COMP9334 - Project

Tianwei Zhu, z5140081

This project program is written in python 3.6 and excel is used to generate data graph.

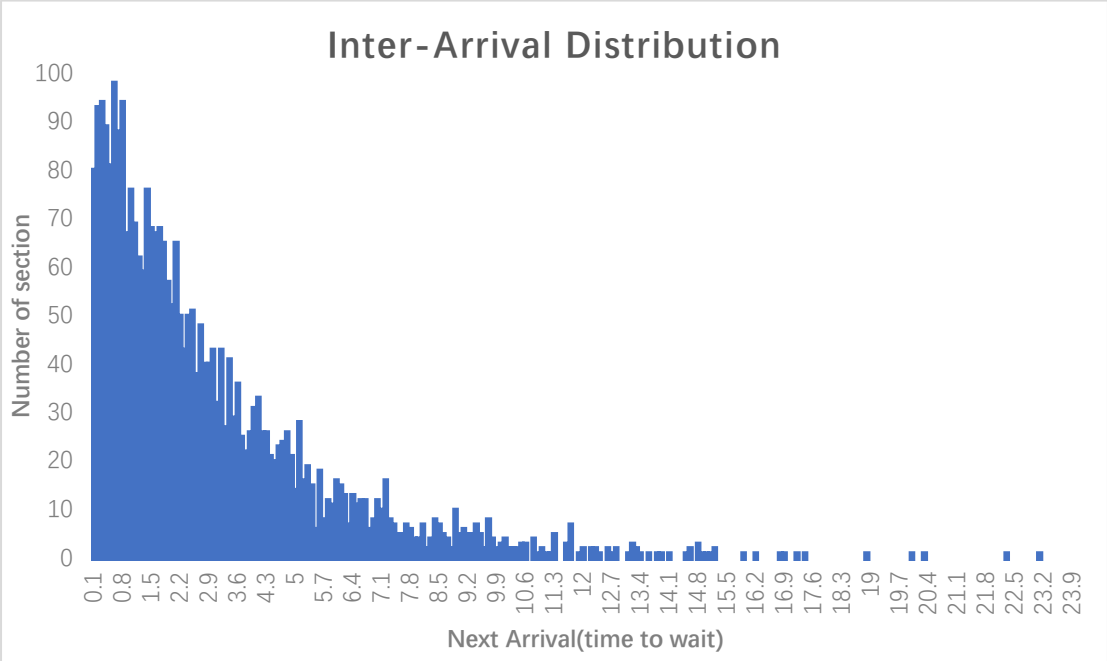
Part 1

For this simulation program, I used simulated system time clock to complete the main function. After testing several cases (combination of different input parameters), I found that 0.01 clock precision is good enough for the system. A more accurate time clock (such as 0.001) will make the complexity higher but does little help to get some better results.

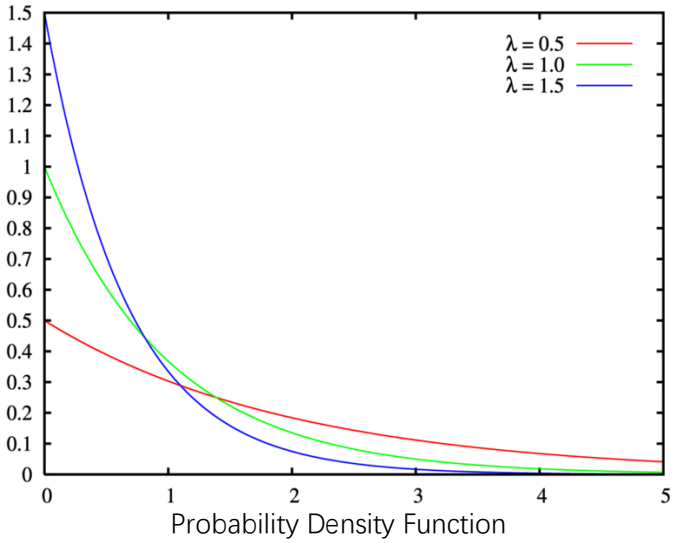
To generate exponential distribution inter-arrival jobs, a simple way to do so is to use python build in library *random.expovariate()*.

```
A = [random.expovariate(λ) for x in range(number_of_jobs)]
```

Let' s set $\lambda = 0.35$ and number of jobs equal 10,000. We can get a primary list which contains fluctuated times to wait between two jobs. Here is the graph for distribution of inter-arrival:



