

# 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—

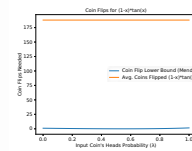
- an  $\epsilon$  of  $1 - (x + c) * 1.001$  was used (or 0.0001 if  $\epsilon$  would be greater than 1), and
- an  $\epsilon$  of  $(x - c) * 0.9995$  for the subtraction variants.

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.

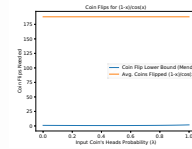
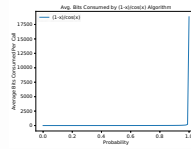
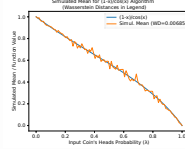
## 1.1 The Charts

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

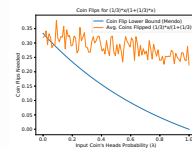
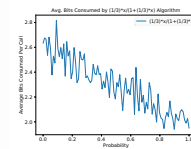
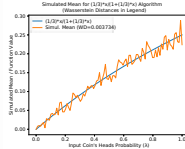
$$(1-x)*\tan(x)$$



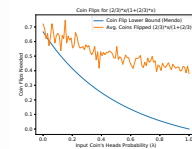
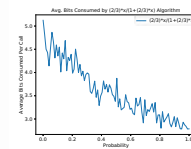
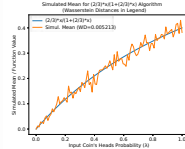
$$(1-x)/\cos(x)$$



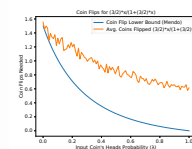
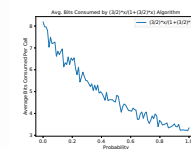
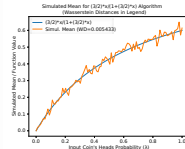
$$(1/3)*x/(1+(1/3)*x)$$



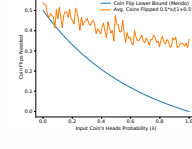
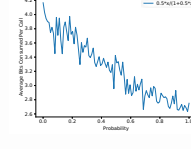
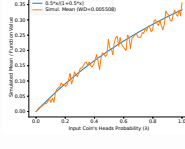
$$(2/3)*x/(1+(2/3)*x)$$



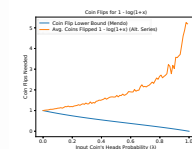
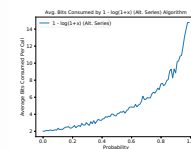
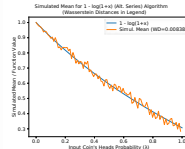
$$(3/2)*x/(1+(3/2)*x)$$



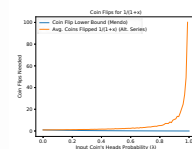
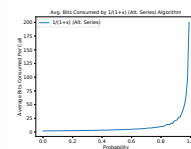
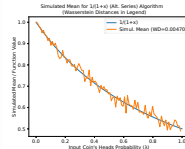
$$0.5*x/(1+0.5*x)$$



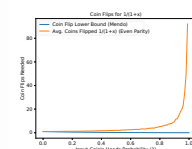
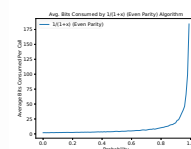
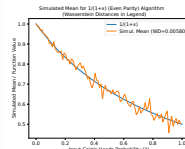
$$1 - \ln(1+x) \text{ (Alt. Series)}$$

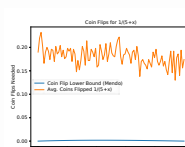
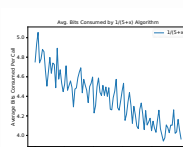
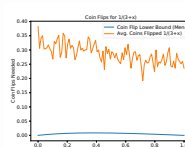
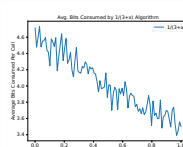
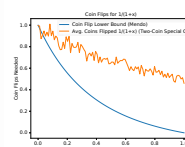
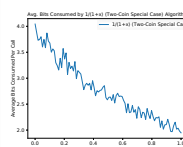


$$1/(1+x) \text{ (Alt. Series)}$$



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

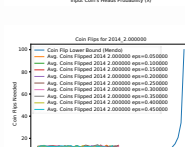
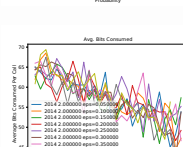
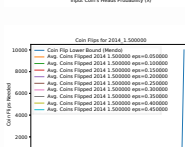
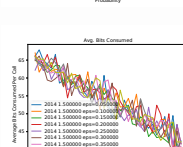
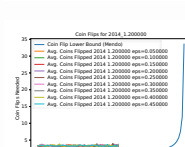
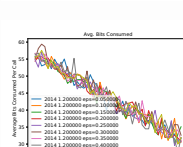




