COMP9334: Project Report - Discrete Event Simulation

Changfeng Li (z5137858)

UNSW
School of Computer Science and Engineering
Sydney, Australia

15/05/2018

Part 1 - Implentation of the source code

1.1 Theoretical and algorithm Introduction

At the beginning of the simulation, according to the topic requirements, we will implement the whole simulation algorithm by divide the program into two parts: *Random* mode and *Trace* mode.

In *Random* mode, we need to generate a job assembly lines based on *arrive_rate* and *service_rate*, and we need to specify a fixed completion time *time_end*. After *time_end*, those jobs have not been done by the server should be discarded. In the *trace* mode, we need to directly use the list of time for known events, we should be given list *arrive_time_list*, and the corresponding list of *service_time_list* as input for simulation. In both simulation modes we need to calculate the departure time recorded with their corresponding arrival time of all jobs and Mean Response Time (MRT) for follow-up computation and analysis.

1.1.1 Module of the body simulation algorithm

1. Simulation for Random and Trace mode.
2. File I/O Part.
3. Transient test.
4. Proper tc test.
5. Prove the correctness of modeling.
6. Other functional class or function provided.

Detailed introduction are below.

1.1.2 Principle of generating random number

Based on the principle of trace mode, We have so far assume that we can look up a list of arrival times and service times for the next customer which might be clear. However, sometimes we are required to solve a queue with a specific inter-arrival rate and service time probability distribution,
The method we apply for generating random number from a particular distribution is called "Inverse transform method", In this method, generating random numbers with cumulative density function (CDF) $F(x) = Prob[X \leq x]$, the following procedure should be executed:
1.Generate a number u which is uniformly distributed in (0,1)
2.Compute the number $F^{-1}(u)$, In expoential distribution, CDF is $1 - e^{-\lambda x}$.

Therefore, Given a sequence $\{U_1, U_2, U_3, ...\}$which is uniformly distributed in $(0, 1)$, The sequence $\frac{-log(1-u_k)}{\lambda}$is exponentially distributed with rate λ. Similarily, Based on the rules prescribed by this project, we use log sum of 3 random variables t_{k1}, t_{k2}, t_{k3} as the service time sequence. That is $\frac{-log(1-t_{k1})-log(1-t_{k2})-log(1-t_{k2})}{\mu}$.

At most case, It's enough for us to generate random number based on Linear Congruential Generator (LCG), LCG generates a sequence of integers $Z_1, Z_2, Z_3, ...$ according to the recursion $Z_k = aZ_{k-1} + c(\ MOD\ m)$ where a, c and m are integers, This result has a cycle of $m - 1$ we get a series of random number which is called $pseudo - random$ series. In this formula, Z_1 is the *seed*. This is the importance of using uniformly distributed random numbers for this simulation program.

1.2 Verifiy the correctness of the model

1.2.1 Prove the correctness for generating the distribution

According to the requirement of this project, we need to look for evidence that we have verified the correctness of the inter-arrival probability distribution and service time distribution correctly. Thus, we should do the test below.
Based on week2's slide, a continuous random variable is exponentially distributed with rate λ if it has probability density function:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

