Project

Name: Jie Wang
ZID: z5119770

Part 1.Test program

1.Indruduce process:

1.1 get the data we need

a)trace mode:

in the wrapper.py to read txt files and store the data into arrival_time and service_time ,we also get the server number ,setup time and delayoff time.

Then we send these parameters into simulation_method function

```
if mode == "trace":
    final_job = simulation.simulation_method(mode,arrival_time,service_time,ServerNum,SetupTime,DelayoffTime,time_end =0)
```

b)random mode:

this mode is to generate service_time and arrival_time randomly ,so in the txt files ,we only get the λ and μ and other parameters.

Then we send parameters into simulation_method function:

```
if mode =="random":
    TimeEnd = float(ore_data[3])
    final_job = simulation.simulation_method(mode, arrival_time, service_time, ServerNum, SetupTime, DelayoffTime,TimeEnd)
```

We use the λ and μ to exponentially generate arrival_time and service_time.

```
def random_number(number):
    aa = random.uniform(0,1)
    return - math.log(1-aa)/number
    # return random.expovariate(number)
if mode == "random":
    current_time = random_number(arrival[0])
    while current_time <= float(time_end):
        arrival_time.append(round(current_time,3))
        ser_time = round((random_number(service[0]) + random_number(service[0]) +random_number(service[0])),3)
        service_time.append(ser_time)
        tem_t = random_number(arrival[0])
        current_time = current_time + tem_t
```

1.2 initial data

After we get the arrival_list and service_list ,we need to initiail the state list and the job_list and choose when the job needs to be put into waiting list:

```
    for j in range(len(service_time)):
        job_list.append(str(j + 1) + " " + str(arrival_time[j]) + " " + str(service_time[j]) + " " + "num" + " " + "unmarked")
while len(finished_job) != len(service_time) or len(total_record[0]) != ServerNum :
    if len(job_list) >0 and round(float(job_list[0].split(" ")[1]),3) == round(time,3):
        waiting_list.append(job_list[0])
        job_list = job_list[1:len(job_list)]
```

At first, we need to set every job as “unmarked” and decide to put this job into waitinglist if the time is equal to this job’s arrival time.

1.3 process

There are four states in this project: off , setup, busy, off

First, when there is a arrival job, we need to check the “delayoff” state whether is empty, if it has the server ,we need put the current job into delayoff’ server to deal with, else we put the job into “off” state to setup and mark this job.

When the min time in the setup list is the current time, then we deal with the marked Job and put this server to the “busy” state.

In the “busy” list , if the min time in the busy list is equal to the current time , we need to look at the waiting list to see whether there is a job to deal with ,if have ,we just update the busy state ,after that we need to decide whether the current job is “unmarked”, if it’s marked ,we need to put the marked server off and if it’s “unmarked”, we just deal with this job inmidietly, else we put this server into “delayoff” state.

In the “delayoff” list ,if the time in the delayoff list add the delayofftime is equal to current time, put this server into “off” state, else we look the waiting list ,if it’s not empty ,we need to use current server to deal with job and put this server into “busy” state.

2.Several test results:

I tested several complex situations such as the same departure time :

```
arrival_time =[10,18,20,23,28,32,33,34,35,57,86,92]
service_time =[2,4,14,5,6,21,2,16,9,4,15,9]
final_job = simulation_method("trace", arrival_time, service_time, 3, 50, 100,1)
print(final_job)
total_time = 0.0
total_num  =len(final_job)
with open('test.txt', 'w') as f:
    for j in range(len(final_job)):
        aa = float(final_job[j].split(" ")[1])
        bb = float(final_job[j].split(" ")[2])
        print(aa)
        print(bb)
        f.write('%.3f' % aa + "\t" + '%.03f' % bb + "\n")
        total_time = total_time + (float(final_job[j].split(" ")[2]) - float(final_job[j].split(" ")[1]))
print("mean response time : " ,str('%.3f'%(round(total_time/total_num,3))))
```

Result:

```
10.000  62.000
18.000  66.000
23.000  73.000
28.000  76.000
33.000  78.000
20.000  80.000
35.000  89.000
57.000  93.000
32.000  94.000
34.000  94.000
92.000  103.000
86.000  108.000
```