**Simulated Mean Function Value**

**(Wasserstein Distances in Legend)**

- 2014, 1.3500000
- Simul. Mean of 2014, 1.2000000  $\alpha=0.050000$  [WD=0.0691]
- Simul. Mean of 2014, 1.2000000  $\alpha=0.100000$  [WD=0.0707]
- Simul. Mean of 2014, 1.2000000  $\alpha=0.150000$  [WD=0.0683]
- Simul. Mean of 2014, 1.2000000  $\alpha=0.200000$  [WD=0.0690]
- Simul. Mean of 2014, 1.2000000  $\alpha=0.250000$  [WD=0.0714]
- Simul. Mean of 2014, 1.2000000  $\alpha=0.300000$  [WD=0.0694]
- Simul. Mean of 2014, 1.2000000  $\alpha=0.350000$  [WD=0.0695]
- Simul. Mean of 2014, 1.2000000  $\alpha=0.400000$  [WD=0.0664]
- Simul. Mean of 2014, 1.2000000  $\alpha=0.450000$  [WD=0.0690]



The graph displays simulated means for various waterbody distances in meters for the year 2014. Each line represents a different 'apeps' value, with its corresponding Root-Square Deviation (RSD) indicated in parentheses next to the label.

Line Color	Simulation Label	RSD (%)
Blue	Simul. Mean of 2014 3.0000000	
Orange	Simul. Mean of 2014 3.0000000 apeps=0.050000	(RSD=0.0112)
Red	Simul. Mean of 2014 3.0000000 apeps=0.100000	(RSD=0.0127)
Purple	Simul. Mean of 2014 3.0000000 apeps=0.150000	(RSD=0.0129)
Dark Purple	Simul. Mean of 2014 3.0000000 apeps=0.200000	(RSD=0.0097)
Light Blue	Simul. Mean of 2014 3.0000000 apeps=0.250000	(RSD=0.0124)
Green	Simul. Mean of 2014 3.0000000 apeps=0.300000	(RSD=0.0121)
Yellow-Green	Simul. Mean of 2014 3.0000000 apeps=0.350000	(RSD=0.0123)
Yellow	Simul. Mean of 2014 3.0000000 apeps=0.400000	(RSD=0.0127)
Cyan	Simul. Mean of 2014 3.0000000 apeps=0.450000	(RSD=0.0120)

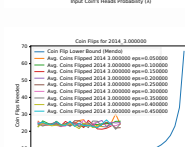
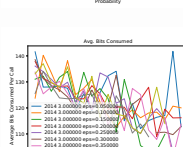
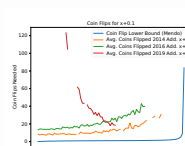
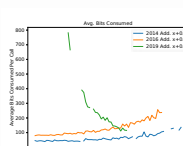
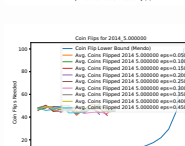
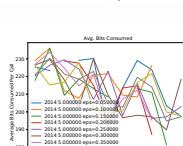


Figure 10 is a line graph showing the Simulated Mean Function Value (Y-axis, ranging from 0.0 to 1.0) versus the Simulated Mean (Waterstein Distances in Legend) (X-axis, ranging from 0.0 to 1.0). The graph displays 10 lines, each representing a different combination of year (2014, 2015, 2016) and mean (5.000000, 5.000000, 5.000000, 5.000000, 5.000000, 5.000000, 5.000000, 5.000000, 5.000000, 5.000000). The lines are color-coded and labeled with their respective Waterstein Distances (WD) and R-squared values (R<sup>2</sup>). The legend indicates that the lines represent the Simulated Mean Function Value for each combination of year and mean.