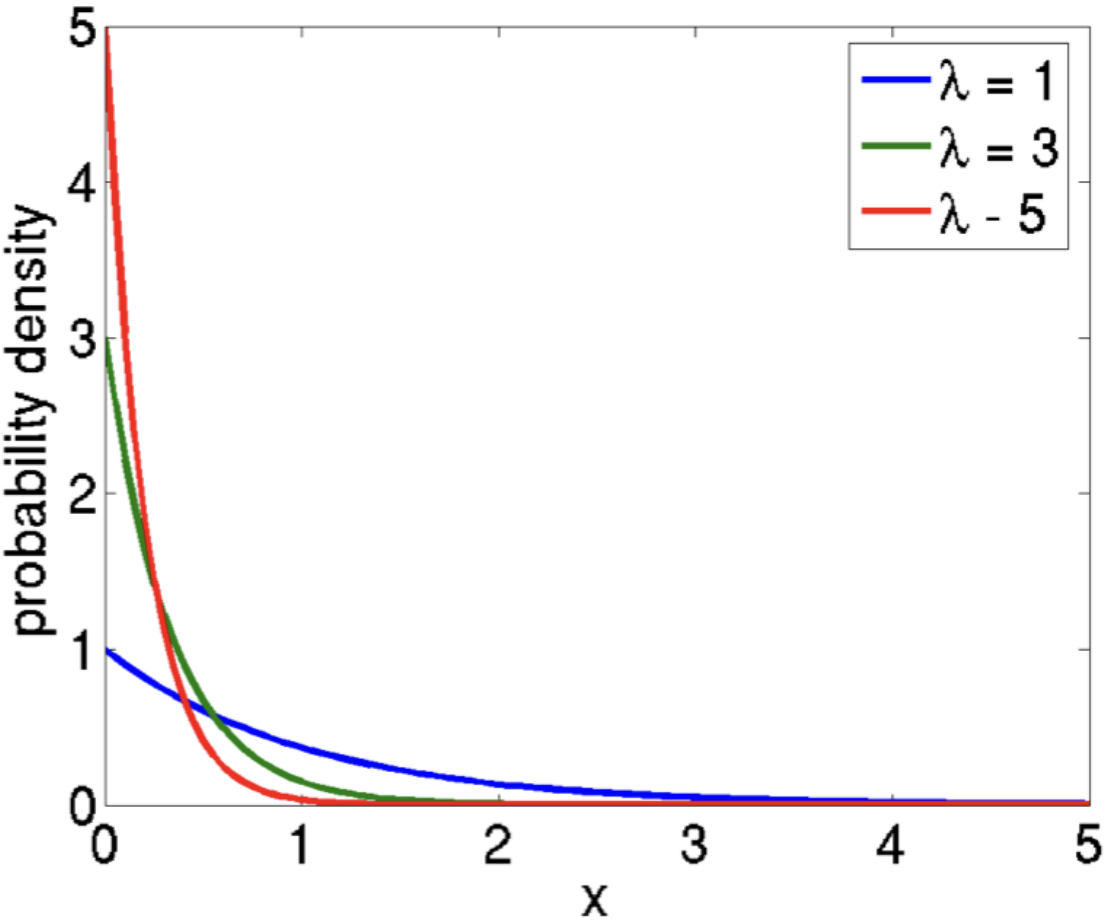

Figure 1: trace 03

In order to prove that the inter-arrival probability distribution and service time distribution have been generated correctly. These steps are processed to prove it. The detailed steps for the task list below.
1.Generate 10,000 uniformly distributed numbers in (0,1)
2. Compute $-log(1-u_k)/2$ where uk are the numbers generated in Step 1
3. The plot shows
3.1. The histogram of the numbers generated in Step 2 in 50 bins
3.2. The red line show the expected number of exponential distributed numbers in each bin