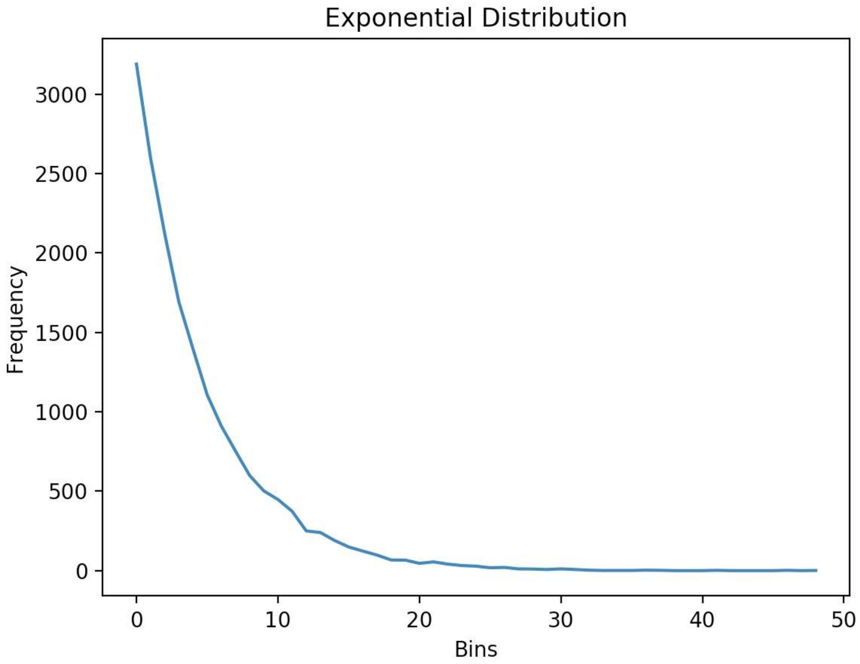
And this situation that the busy times are the same ,I deal with the left server and put the rest same time serve into “delayoff” state.

3. Exponential Distribution:

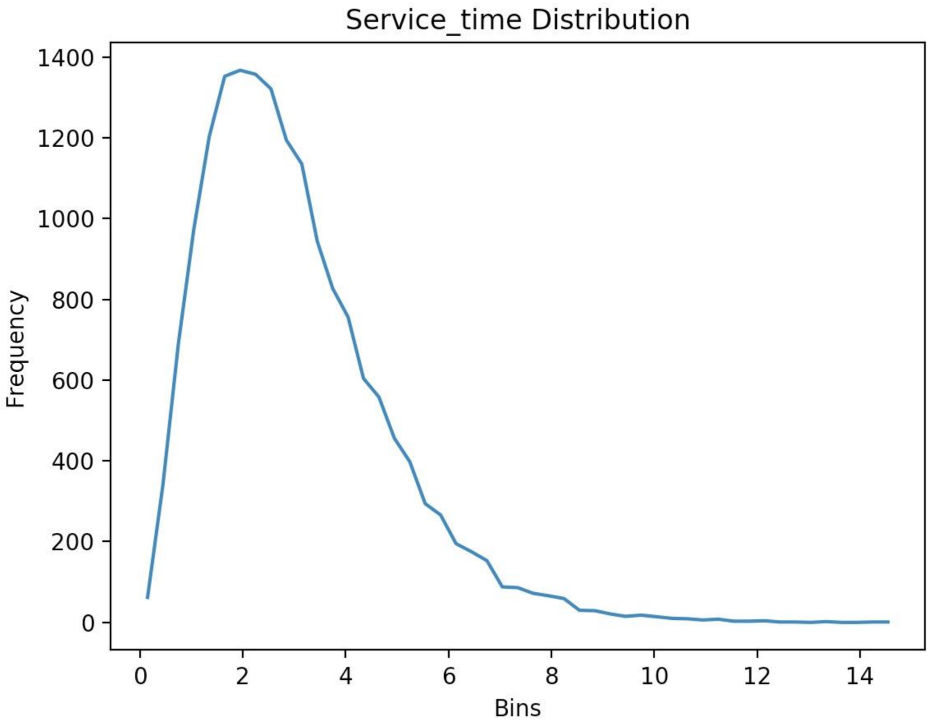
In this part ,we need to evident that the arrival time and service time generated in the random mode distribute exponentially.

