My Simulator - Step Explanation

October 28, 2021

Introduction

To get the plot data, we need to calculate the probability based on the randomly generating number. So we need to assign these numbers to corresponding interval. At first, I sort the array and iterate over the array to assign these numbers. I tried another method that does not require sorting. The performance does not improve a lot. Through time analysis, I found the main cost was from the iteration. So reducing the number of cycles is critical to performance. Based on the character of sorting, I tried to increase index more than 1 in each cycle. Fortunately, it worked. I tried to analyze how many steps we need to increase every cycle through probability.

Probability

Uniform Distribution

Suppose we have 10,000 data and 100 bins. For uniform distribution, in ideal state, each bin has 100 data. However, there exists bias. Suppose the step is x. The bias can be 1,2,...x-1. Each has $\frac{1}{x}$ probability. The total cycles for one bin can be calculated by

$$\frac{100}{x} + \frac{x*(x-1)}{2*x} = \frac{100}{x} + \frac{x-1}{2} \tag{1}$$

To minimize the formula, we should choose $x = \sqrt{200} = 14$. I used program to verify the idea. The result is same.

```python
for step in range(1,40):
    total_time = 0
    total_cycle = 0
    for i in range(0,100):
        mu, sigma = 3,7
        x,y,t,cycle = Uniform(mu, sigma,step)
        total_time += t
        total_cycle += cycle
    print(step,total_time/100,total_cycle/100)
```

```
1 0.011866888999938964 10000.0
2 0.006400179862976074 5075.23
3 0.0046367192268371585 3466.82
4 0.0038373327255249025 2687.62
5 0.0034091639518737792 2240.8
6 0.003099534511566162 1958.0
7 0.0028620505332946776 1770.4
8 0.0027508282661437987 1644.8
9 0.0026365232467651365 1556.48
10 0.0025994491577148436 1497.61
11 0.0025410032272338867 1448.4
12 0.0025718331336975096 1426.71
13 0.0025388026237487793 1419.64
14 0.0025403165817260744 1413.24
15 0.0025771522521972658 1418.56
16 0.002607076168060303 1415.95
17 0.0026194000244140624 1441.28
18 0.002629528045654297 1451.21
19 0.002665207386016846 1469.08
20 0.0027320218086242676 1505.67
21 0.002742805480957031 1535.6
22 0.0027956008911132813 1550.02
23 0.0028055286407470703 1568.06
24 0.0028483033180236816 1606.15
25 0.0029718780517578123 1665.04
26 0.003029818534851074 1737.25
27 0.0030639100074768065 1757.48
28 0.0030347156524658204 1756.63
29 0.0030283665657043456 1752.04
30 0.0030641245841979982 1730.94
31 0.0030532956123352053 1756.6
32 0.0031549382209777833 1834.6
33 0.003326010704040527 1946.24
```

Figure 1: Uniform Distribution Step

Normal Distribution

For normal distribution, I calculate the number of data in each bin in ideal state and change the step based on it. Though the total cycle decrease, the extra calculation costs more time. I decided to estimate one value for step. Suppose the total interval length is 8σ. So every 2σ contains 25 bins. Based on the character of Normal Distribution and the assumption that the bias has uniform distribution, we can get our round estimation.

$$\frac{6820}{25*x}+\frac{x-1}{2}+\frac{2720}{25*x}+\frac{x-1}{2}+\frac{420}{25*x}=\frac{9960}{25*x}+\frac{3*x}{2}-\frac{3}{2} \tag{2}$$

The best step is about 16. And the program shows similar result.