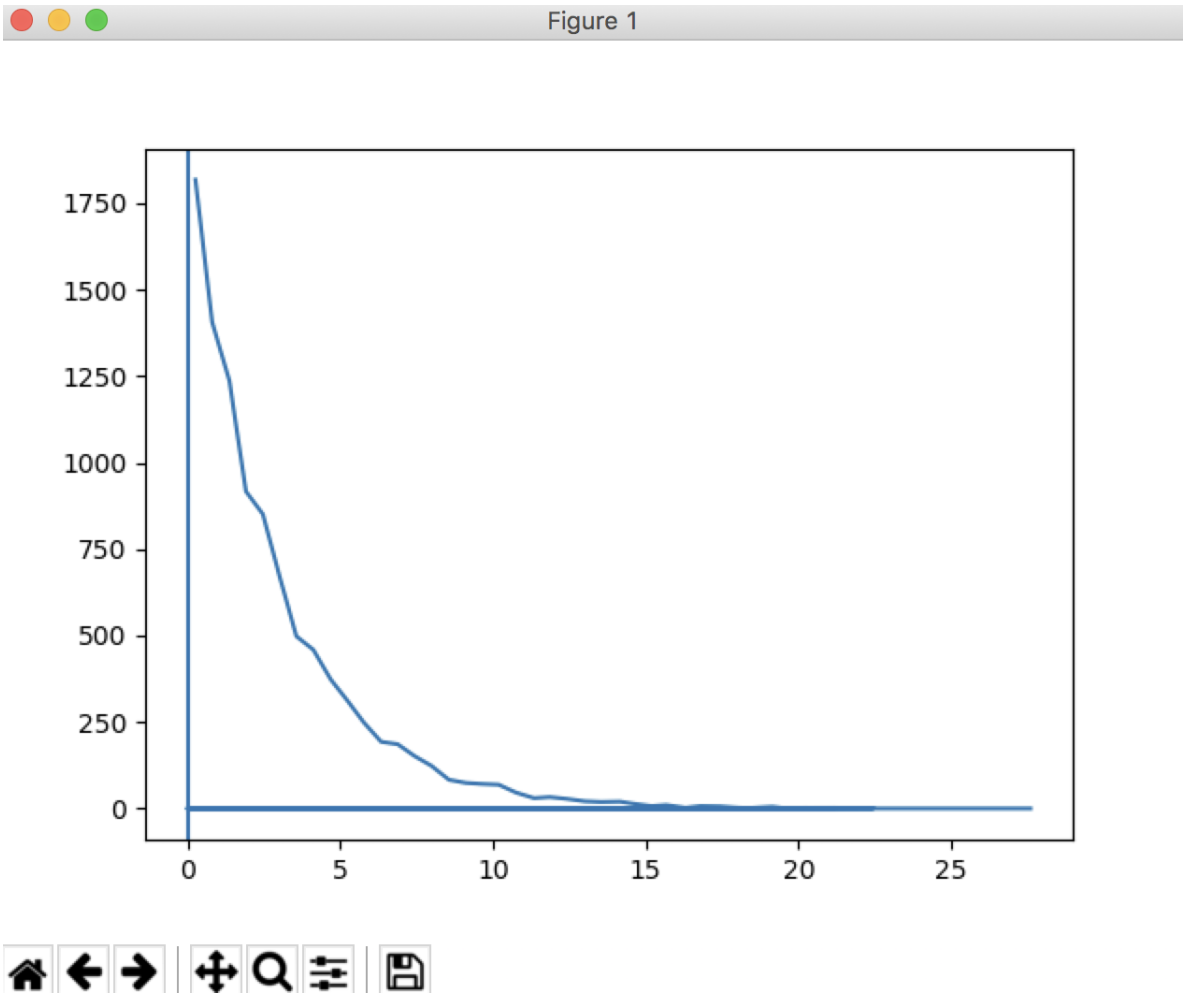

Figure 2: ExpType

Similarily, We can draw the figure of distribution service, It's easy to prove that it is a phase type distribution. and we can compare the value with its thereotical basis function
The distribution function of X is given by,