I use the method in the lecture nodes to separate the data into 50 bins and calculate the frequency of the data in each bin and draw the image:

Arrival time:



Service time needs to exponentially generate three numbers and add them up then this is the new service time:



4. Result of section 3.2:

Example1:

given data:

In this example, there are $m = 3$ servers. The arrival and service times of the jobs are shown in Table 1. We assume all servers are in the OFF state at time zero. The setup time is assumed to be 50. The initial value of the countdown timer is $T_c = 100$. Table 2 shows the on-paper simulation with explanatory comments.

Arrival time	Service time
10	1
20	2
30	3
33	4

result:

```
10.000  61.000
20.000  63.000
30.000  66.000
33.000  70.000
```

mrt: 41.750

Example2:

given data:

In this example, there are $m = 3$ servers. In order to shorten the description, we will start from time 10 and the state of the system at this time is shown in Table 4. The arrival and service times of the job after time 10 are shown in Table 3. The setup time is assumed to be 5. The initial value of the countdown timer is $T_c = 10$. Table 4 shows the on-paper simulation with explanatory comments.

Arrival time	Service time
11	1
11.2	1.4
11.3	5
13	1

result:

```
11.000  12.000
11.200  12.600
13.000  14.000
11.300  17.000
```

mrt: 2.275

Part 2. Reproducible

When the mode is “random” ,I use the same seed and the same para.txt to run the simulation_mode ,and make the μ and λ different and change and the results generated need to be different:

The results:

Departure:

departure_3.txt

0.412	9.030
2.648	10.050
2.367	10.573
6.753	10.900
8.266	11.728
10.618	12.732
9.912	12.941
7.569	14.632
7.495	15.303
15.811	17.858
29.807	37.306
33.459	41.852
38.522	44.209
44.640	48.105
44.740	50.515
48.727	51.717
45.283	52.872
46.625	53.078
54.489	57.497
50.283	57.778
48.854	57.917
57.446	61.727
56.549	62.020
61.820	67.060
66.552	68.663
66.717	69.824
68.724	70.253
71.014	72.348
70.933	73.470
77.543	79.254
77.495	80.544
77.874	80.729
76.653	81.545
78.371	81.796
81.992	83.235
81.923	83.560
82.321	85.570
84.979	86.717
85.432	88.095
88.671	93.420
93.238	95.263
94.674	96.916
99.783	100.620
94.847	100.724
95.892	101.021
105.811	110.837
111.703	113.315
108.224	114.077
112.361	115.219
112.676	116.217
113.800	119.589
115.624	120.541
114.441	122.487
123.313	127.037
125.459	128.303
125.206	131.000
129.526	132.742
130.714	134.005
139.846	141.973
139.531	142.660
144.279	147.414
148.630	150.204
148.874	152.230
156.277	158.760
161.302	162.521

departure_4.txt

0.289	8.907
1.854	9.256
1.657	9.863
5.786	10.691
4.727	10.777
6.939	11.460
7.432	11.670
11.067	13.114
5.298	13.786
5.247	14.509
20.865	23.364
23.421	27.967
26.966	30.324
31.248	34.713
31.318	37.123
34.109	38.325
31.698	39.450
32.637	39.493
38.142	42.117
35.198	43.231
34.198	43.837
39.584	44.107
40.212	44.161
43.274	48.514
46.586	48.697
48.107	49.636
46.702	49.809
49.710	51.044
49.653	52.190
54.512	55.987
54.280	55.991
54.860	56.112
54.246	57.295
53.657	58.549
57.394	58.637
57.346	58.983
57.625	60.874
59.486	61.224
59.802	62.465
62.070	66.819
65.267	67.292
66.272	68.514
69.848	70.685
67.124	72.253
66.393	72.270
74.068	79.094
78.192	79.804
78.653	80.557
78.873	81.013
75.757	81.610
79.660	84.030
80.937	84.117
80.109	86.379
86.319	90.043
87.821	90.665
87.644	93.438
90.668	93.884
91.500	94.791
97.892	100.019
97.672	100.801
100.995	104.130
104.041	105.615
104.212	107.568
109.394	111.877
112.912	114.131

