Project Report

1.Introduction

The aim of this project is to apply discrete event simulation to a performance analysis problem, then use statistically sound methods to analyse simulation outputs and find out an optimal solution for an specific system at last.

2.Simulation programming

First, I decided to write the program by Python3. According to project description, there are two kinds of simulation, trace mode and random mode. I used while loop based on master clock to do this two simulations. For trace mode, all the parameters, arrival time and service time as well, are stored in files and it is not hard to implement it step by step according rules specified in project description.

2.1 Exponential distribution

For random mode, I used to **expovariate** function in **random** library to simulate the arrival time and service time. And there are some figures in the directory which I draw to prove the correctness of the inter-arrival probability distribution and service time distribution.

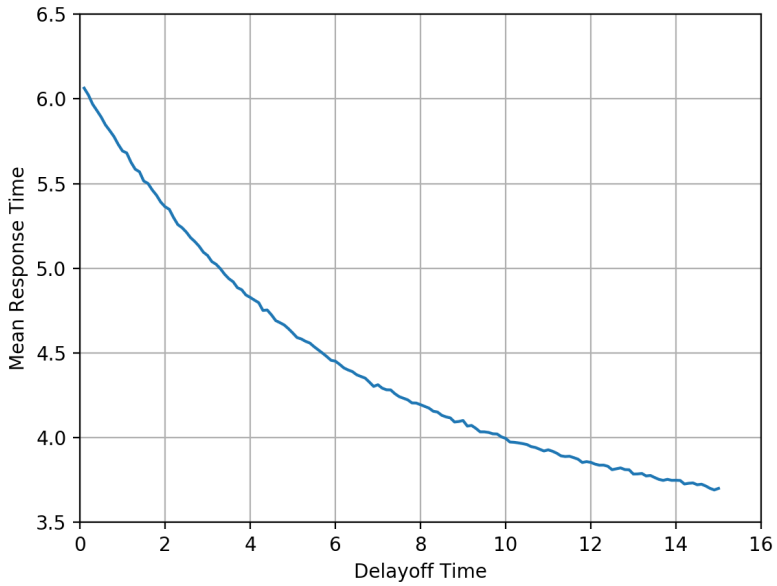
2.2 Reproducibility

Seed is also required when using **random** to make sure the results are replicable.I use Tc multiply i which is the index of simulation, which makes the seed is unique for every pair Tc and simulation. Here is how I decide the seed:

```
random.seed(int(delayoff_time * i * 10))
```

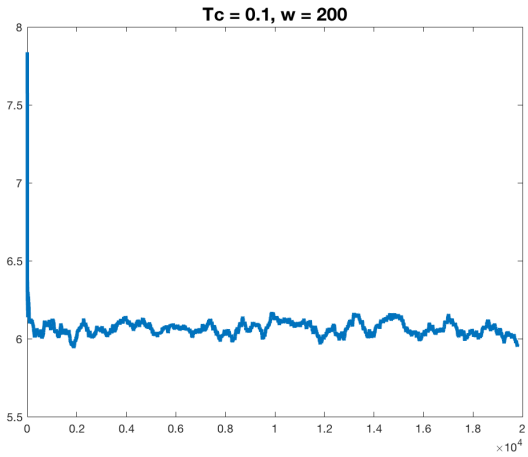
2.3 Analysing

After finishing coding, I began to solve the design problem by the simulation program. Given the following parameter values: number of servers is 5, setup time is 5, λ(arrival rate) is 0.35, μ(service rate) is 1. As Law and Kelton requires that there should be at least 5 replications, so I decided to do 10 replications to obtain more reasonable results. I want to make sure that stable part is larger than the transient part, so I set time_end value as 60000 which will give us at least 20000 results. Then I did the simulations by adding 0.1Tc every time and Tc is from 0.1 to 15.
There are 150 Tc (15/0.1 = 150), it is very difficult to do transient removals to all of them. So I could find a possible range, which might cover the final result I want, by comparing mean response time of different Tc before removing transient part. The mean response time of baseline is about 6.1 according to the figure below, and the improved system's response time must be 2 units less than that it, which is around 4.1. So I chose the delay off time between [9,10].