$$F(x) = 1 - \boldsymbol{\alpha} \exp(Sx)\mathbf{1}$$

and the density function,

$$f(x) = \boldsymbol{\alpha} \exp(Sx)\mathbf{S^0}$$

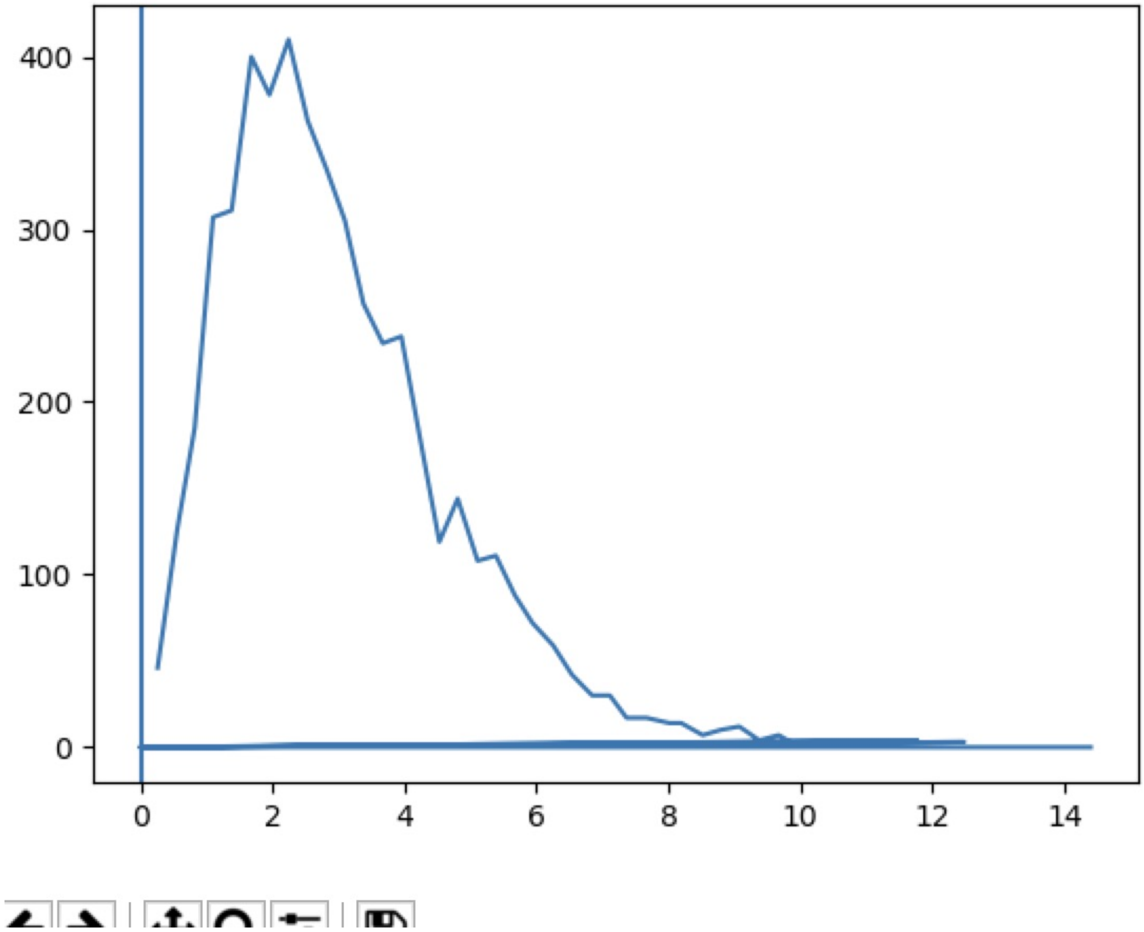

Figure 3: PhaseType

1.3 Assessment of the simulation code

1.3.1 Trace mode

```
In[1]: simulation('trace', [10,20,32,33], [1,2,3,4],3,50,100,50)
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
== Tuple of job arrive time and depart time :
== [('10.000', '61.000'), ('20.000', '63.000'), ('32.000', '66.000'), ('33.000', '70.000')]
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
Out[2]:
'41.250'
```

Figure 4: trace 01

```
In[2]: simulation('trace', [10,18,20,23,28,32,33,34,35,57,86,92], [2,4,14,5,6,21,2,16,9,4,15,9],3,50,100,50)
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
== Tuple of job arrive time and depart time :
== [('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')]
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
Out[3]:
'45.667'
```

Figure 5: trace 02

```
In[3]: simulation('trace', [1,1.1,11,11.2,11.3,13], [4,0.9,1,1.4,5,1],3,5,10,50)
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
== Tuple of job arrive time and depart time :
== [('1.100', '7.000'), ('1.000', '10.000'), ('11.000', '12.000'), ('11.200', '12.600'), ('13.000', '14.000'), ('11.300', '17.000')]
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
Out[4]:
'4.000'
```

Figure 6: trace 03

1.3.2 Random mode

```
In[6]: simulation('random',0.35, 1, 5, 5, 0.1, 100)
  ...:
Enter a value for seed: >? 1
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
== Tuple of job arrive time and depart time :
== [('0.412', '9.030'), ('2.648', '10.051'), ('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', '15.304'), ('15.811', '22.857'), ('29.807', '37.306'), ('33.459', '41.852'), ('38.522', '44.209'),
 ('44.740', '52.150'), ('44.640', '53.105'), ('48.727', '53.352'), ('45.283', '53.415'), ('46.625', '54.420'), ('54.489', '57.497'), ('50.283',
 '58.258'), ('48.854', '59.305'), ('56.549', '62.020'), ('57.446', '62.207'), ('61.820', '67.260'), ('66.552', '69.370'), ('66.717', '72.478'),
 ('68.724', '73.081'), ('71.014', '74.415'), ('70.933', '75.015'), ('77.543', '84.253'), ('77.874', '84.349'), ('78.371', '84.623'), ('77.495',
 '85.543'), ('81.992', '85.592'), ('81.923', '85.891'), ('76.653', '86.544'), ('84.979', '87.281'), ('82.321', '87.872'), ('85.432', '88.255'),
 ('88.671', '98.420')]
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
Out[7]:
'5.995'
```