departure_5.tx

0.192	7.604
1.236	7.837
1.104	8.241
3.858	8.793
3.151	8.851
4.626	9.306
4.955	9.446
7.378	10.215
3.532	10.857
3.498	11.339
13.910	15.576
15.614	18.645
17.977	19.548
20.879	22.485
20.832	23.142
21.132	23.220
22.739	23.944
21.758	24.348
23.466	27.215
22.799	27.354
25.428	27.433
26.808	29.987
26.390	30.230
28.849	32.342
31.057	32.464
31.134	33.205
32.071	33.362
33.140	34.094
33.102	34.794
36.187	37.327
36.164	38.196
36.341	38.310
36.573	39.031
35.771	39.032
38.231	39.401
38.263	39.860
39.657	40.815
38.416	41.198
39.868	41.643
41.380	44.546
43.511	44.861
44.181	45.676
46.566	47.124
44.750	48.280
44.262	48.464
49.378	52.728
52.128	53.203
52.435	53.998
50.505	54.407
52.582	54.630
53.958	56.750
53.107	56.912
53.406	58.587
57.546	60.028
58.547	60.483
58.430	62.293
60.445	62.589
61.000	63.194
65.261	66.679
65.114	67.200
67.330	69.420
69.361	70.410
69.474	71.711
72.929	74.584
75.274	76.086

Mrt:

mrt_3.txt

4.097

mrt_4.txt

3.770

mrt_5.txt

2.327

Part 3. Suitable Value of Tc

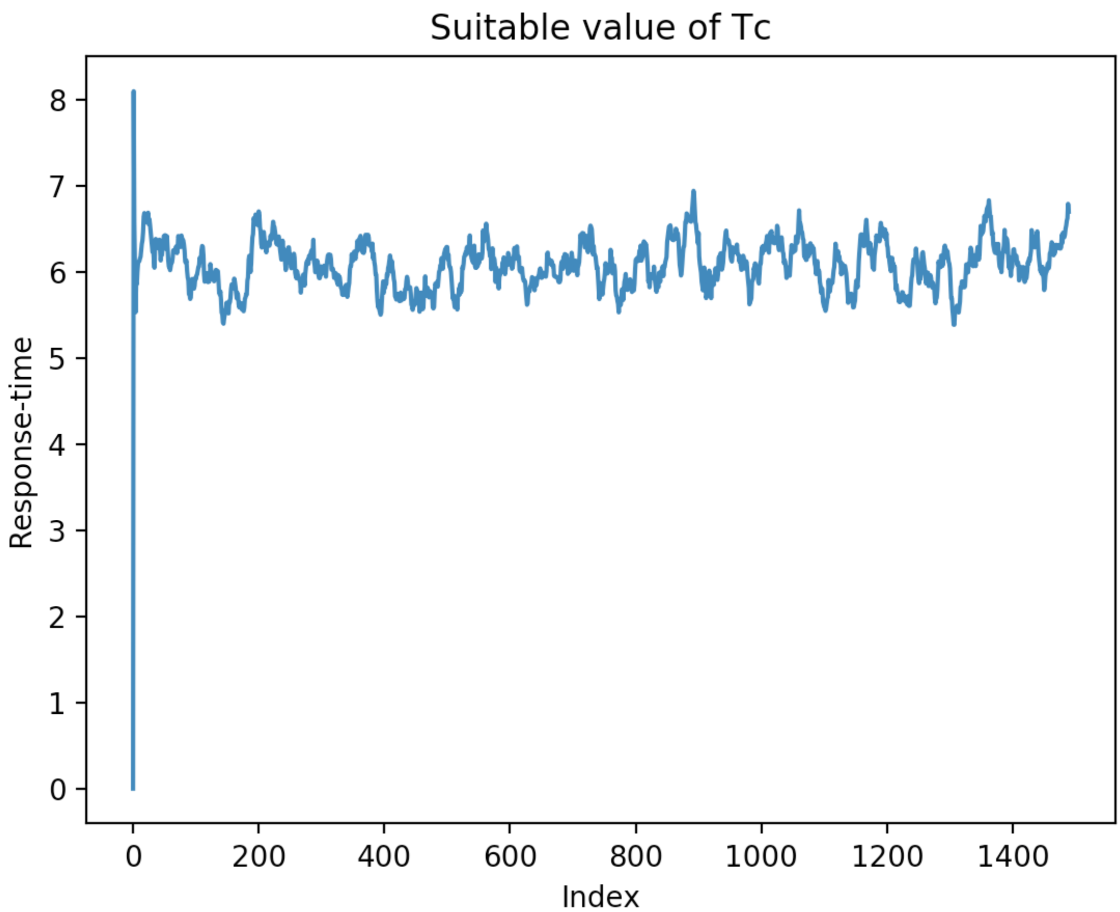
1. baseline

The parameters :