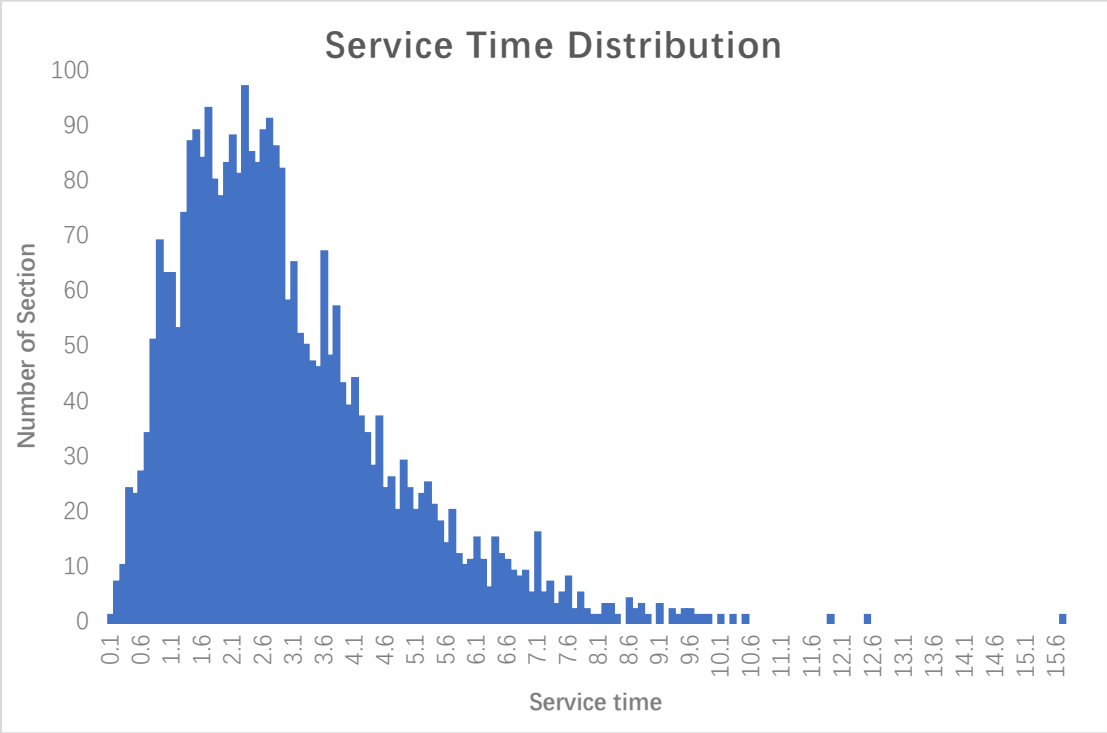
If we compare it with the Probability Density Function, we can find that the inter-arrival follows exponential distribution.



To generate service time in exponential distribution, I used *random.expovariate() to make three independent values and added them up*. The program run this function triple times than jobs number at first, then sum up each three of them.

```
B = [random.expovariate(u) for y in range(3*number_of_job)]
```

Let' s set u=1 and number of jobs to 10,000. Here is the graph for the distribution of service time:



It is obvious that the graph follows Phase-type distribution.