Figure 7: rand 01

```
In[7]: simulation('random', 1, 2.7, 6, 5.5, 1.5, 100)
Enter a value for seed: >? 1
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
== Tuple of job arrive time and depart time :
== [('0.144', '6.984'), ('0.927', '7.317'), ('0.828', '7.516'), ('2.364', '7.677'), ('2.893', '7.984'), ('3.469', '8.313'), ('3.716', '8.346'),
 ('5.534', '8.881'), ('2.649', '9.019'), ('2.623', '9.262'), ('10.432', '11.358'), ('11.711', '13.394'), ('13.483', '14.356'), ('15.624', '16.907'),
  ('15.659', '17.800'), ('15.849', '18.960'), ('16.319', '19.995'), ('17.055', '20.440'), ('19.071', '22.463'), ('17.099', '22.737'), ('17.599',
 '22.976'), ('20.106', '24.017'), ('23.293', '24.075'), ('19.792', '24.138'), ('24.053', '24.641'), ('21.637', '24.678'), ('23.351', '25.168'),
 ('24.855', '25.349'), ('24.827', '25.766'), ('26.828', '28.640'), ('27.123', '29.769'), ('27.140', '30.403'), ('27.256', '30.949'), ('27.430',
 '31.413'), ('28.673', '32.019'), ('28.697', '32.480'), ('29.743', '33.267'), ('29.901', '33.626'), ('28.812', '33.683'), ('32.633', '34.017'),
 ('33.136', '34.457'), ('31.035', '34.515'), ('34.924', '35.234'), ('33.196', '35.860'), ('33.562', '35.916'), ('37.034', '38.895'), ('37.878',
 '41.063'), ('39.096', '41.660'), ('39.326', '42.365'), ('39.437', '43.158'), ('39.830', '44.777'), ('40.469', '45.954'), ('43.160', '46.205'),
 ('40.054', '46.918'), ('43.911', '47.259'), ('43.822', '48.100'), ('45.334', '48.109'), ('45.750', '48.478'), ('48.946', '49.734'), ('48.836',
 '49.995'), ('50.498', '51.659'), ('52.020', '52.603'), ('52.106', '53.846'), ('54.697', '55.617'), ('56.456', '56.907'), ('56.578', '57.825'),
 ('61.014', '66.830'), ('62.058', '67.936'), ('63.006', '67.983'), ('61.291', '68.230'), ('67.227', '68.765'), ('66.860', '69.032'), ('61.766',
 '69.150'), ('62.099', '69.192'), ('65.695', '70.077'), ('65.453', '70.572'), ('71.324', '72.317'), ('74.150', '80.116'), ('75.219', '80.577'),
 ('76.020', '81.007'), ('74.938', '81.063'), ('74.531', '81.690'), ('76.531', '81.881'), ('81.235', '82.142'), ('76.364', '82.288'), ('79.512',
 '83.241'), ('83.753', '84.769'), ('86.166', '87.183'), ('87.795', '89.864'), ('88.965', '90.210'), ('90.220', '90.760')]
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
Out[8]:
'3.399'
```