Server number =5
Setup time =5
Tc = 0.1
End time = 5000
λ = 0.35
μ = 1
number of replications = 15

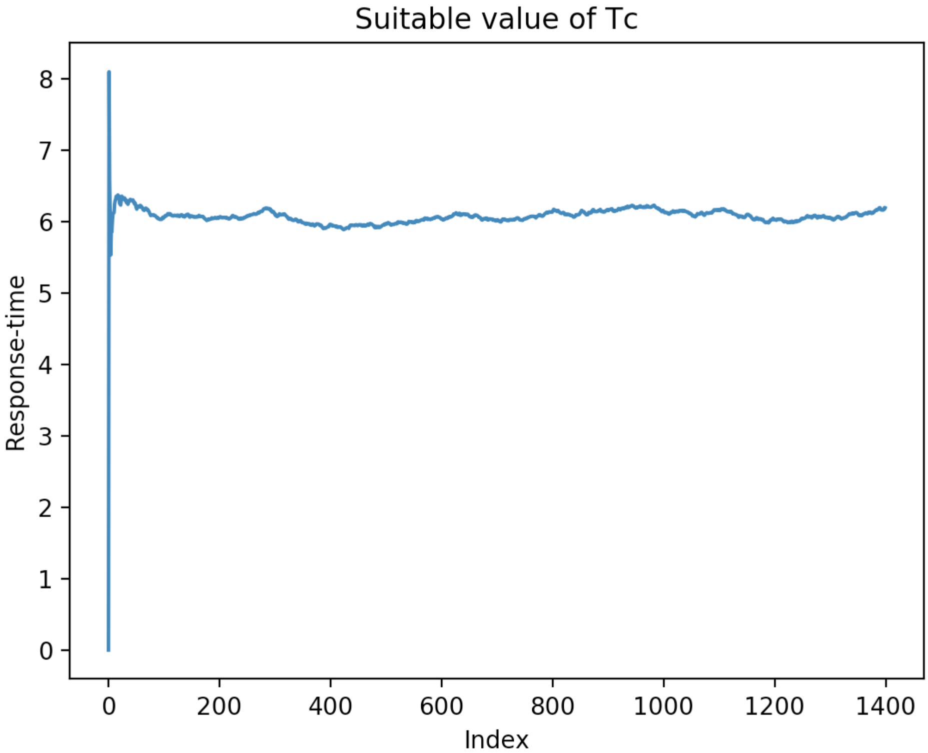
Run the program when the seed changes and get the several finish_job lists ,then we get first 1500 elements of each list ,and calculate the mean number of the same index of these five list and then use w to smooth the data and draw the image.

First we get the w =10:



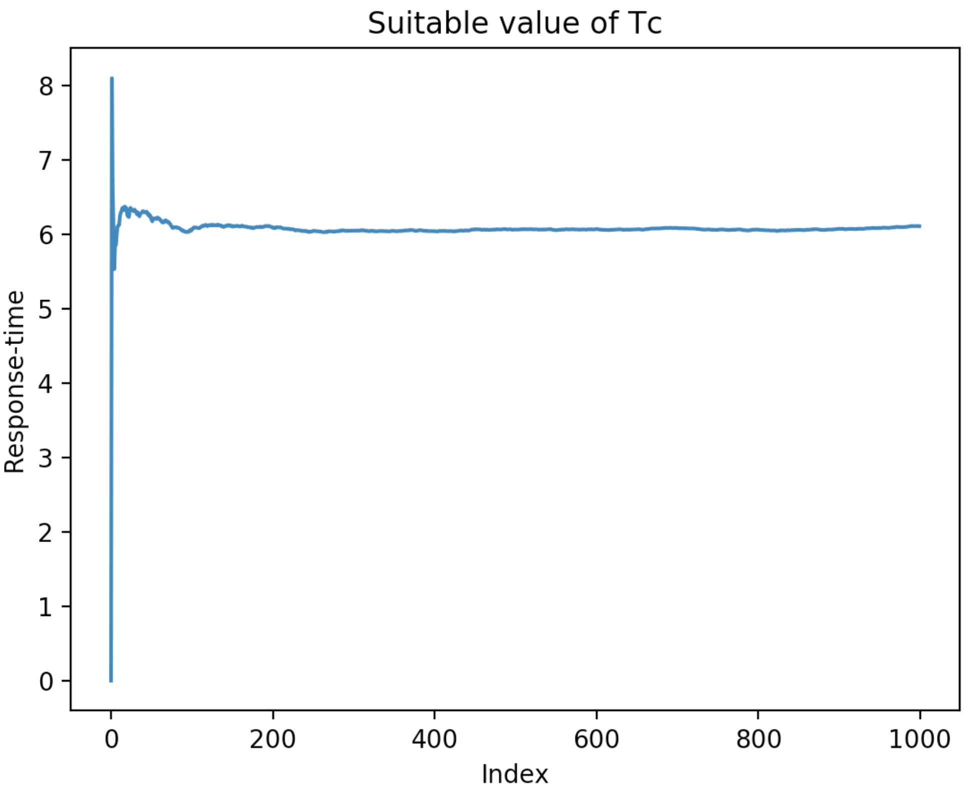
When w =10 ,we can see that the data is not stable ,so we need to add smooth which means to add w.