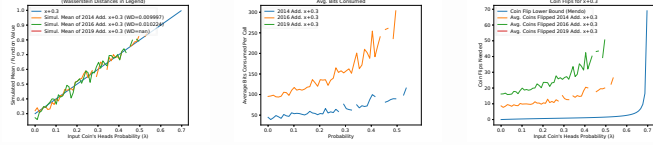
Year	Mean	WD	R <sup>2</sup>
2014	5.000000	0.000000	0.000000
2015	5.000000	0.000000	0.000000
2016	5.000000	0.000000	0.000000
2014	5.000000	0.000000	0.000000
2015	5.000000	0.000000	0.000000
2016	5.000000	0.000000	0.000000
2014	5.000000	0.000000	0.000000
2015	5.000000	0.000000	0.000000
2016	5.000000	0.000000	0.000000
2014	5.000000	0.000000	0.000000



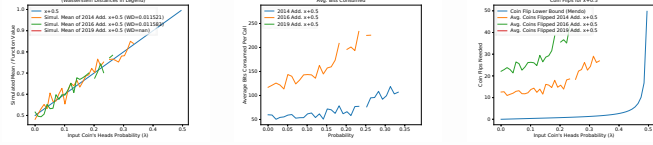
2014 Add.  $x+0.2$



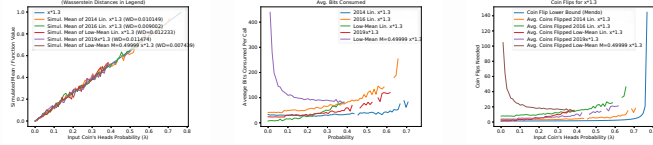
2014 Add.  $x+0.3$



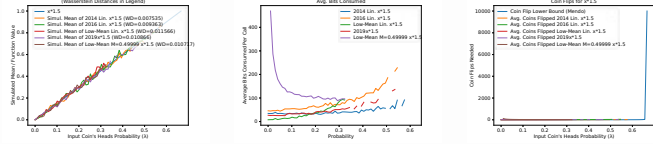
2014 Add.  $x+0.5$



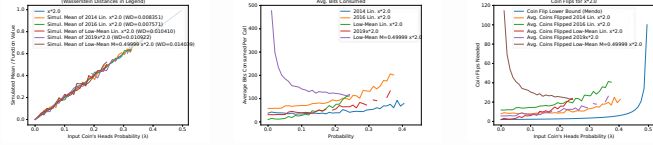
2014 Lin.  $x*1.3$



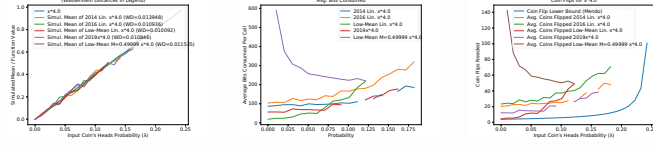
2014 Lin.  $x*1.5$



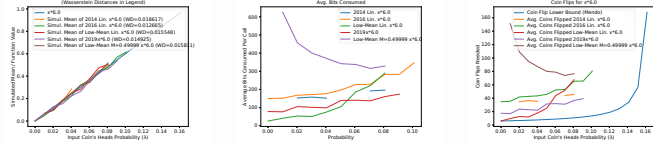
2014 Lin.  $x*2.0$



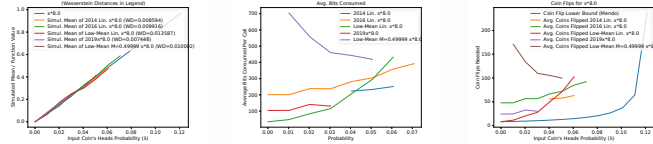
2014 Lin.  $x*4.0$



2014 Lin.  $x*6.0$



2014 Lin.  $x*8.0$



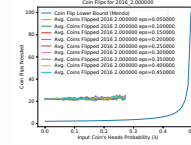
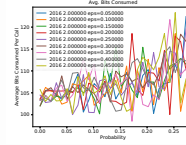
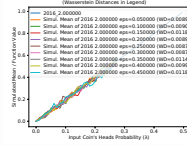
2016 1.200000  
eps=0.050000



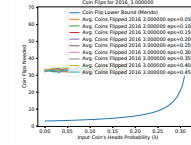
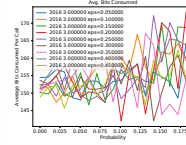
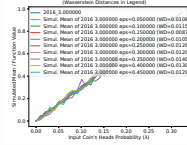
2016 1.500000  
eps=0.050000