Figure 8: rand 02

```
In[8]: simulation('random', 0.8, 2.3, 4, 6, 0.8, 100)
Enter a value for seed: >? 1
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
== Tuple of job arrive time and depart time :
== [('0.180', '7.753'), ('1.159', '8.203'), ('1.035', '8.429'), ('2.954', '8.566'), ('3.617', '8.927'), ('4.645', '9.380'), ('4.337', '9.454'),
 ('3.311', '10.194'), ('6.917', '10.270'), ('3.279', '10.487'), ('13.040', '20.127'), ('16.854', '21.663'), ('14.638', '22.104'), ('19.574',
 '23.151'), ('19.530', '23.170'), ('21.318', '23.692'), ('19.811', '24.215'), ('20.398', '24.367'), ('23.839', '25.674'), ('21.999', '26.348'),
 ('21.374', '26.388'), ('24.740', '27.641'), ('25.132', '28.065'), ('27.046', '29.324'), ('29.116', '30.242'), ('29.189', '31.593'), ('30.067',
 '32.258'), ('31.033', '33.361'), ('31.069', '33.941'), ('33.535', '36.068'), ('33.904', '37.393'), ('33.925', '38.137'), ('34.070', '38.778'),
 ('34.287', '39.323'), ('35.842', '40.035'), ('35.871', '40.076'), ('37.178', '40.790'), ('37.376', '41.228'), ('36.015', '41.317'), ('40.792',
 '41.672'), ('38.794', '42.141'), ('41.420', '42.395'), ('41.496', '44.051'), ('41.953', '44.183'), ('43.655', '44.415'), ('46.292', '54.477'),
 ('48.870', '55.178'), ('49.158', '55.698'), ('47.348', '55.893'), ('49.296', '56.088'), ('49.788', '57.078'), ('50.586', '57.275'), ('53.949',
 '57.707'), ('50.068', '58.424'), ('54.888', '58.512'), ('56.668', '59.106'), ('54.778', '59.598'), ('57.188', '59.855'), ('61.182', '68.107'),
 ('61.045', '68.405'), ('65.026', '69.090'), ('63.122', '69.470'), ('68.371', '70.202'), ('65.132', '70.549'), ('70.570', '71.100'), ('70.723',
 '71.800'), ('76.268', '82.638'), ('77.573', '83.994'), ('76.613', '84.303'), ('78.758', '84.445'), ('77.207', '85.361'), ('77.623', '85.491'),
 ('84.034', '85.794'), ('83.575', '86.302'), ('82.119', '86.904'), ('81.816', '87.398'), ('89.156', '96.321'), ('92.688', '96.868'), ('93.164',
 '98.815'), ('94.024', '98.978'), ('95.025', '99.483'), ('93.673', '99.799')]
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
Out[9]:
'4.620'
```

Figure 9: rand 03

Part 2 - Demonstration of the code

2.1 Reproducible property demonstration

The property of reproducible result means the arrival lists and service lists generated in random mode are all the same in duplicate experiments when we only change the value of *time_end*, Here are the result of my tests. The seed for generating random is 1, the value of *time_end* is set to 12.

```
In[4]: transient_test(12)
arrival = [0.41226018317002633]
service = [3.6175889076615513]

arrival = [0.41226018317002633, 2.3667138393403127]
service = [3.6175889076615513, 3.205882828163837]

arrival = [0.41226018317002633, 2.3667138393403127, 2.6483167063750424]
service = [3.6175889076615513, 3.205882828163837, 2.402199851942232]

arrival = [0.41226018317002633, 2.3667138393403127, 2.6483167063750424, 6.753065607385788]
service = [3.6175889076615513, 3.205882828163837, 2.402199851942232, 1.870074302492911]

arrival = [0.41226018317002633, 2.3667138393403127, 2.6483167063750424, 6.753065607385788, 7.49523289293945]
service = [3.6175889076615513, 3.205882828163837, 2.402199851942232, 1.870074302492911, 5.2533861361101994]

arrival = [0.41226018317002633, 2.3667138393403127, 2.6483167063750424, 6.753065607385788, 7.49523289293945, 7.568876338372012]
service = [3.6175889076615513, 3.205882828163837, 2.402199851942232, 1.870074302492911, 5.2533861361101994, 4.058913815881857]

arrival = [0.41226018317002633, 2.3667138393403127, 2.6483167063750424, 6.753065607385788, 7.49523289293945, 7.568876338372012, 8.266336590359733]
service = [3.6175889076615513, 3.205882828163837, 2.402199851942232, 1.870074302492911, 5.2533861361101994, 4.058913815881857, 0.8284864532970078]

arrival = [0.41226018317002633, 2.3667138393403127, 2.6483167063750424, 6.753065607385788, 7.49523289293945, 7.568876338372012, 8.266336590359733, 9.912203555474374]
service = [3.6175889076615513, 3.205882828163837, 2.402199851942232, 1.870074302492911, 5.2533861361101994, 4.058913815881857, 0.8284864532970078, 1.2126759081328666]

arrival = [0.41226018317002633, 2.3667138393403127, 2.6483167063750424, 6.753065607385788, 7.49523289293945, 7.568876338372012, 8.266336590359733, 9.912203555474374, 10
 .617631574082765]
service = [3.6175889076615513, 3.205882828163837, 2.402199851942232, 1.870074302492911, 5.2533861361101994, 4.058913815881857, 0.8284864532970078, 1.2126759081328666, 0
 .9793588286236408]
```