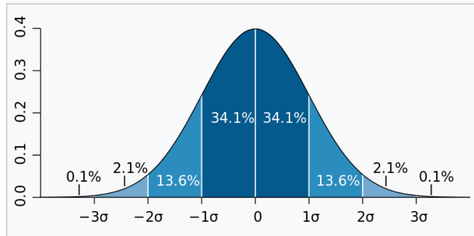

Figure 2: Normal Distribution. Source: Wikipedia

```python
for step in range(1,40):
    total_time = 0
    total_cycle = 0
    for i in range(0,100):
        mu, sigma = 3,7
        x,y,t,cycle = normB(mu, sigma,step)
        total_time += t
        total_cycle += cycle
    print(step,total_time/100,total_cycle/100)
```

```
1 0.011763937473297119 10000.0
2 0.006432297229766845 5067.34
3 0.004783821105957031 3451.78
4 0.004012978076934815 2664.04
5 0.0034560465812683103 2209.56
6 0.003127739429473877 1914.9
7 0.003084707260131836 1721.32
8 0.0029529190063476564 1582.29
9 0.0029254841804504396 1481.84
10 0.002750706672668457 1402.03
11 0.002683298587799072 1355.7
12 0.0025928497314453127 1321.33
13 0.0025288748741149904 1285.72
14 0.002563655376434326 1271.02
15 0.002576150894165039 1269.32
16 0.0025766658782958984 1254.1
17 0.002517256736755371 1257.92
18 0.002516908645629883 1251.12
19 0.00254561185836792 1272.7
20 0.0026000094413757323 1277.67
21 0.0025832962989807127 1297.6
22 0.002551362514495849 1287.1
23 0.002620418071746826 1304.5
24 0.002950873374938965 1324.17
25 0.0026840567588806152 1339.84
26 0.0027476143836975097 1375.5
27 0.0027990031242370607 1384.12
28 0.002798354625701904 1393.48
29 0.002850031852722168 1416.04
30 0.002904794216156006 1440.36
```

Figure 3: Normal Distribution Step

Gamma Distribution

It is hard to analyze the Gamma Distribution. But based on python program, for most cases (except shape is close to 0), 17 is a good choice.

```
for step in range(1,40):
    total_time = 0
    total_cycle = 0
    for i in range(0,100):
        shape,scale = 3,7
        x,y,t,cycle = Gamma(shape,scale,step)
        total_time += t
        total_cycle += cycle
    print(step,total_time/100,total_cycle/100)
```

```
1 0.012365310192108155 10000.0
2 0.006805665493011474 5060.8
3 0.004944388866424561 3439.54
4 0.0041567254066467284 2645.44
5 0.0036684346199035646 2186.08
6 0.003330769538879394 6 1888.25
7 0.003173038959503174 1687.36
8 0.003015613555908203 1544.35
9 0.0028899645805358885 1444.4
10 0.002818622589111328 1361.08
11 0.002792818546295166 1303.9
12 0.0027403974533081053 1256.21
13 0.002704651355743408 1235.32
14 0.002694427967071533 1209.14
15 0.002689361572265625 1191.06
16 0.002694599628448486 1176.85
17 0.0026636862754821776 1166.88
18 0.002705013751983 6425 1174.11
19 0.002679414749145508 1176.04
20 0.0027202415466308595 1183.43
21 0.0026968717575073242 1186.4
22 0.0027088451385498045 1193.86
23 0.0028308129310607912 1198.02
24 0.002881004810333252 1211.24
25 0.0028670930862426756 1212.16
26 0.0028252005577087402 1239.25
27 0.002878935337066 65 1265.82
28 0.002876634597778 32 1268.74
29 0.002967555522918701 1288.08
30 0.002857973575592041 1286.66
```

Figure 4: Normal Distribution Step

Finally, I decrease the Simulation time to less than 3ms.