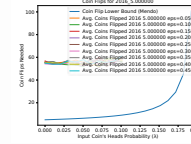
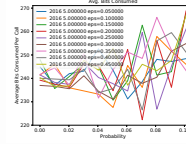
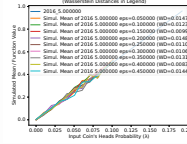
2016 2.000000  
eps=0.050000



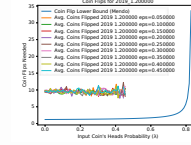
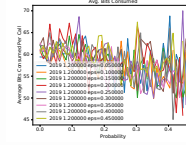
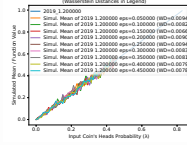
2016 3.000000  
eps=0.050000



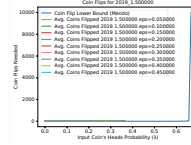
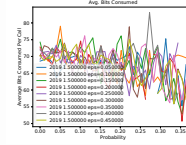
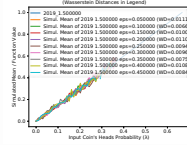
2016 5.000000  
eps=0.050000



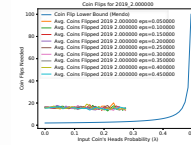
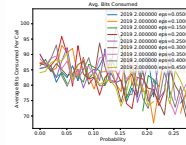
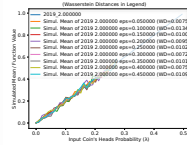
2019 1.200000  
eps=0.050000



2019 1.500000  
eps=0.050000

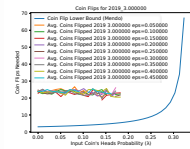


2019 2.000000  
eps=0.050000

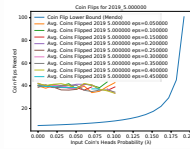


2019 3.000000

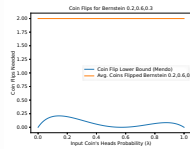
$\epsilon=0.050000$



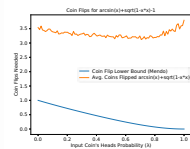
2019 5.000000  
 $\epsilon=0.050000$



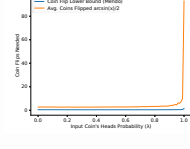
Bernstein  
0.2,0.6,0.3



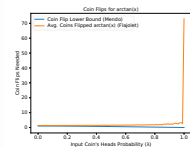
$\arcsin(x)+\sqrt{1-x^2}-1$



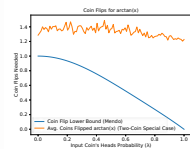
$\arcsin(x)/2$



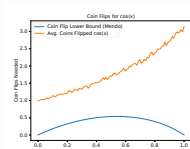
$\arctan(x)$   
(Flajolet)



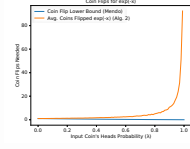
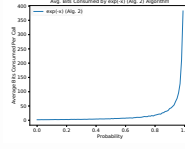
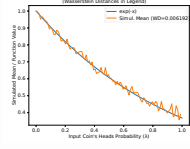
$\arctan(x)$  (Two-Coin Special Case)



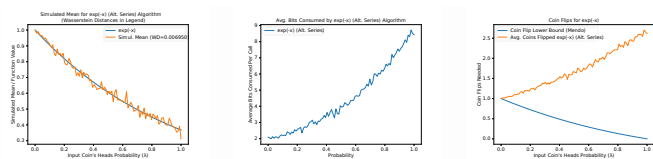
$\cos(x)$



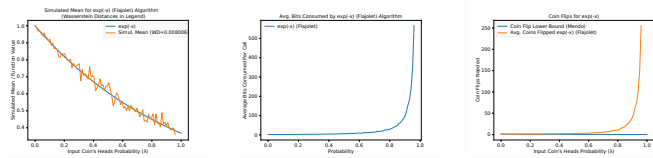
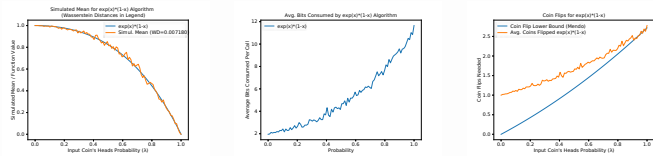
$\exp(-x)$  (Alg. 2)