Figure 10: verification

2.2 transient removal

We set the *time_end* as 10000, everytime when the number of jobs increase by 1, we record the arrival list, and we get a series of arrival list(approximately 3500 lists). We plot every corresponding mrt in the following figure.
By observation, we choose $w = 480$, which means we remove the first 480 jobs. The remain jobs are under steady state

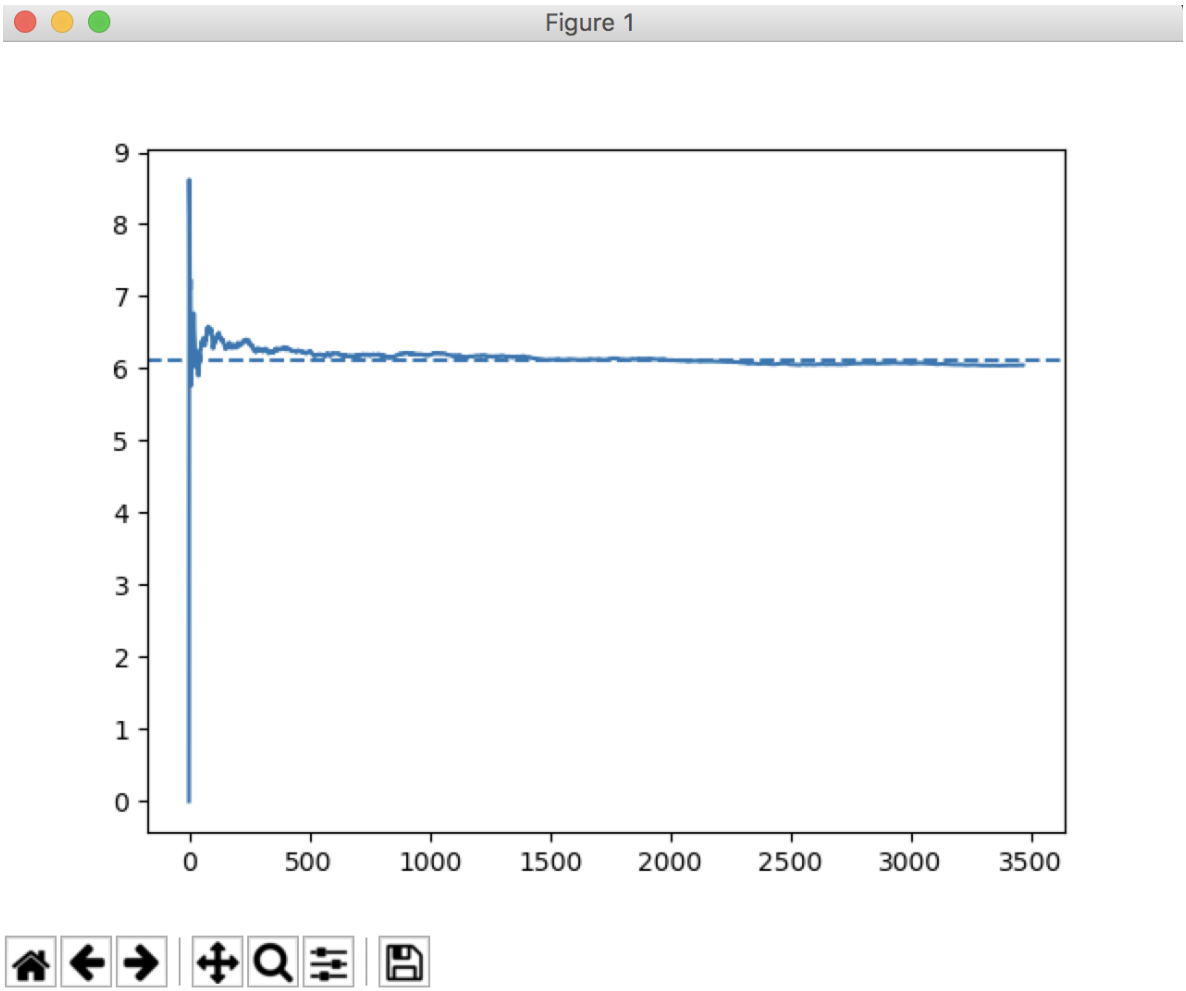

Figure 11: mrt_vary

Part 3 - Determination a suitable value of T_c

3.1 Optimal choice of parameters

3.1.1 length of the simulation

Because determining a not too large and not too small *time_end* can generate enough job, thus the accuracy of transient removal can be evident. So we choose the length of executing time of *time_end* as 10000, And for tc test times, Firstly, we select a raw number for starting the loop, Based on observation, we find when *tc* is at 9.5, compared with the *baseline_mrt*, the difference of $baseline\_mrt - modified\_mrt$ approach nearly 2 units. when *tc* is at 11 the difference approach nearly larger than 2.1, So everytime we increase the *tc_chosen* by 0.05, calculate the corresponding confidence interval.

3.1.2 number of replications

It's obvious that if we apply more testing, the result will be more accurate. we select $deg = 30$ replications.

3.1.3 end of transient

Based on 2.2 we can preprocess the job list by a transient removal procedure. $w = 480$

3.2 The analysis of a proper interval for T_c

In order to determine a proper tc, we need to calculate multiple confidence interval for observation. After transient removal, we should compute the following parameter for each test. The sample mean:

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

The standard deviation:

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