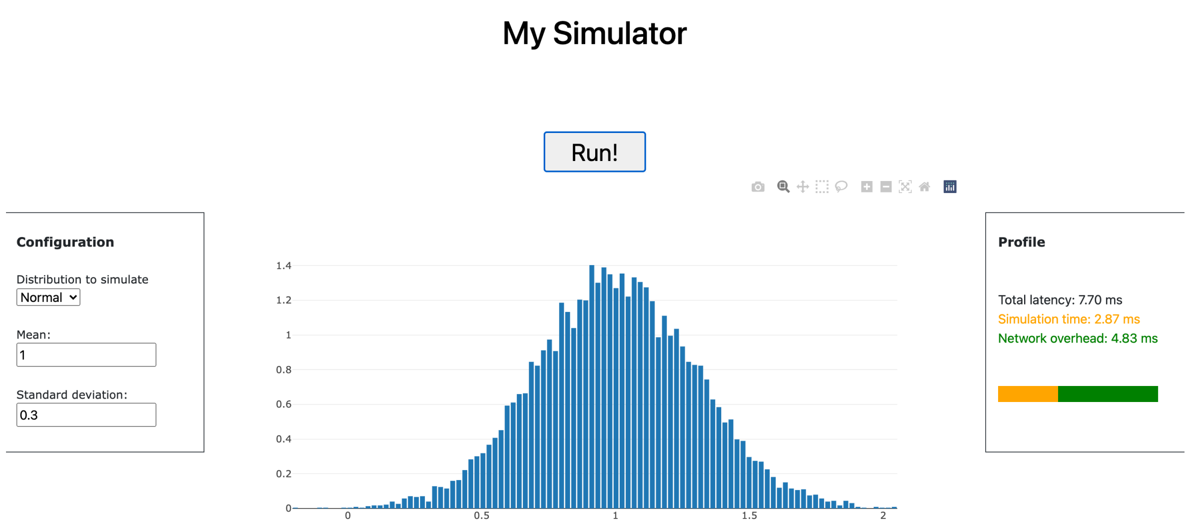

Figure 5: Normal Distribution Simulator

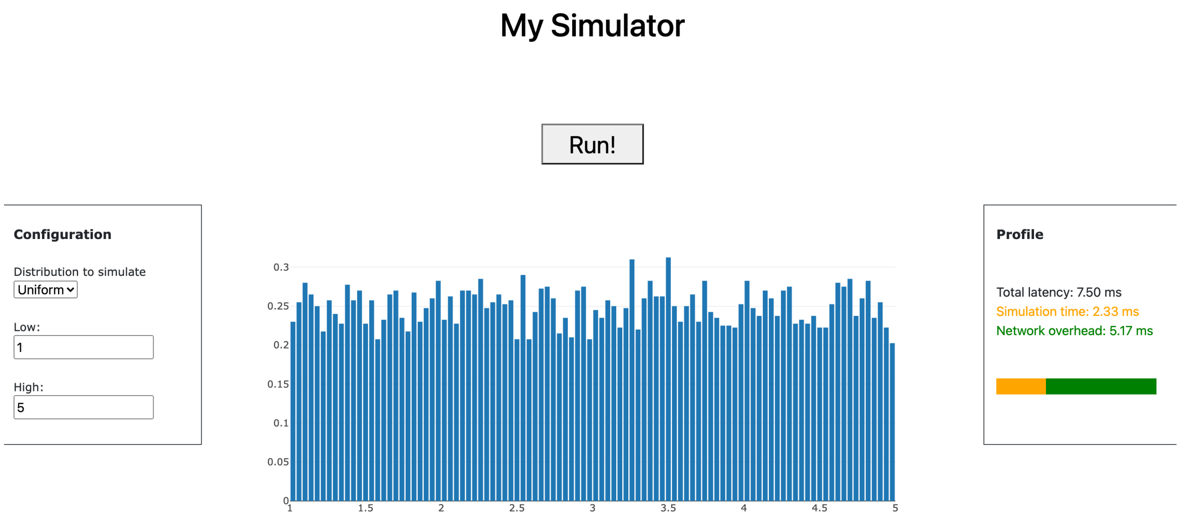

Figure 6: Uniform Distribution Simulator

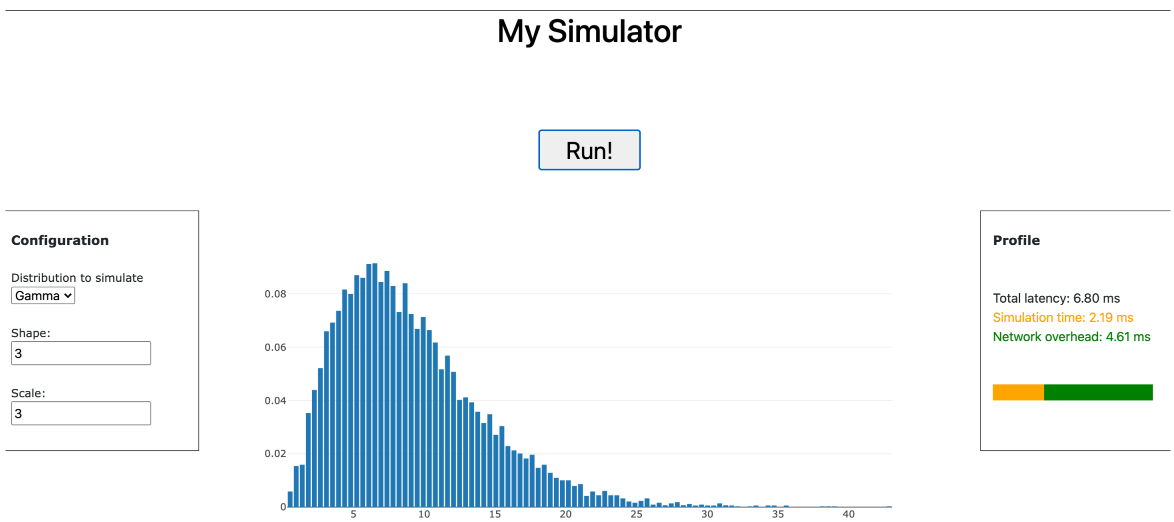

Figure 7: Gamma Distribution Simulator

All the data is got from test on Macbook Pro 2016, 2.7 GHz Quad-Core Intel Core i7.