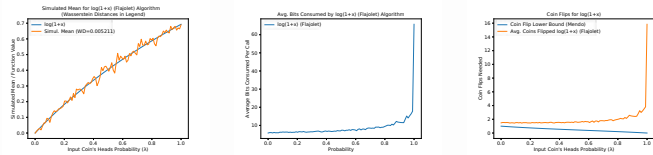
exp(-x) (Alt.  
Series)



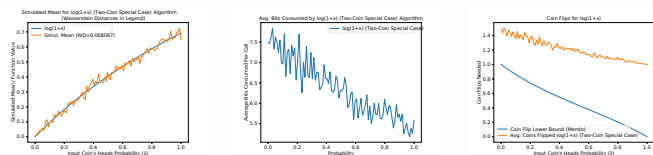
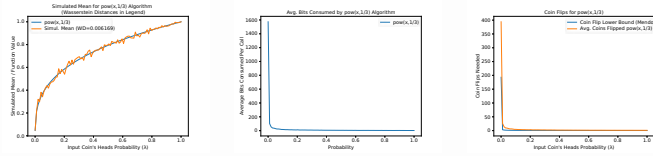
exp(-x) (Flajolet)


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


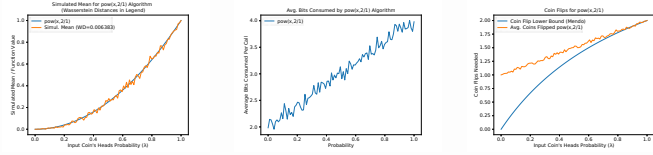
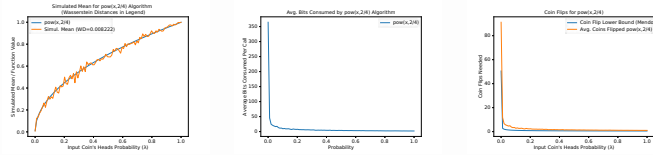
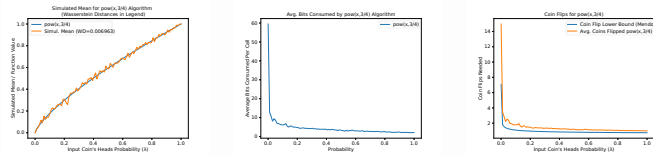
$\ln(1+x)$  (Flajolet)



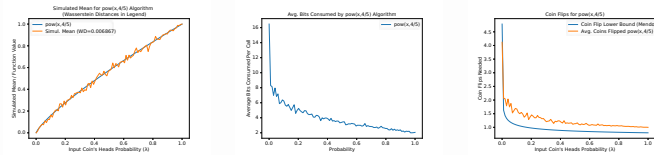
### ln(1+x) (Two-Coin Special Case)

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

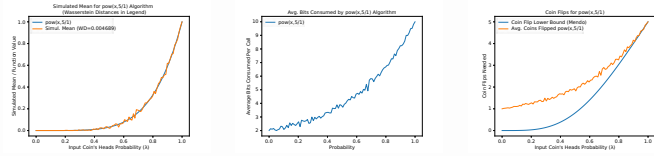
pow(x,2/1)

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

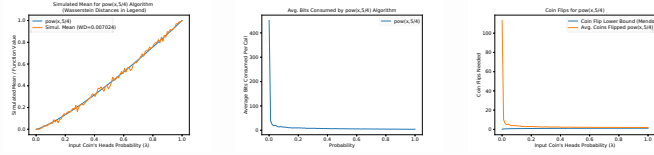
$\text{pow}(x, 4/5)$



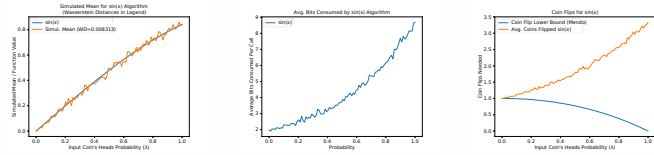
$\text{pow}(x, 5/1)$



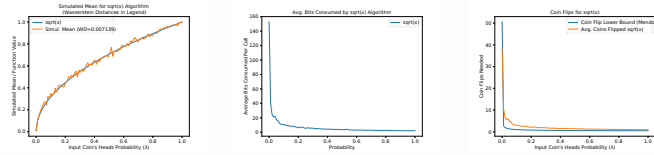
$\text{pow}(x, 5/4)$



$\sin(x)$



$\text{sqrt}(x)$



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