Here is the sample output for example 1 in section 3.2.1. It records every event during the running period and output start-departure time pairs as well as mean response time in the end. You can find that it is same as the <Table 2: Table illustrating the updates for Example 1>.

```
/Users/freshmilk/venv/bin/python /Users/freshmilk/Desktop/COMP9334/ass2/wrapper.py
Time=0, Dispatcher: []
Servers1, OFF, setup:0, busy:[], delay:0
Servers2, OFF, setup:0, busy:[], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=10.0, Dispatcher: [(10.0, 1.0, 'MARKED')]
Servers1, SETUP, setup:60.0, busy:[], delay:0
Servers2, OFF, setup:0, busy:[], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=20.0, Dispatcher: [(10.0, 1.0, 'MARKED'), (20.0, 2.0, 'MARKED')]
Servers1, SETUP, setup:60.0, busy:[], delay:0
Servers2, SETUP, setup:70.0, busy:[], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=32.0, Dispatcher: [(10.0, 1.0, 'MARKED'), (20.0, 2.0, 'MARKED'), (32.0, 3.0, 'MARKED')]
Servers1, SETUP, setup:60.0, busy:[], delay:0
Servers2, SETUP, setup:70.0, busy:[], delay:0
Servers3, SETUP, setup:82.0, busy:[], delay:0
Time=33.0, Dispatcher: [(10.0, 1.0, 'MARKED'), (20.0, 2.0, 'MARKED'), (32.0, 3.0, 'MARKED'), (33.0, 4.0, 'UNMARKED')]
Servers1, SETUP, setup:60.0, busy:[], delay:0
Servers2, SETUP, setup:70.0, busy:[], delay:0
Servers3, SETUP, setup:82.0, busy:[], delay:0
Time=60.0, Dispatcher: [(20.0, 2.0, 'MARKED'), (32.0, 3.0, 'MARKED'), (33.0, 4.0, 'UNMARKED')]
Servers1, BUSY, setup:0, busy:[10.0, 61.0], delay:0
Servers2, SETUP, setup:70.0, busy:[], delay:0
Servers3, SETUP, setup:82.0, busy:[], delay:0
Time=61.0, Dispatcher: [(32.0, 3.0, 'MARKED'), (33.0, 4.0, 'MARKED')]
Servers1, BUSY, setup:0, busy:[20.0, 63.0], delay:0
Servers2, SETUP, setup:70.0, busy:[], delay:0
Servers3, SETUP, setup:82.0, busy:[], delay:0
Time=63.0, Dispatcher: [(33.0, 4.0, 'MARKED')]
Servers1, BUSY, setup:0, busy:[32.0, 66.0], delay:0
Servers2, SETUP, setup:70.0, busy:[], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=66.0, Dispatcher: []
Servers1, BUSY, setup:0, busy:[33.0, 70.0], delay:0
Servers2, OFF, setup:0, busy:[], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=70.0, Dispatcher: []
Servers1, DELAY, setup:0, busy:[], delay:170.0
Servers2, OFF, setup:0, busy:[], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=170.0, Dispatcher: []
Servers1, OFF, setup:0, busy:[], delay:0
Servers2, OFF, setup:0, busy:[], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
[10.0, 61.0]
[20.0, 63.0]
[32.0, 66.0]
[33.0, 70.0]
mean response time:  41.25
```

Same idea, here is the output for example 2 in section 3.2.2.

```
/Users/freshmilk/venv/bin/python /Users/freshmilk/Desktop/COMP9334/ass2/wrapper.py
Time=10, Dispatcher: []
Servers1, DELAY, setup:0, busy:[], delay:20
Servers2, DELAY, setup:0, busy:[], delay:17
Servers3, OFF, setup:0, busy:[], delay:0
Time=11.0, Dispatcher: []
Servers1, BUSY, setup:0, busy:[11.0, 12.0], delay:0
Servers2, DELAY, setup:0, busy:[], delay:17
Servers3, OFF, setup:0, busy:[], delay:0
Time=11.2, Dispatcher: []
Servers1, BUSY, setup:0, busy:[11.0, 12.0], delay:0
Servers2, BUSY, setup:0, busy:[11.2, 12.6], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=11.3, Dispatcher: [(11.3, 5, 'MARKED')]
Servers1, BUSY, setup:0, busy:[11.0, 12.0], delay:0
Servers2, BUSY, setup:0, busy:[11.2, 12.6], delay:0
Servers3, SETUP, setup:16.3, busy:[], delay:0
Time=12.0, Dispatcher: []
Servers1, BUSY, setup:0, busy:[11.3, 17.0], delay:0
Servers2, BUSY, setup:0, busy:[11.2, 12.6], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=12.6, Dispatcher: []
Servers1, BUSY, setup:0, busy:[11.3, 17.0], delay:0
Servers2, DELAY, setup:0, busy:[], delay:22.6
Servers3, OFF, setup:0, busy:[], delay:0
Time=13.0, Dispatcher: []
Servers1, BUSY, setup:0, busy:[11.3, 17.0], delay:0
Servers2, BUSY, setup:0, busy:[13.0, 14.0], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=14.0, Dispatcher: []
Servers1, BUSY, setup:0, busy:[11.3, 17.0], delay:0
Servers2, DELAY, setup:0, busy:[], delay:24.0
Servers3, OFF, setup:0, busy:[], delay:0
Time=17.0, Dispatcher: []
Servers1, DELAY, setup:0, busy:[], delay:27.0
Servers2, DELAY, setup:0, busy:[], delay:24.0
Servers3, OFF, setup:0, busy:[], delay:0
Time=24.0, Dispatcher: []
Servers1, DELAY, setup:0, busy:[], delay:27.0
Servers2, OFF, setup:0, busy:[], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
Time=27.0, Dispatcher: []
Servers1, OFF, setup:0, busy:[], delay:0
Servers2, OFF, setup:0, busy:[], delay:0
Servers3, OFF, setup:0, busy:[], delay:0
[11.0, 12.0]
[11.2, 12.6]
[13.0, 14.0]
[11.3, 17.0]
mean response time:  2.275
```