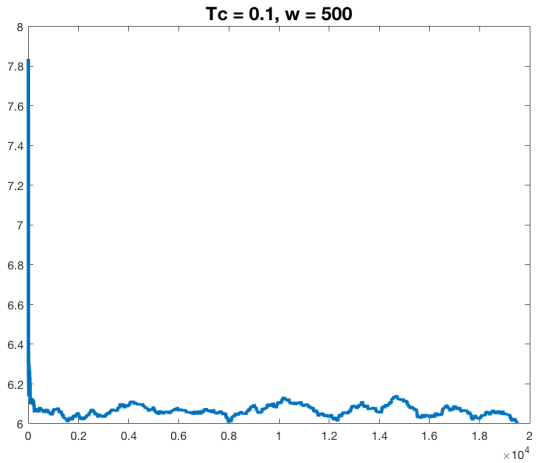


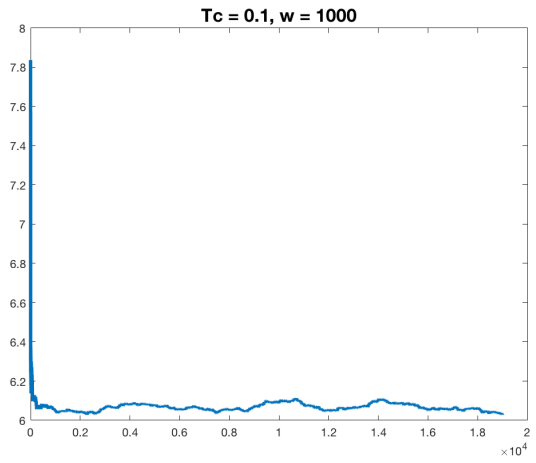
3.Figure out the transient end

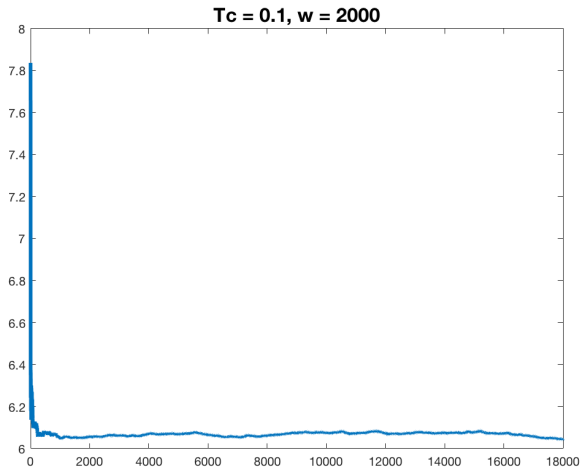
Please let me acknowledge the matlab file week07_q1_a.m based on Law and Kelton provided by Prof.Chou, which is modified by me to conveniently draw the graph.
First of all I choose w = 200, then get the figure below:



You see a lot of oscillation in the graph so I will need to increase the value of w to smooth it out. Then I increase w to 500, 1000, 2000 and the figures are shown below.







The graph is a smoother when w is 2000. So the transient end is 2000 and the mean response time is 6.0655 and the standard deviation is 0.0157 after cutting away the first 2000 results. Using the same method again, I did transient removals for all the Tc from 9.0 to 10.0. This procedure is done by trial-and-error and I designed a table below to show results.

Tc	w	MRT	STD
0.1	2000	6.0655	0.0157
9.0	1000	4.0812	0.0125
9.1	1000	4.0822	0.0255
9.2	1000	4.0680	0.0328
9.3	1000	4.0492	0.0173
9.4	2000	4.0508	0.0139
9.5	1000	4.0356	0.0261
9.6	1000	4.0309	0.0229
9.7	1000	4.0221	0.0141
9.8	1000	4.0190	0.0104
9.9	1000	4.0017	0.0213
10.0	2000	4.0007	0.0266

4.Compute confidence interval:

After transient removals, I am going to compute the confidence interval with the mean response time and the standard deviations for each system. The theoretical method has been provided in week07 lecture slides:

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

To calculate the confidence interval, I wrote a short Python program to help me based on the method above. Here is the code and the output:

```python
mrt = [4.0812, 4.0822 ,4.0680, 4.0492, 4.0508, 4.0356, 4.0309, 4.0221, 4.0190, 4.0017, 4.0007]
std = [0.0125, 0.0255 ,0.0328, 0.0173, 0.0139, 0.0261, 0.0229, 0.0141, 0.0104, 0.0213, 0.0266]



n = 10 # 10 replications
t = 2.262 # according the t-distribution table

def compute(T, t, S, n):
    left = T - (t * S) / (n)**0.5
    right = T + (t * S) / (n)**0.5
    return left, right

for i in range(len(mrt)):
    left, right = compute(mrt[i], t, std[i], n)
    print("The 95% confident interval is [" + str(left) + ", " + str(right) + "].")
```

```
The 95% confident interval is [4.072258659915874, 4.090141340084126].
The 95% confident interval is [4.063959666228383, 4.100440333771617].
The 95% confident interval is [4.044537923619253, 4.091462076380746].
The 95% confident interval is [4.036825185323569, 4.061574814676431].
The 95% confident interval is [4.040857229826451, 4.060742770173548].
The 95% confident interval is [4.016930481904344, 4.054269518095655].
The 95% confident interval is [4.014519464965881, 4.047280535034119].
The 95% confident interval is [4.012014168385106, 4.032185831614894].
The 95% confident interval is [4.011560805050007, 4.026439194949993].
The 95% confident interval is [3.9864639564966486, 4.0169360435033505].
The 95% confident interval is [3.98167282830098, 4.019727171699021].
```

From the output above, it is clear to observe the confidence interval for each system. I noticed that when the Tc is 9.3 and 9.4, the highest response time is also 2 units less than that of the baseline system. So I decided to compare these two systems with baseline system(-2 units)to get the final answer in the next step.

