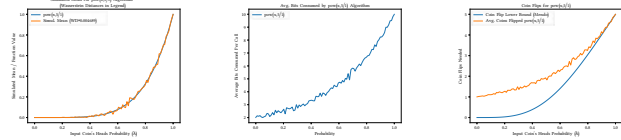
$\text{pow}(x, 3/4)$



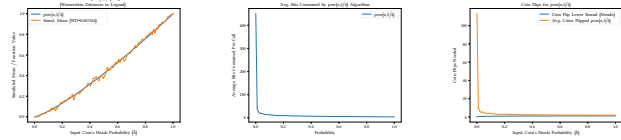
$\text{pow}(x, 4/5)$



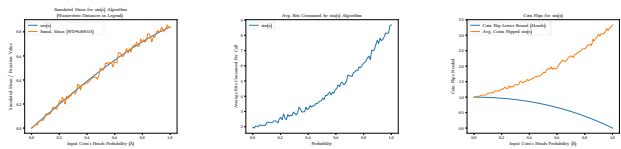
$\text{pow}(x, 5/1)$



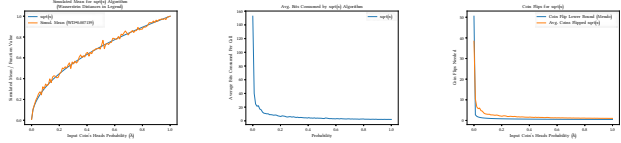
$\text{pow}(x, 5/4)$



sin(x)



sqrt(x)



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