When w =100:



So we can see the result still has a little fluctuation but it’s near like a line.

When w = 500:



This result like a line ,and we remove the front data which very high or very low and then calculate the mean response time:

Mean response time : 6.06692028358

We use the method in the lecture notes to generate the confidence interval(confidence =0.95):

```
aa = np.std(tem_list,ddof =1)
bb = stats.t.ppf(1 - (1 - pro) / 2, 14)
temp = bb * aa / np.sqrt(15)
upper = mean + temp
down = mean - temp
```

- Compute the sample mean

$$\hat{T} = \frac{\sum_{i=1}^{n} T(i)}{n}$$

- And the sample standard deviation

$$\hat{S} = \sqrt{\frac{\sum_{i=1}^{n} (\hat{T} - T(i))^2}{n-1}}$$

Note: for sample standard deviation, *(n-1)* is in the denominator, *not n*.

There is a probability (1-α) that the mean response time that you want to estimate lies in the interval

$$[\hat{T} - t_{n-1,1-\frac{\alpha}{2}} \frac{\hat{S}}{\sqrt{n}}, \hat{T} + t_{n-1,1-\frac{\alpha}{2}} \frac{\hat{S}}{\sqrt{n}}]$$

The upper = 6.09683657154
The down = 6.03522138401
So as the project required ,the response time need to be less 2 units than current ,so we need to find a Tc which can get the mean response time around 4.06692028358 and the upper of the response time must less than 4.03522138401

2. Tc = 5:

Use the same sound method to get the mean response time:
Mean response time : 4.54742158771
Upper = 4.55845530296
So we need to add the Tc to get then smaller mrt because the upper > 4.03522138401

3. Tc = 10:

Mean response time: 3.75917924574
Upper = 3.82358655305
So we decide the Tc is between 5 and 10,and then we set the Tc =8

4. Tc = 8:

Mean response time : 4.01213616776
Upper = 4.09208386433
So we need to add the Tc to get then smaller mrt because the upper > 4.03522138401

5. Tc = 9:

Mean response time : 3.9189019555555564
Upper =3.95225216981
The uppe is less than 4.03522138401,so we can try Tc =8.5

6. Tc = 8.5:

Mean response time : 3.9710362222222244
Upper = 4.0274136287
So maybe the Tc is between the 8.5 -9

This is very close to the required response time , and the project require “at least 2 units” so the suitable Tc is better around 8.5 or larger than that .

7.Choose Parameter

(1)Transient removal

We can see that all the images have the transient part before time 3000s ,so we remove the first jobs that has the departure time larger than 3000 to get a closer and stable mean response time.

(2)End time

We choose the end time 5000,because this value can avoid transient part (When the end time is around 3000,the distribution is stable like a line) and it is enough large to get the enough data to get stable result.

(3) Number of replication

The number of replications is also to make sure that we get enough data to calculate the mean response time and make the result more close to the correct value.

We choose the number as 15, because it can help us to estimate a confidence interval of steady state mean response time and when we tried this number loop ,the result image is close to a smooth curve that means it approach to a stable situation.