Compare two different systems:

To calculate the confidence interval, I wrote another short Python program to help me based on the method above. Here is the code:

```python
transient_end = 1000
number_of_jobs = 20000
baseline_system = []
improved_system = []
# read files
for i in range(1, 11):
    with open('tc0.1/trace' + str(i)) as f:
        response_time = [float(j) for j in f.read().splitlines()]
        mean_response_time = sum(response_time[transient_end:]) / (number_of_jobs - transient_end) - 2
        baseline_system.append(mean_response_time)
    with open('tc9.3/trace' + str(i)) as f:
        response_time = [float(j) for j in f.read().splitlines()]
        mean_response_time = sum(response_time[transient_end:]) / (number_of_jobs - transient_end)
        improved_system.append(mean_response_time)

print("    Index        baseline_system-2   improved_system     baseline_system-improved_system")
diff = []
for i in range(10):
    diff.append(baseline_system[i] - improved_system[i])
    print("replication" + str(i+1) + "   " + str(baseline_system[i]) + " " + str(improved_system[i]) + " " +
        str(improved_system[i] - baseline_system[i]))

mean = np.mean(diff)
std = np.std(diff)
n = 10 # 10 replications
t = 2.262 # according the t-distribution table
left,right = compute(mean, t, std, n)
print("The 95% confident interval is [" + str(left) + ", " + str(right) + "].")
```

Here is the result by comparing baseline minus 2 units system with Tc = 9.3:

Index	baseline_system-2	improved_system	improved_system-baseline_system
replication1	4.0753330470940154	4.043767051115368	-0.03156599597864762
replication2	4.063831410144348	4.0670105507051755	0.0031791405608272427
replication3	4.0719704822805705	4.0208785173569765	-0.05109196492359391
replication4	4.095886030981711	4.05546608771888	-0.04041994326283049
replication5	4.046083320345238	4.053678253742928	0.007594933397689907
replication6	4.069690644812055	4.036608219164757	-0.03308242564729813
replication7	4.049962446678076	4.0569966599029765	0.0070342132249008316
replication8	4.045765697701987	4.024590290755405	-0.02117540694658171
replication9	4.044357601913141	4.061096867005351	0.0167392650922098
replication10	4.070734816962242	4.071988451313721	0.0012536343514790715

The 95% confident interval is [-0.03043240654177501, 0.0021254965154060007].

Here is the result by comparing baseline system(-2 units) with Tc = 9.4:

Index	baseline_system-2	improved_system	improved_system-baseline_system
replication1	4.0753330470940154	3.9918705811445223	-0.08346246594949314
replication2	4.063831410144348	3.987672966819419	-0.07615844332492916
replication3	4.0719704822805705	4.018565202090845	-0.053405280189725346
replication4	4.095886030981711	4.010125807498721	-0.08576022348298995
replication5	4.046083320345238	4.035643258912139	-0.010440061433098613
replication6	4.069690644812055	4.022616429009771	-0.04707421580228388
replication7	4.049962446678076	4.001894452141133	-0.048067994536943104
replication8	4.045765697701987	3.9843367834777577	-0.06142891422422947
replication9	4.044357601913141	4.001890076206288	-0.042467525706853415
replication10	4.070734816962242	3.962634647922984	-0.10810016903925757

The 95% confident interval is [-0.08039964093490036, -0.042873417803060365].

The theoretical method of comparing two systems has also been provided in week07 lecture slides:
Let us denote the computed confidence interval by [p, q]
Case 1: p, q > 0: baseline system(-2 units) is better than improved system with probability (1-a)
Case 2: p, q < 0: improved system is better than baseline system(-2 units) with probability (1-a)
Case 3: p < 0 & q > 0: baseline system(-2 units) and improved system are not different with probability (1-a)
According the method I have mentioned above, I cannot determine which system is better under this situation between baseline system(-2 units) with Tc = 9.3, however, I can decide that when Tc = 9.4, the improved system is better than baseline system(-2 units). In conclusion, the delay off time of the improved system is 9.4.