Part 2

In this part I will show the reproducibility of program.

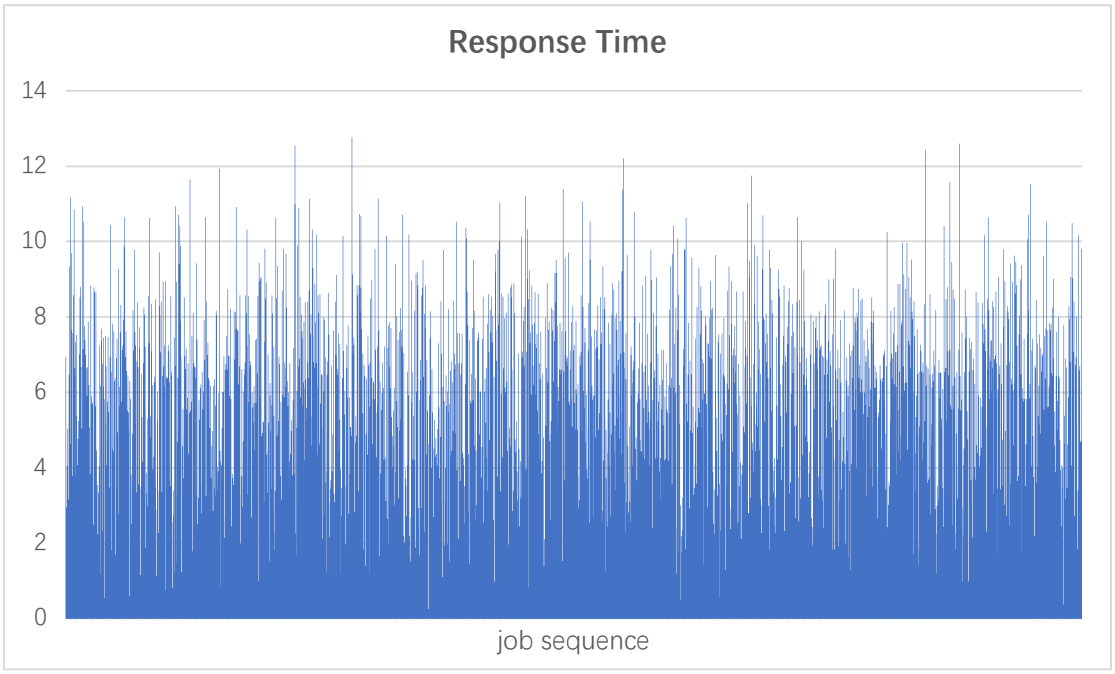
Normally, I set a seed (e.g., 0) for random mode in order to get same result for a fixed set of parameters. In week 6 we have learned that a seed will generate the same random sequence, and this will help us to realize reproducible program.

There are four sets of parameters and output files in my report. The output file set is called *departure_re_*.txt* and *mrt_re_*.txt*. Here are detailed parameters:

1. $\lambda = 0.3, u = 1.2, m = 5, setup = 5, delay = 20, endtime = 3000, seed = 0 \implies MRT = 3.172$
2. $\lambda = 0.5, u = 0.8, m = 5, setup = 7, delay = 10, endtime = 2000, seed = 1 \implies MRT = 5.265$
3. $\lambda = 0.7, u = 0.5, m = 10, setup = 20, delay = 30, endtime = 1500, seed = 2 \implies MRT = 7.174$
4. $\lambda = 0.4, u = 0.6, m = 7, setup = 10, delay = 20, endtime = 2000, seed = 3 \implies MRT = 6.618$

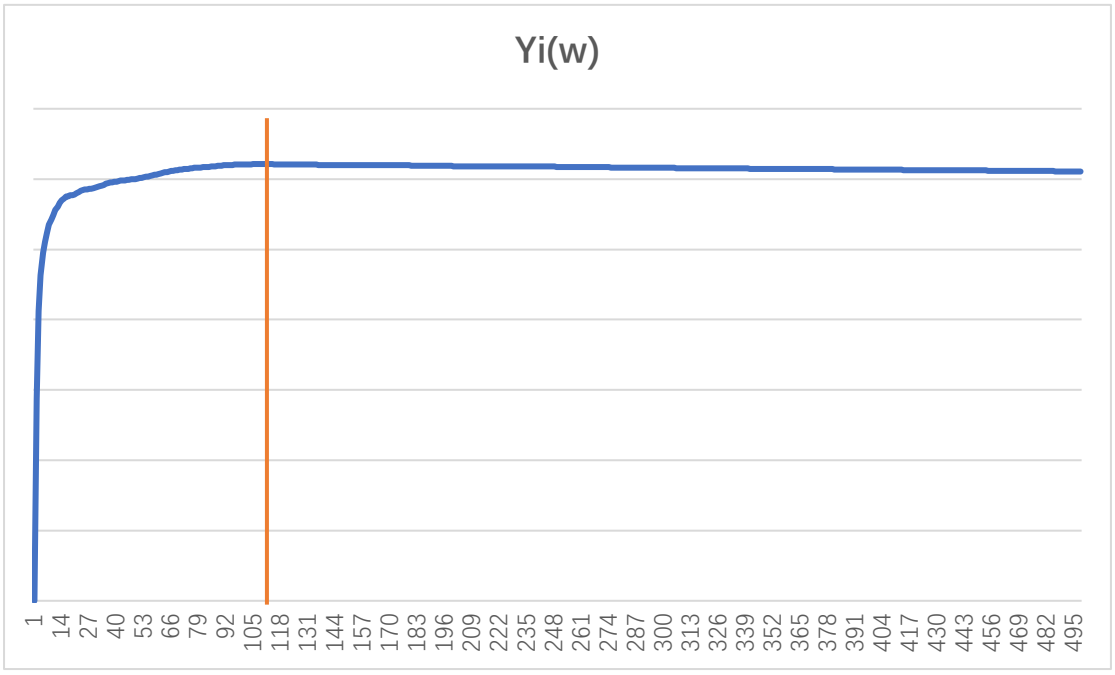
Part 3

Here is a distribution graph for response time of each jobs.



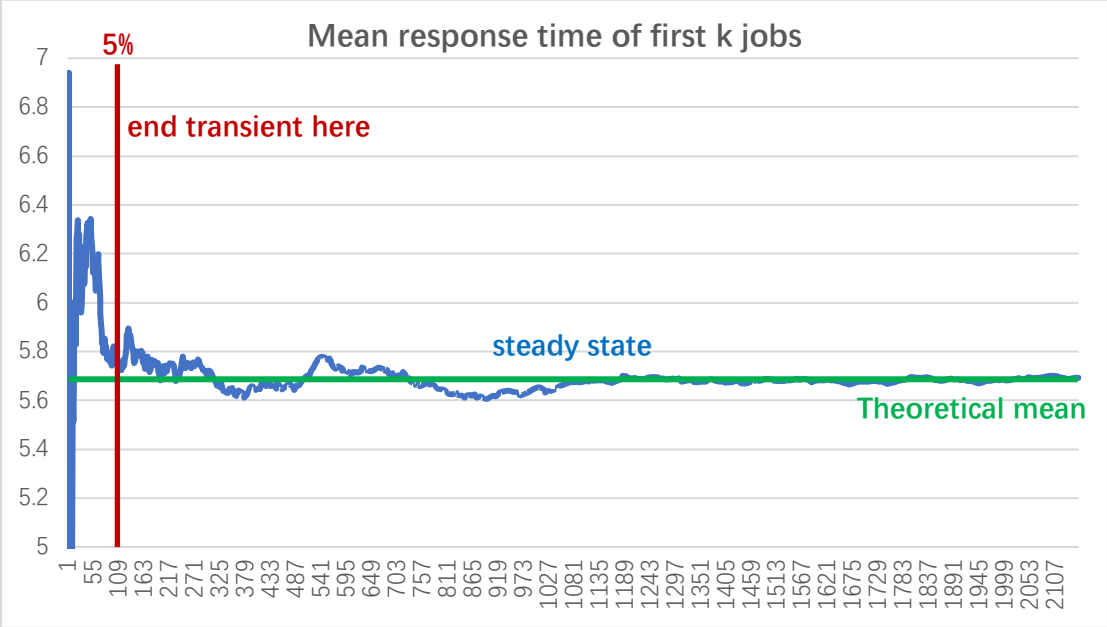
By applying this formula, we can get the Yi(w) graph as below:

$$\bar{Y}_i(w) = \begin{cases} \dfrac{\sum_{s=-w}^{w} \bar{Y}_{i+s}}{2w+1} & \text{if } i = w+1, \ldots, m-w \\ \dfrac{\sum_{s=-(i-1)}^{i-1} \bar{Y}_{i+s}}{2i-1} & \text{if } i = 1, \ldots, w \end{cases}$$



As there are 2107 jobs and we take first 110 as transient state, we can find that the MRT comes into steady state after 5% jobs were done. So, in my program, I dropped first 5% response time at the head of list.

Below is the mean response time graph for first k jobs.



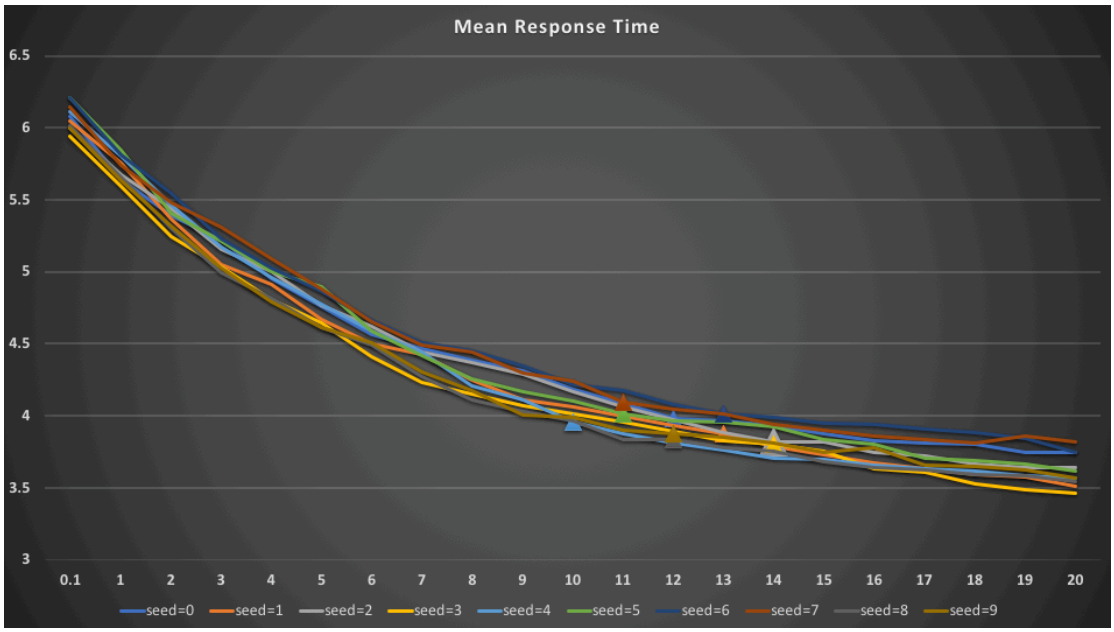
To find a suitable section for Tc, the input parameters are $\lambda = 0.35, u = 1, m = 5, setup = 5, endtime = 3000$

In order to end the simulation before end_time, I set the number of jobs as below:

$$\#jobs = end\_time * \lambda * 0.9$$

This formula can finish all jobs at about 90% end_time. For example, if end_time=3000s, then the program will finish at almost 2700s.

Apart from that, I tried ten different seeds to generate random values. Each seed will run Tc from 0.1 to 20, the final graph is shown below (triangle represents a Tc which makes MRT reduce by 2).



We can find that all suitable **Tc** which make mean response time reduce by 2 units are seat between **10 and 14**. Although a larger Tc value such as 50 can do better in reducing MRT, this will waste much more energy for the system.

I also tried different end_time values, they indeed do a little influence on mean response time but do almost nothing to the selection of a suitable Tc.

Finally, considering applicability of Tc section (different combination of those parameters), we can make this range to **[10, 20].**