The list for $t_{deg-1,1-(\alpha/2)}$ we denote deg as defaultly 30, thus we would calculate $(1-\alpha)$ confidence interval, where we are required to compute 95% confidence interval, so $\alpha = 0.05$ and we should use the $deg - 1 = 29$-th item in the list and $1 - \alpha/2 = 0.975$ part of the ref list.

```
[12.706, 4.303, 3.182, 2.776, 2.571,
2.447, 2.365, 2.306, 2.262, 2.228,
2.201, 2.179, 2.160, 2.145, 2.131,
2.120, 2.110, 2.101, 2.093, 2.086,
2.080, 2.074, 2.069, 2.064, 2.060,
2.056, 2.052, 2.048, 2.045, 2.042
]
```

Figure 12: valuelist

The confidence interval:

$$[\hat{T} - \frac{\hat{S}}{\sqrt{(deg)}} \cdot t_{deg-1,1-(\alpha/2)}, \hat{T} + \frac{\hat{S}}{\sqrt{(deg)}} \cdot t_{deg-1,1-(\alpha/2)}]$$

We write our records to a ".xls" file and demonstrate our testing result through the table generated.
Thus, we calculate the proper interval for choice of tc is $A = [10.05, 10.9]$
Moreover, I find when $tc = 63$, the function tend to be restrained, but we should pick A as the appropriate experimental result.

A	B	C
TC	Confidence_interval	Status
9.5	[1.955962735556349, 1.988903931110319]	Outside
9.55	[1.9602049011302198, 1.9934617655364457]	Outside
9.6	[1.9646938322760028, 1.9975061677239974]	Outside
9.65	[1.968649130201116, 2.0020175364655506]	Outside
9.7	[1.972319546989554, 2.005480453010446]	Outside
9.75	[1.9772462261562123, 2.010353773843787]	Outside
9.8	[1.9819528104176025, 2.0151805229157302]	Outside
9.85	[1.9864930554623936, 2.0203069445376065]	Outside
9.9	[1.9910198080808592, 2.023380191919141]	Outside
9.95	[1.9948048670916034, 2.026861799575062 4]	Outside
10	[1.9995046721582654, 2.0315619945084014]	Outside
10.05	[2.0064462029519485, 2.0376871303813853]	Left
10.1	[2.010851171840119, 2.0409488281598818]	Inside
10.15	[2.014783282010225, 2.0451500513231093]	Inside
10.2	[2.018671460392804, 2.048328539607197]	Inside
10.25	[2.0226385419014643, 2.0526281247652025]	Inside
10.3	[2.027219961662521, 2.057780038337479]	Inside
10.35	[2.032752319019876, 2.0633810143134563]	Inside
10.4	[2.0366819846690722, 2.066518015330928]	Inside
10.45	[2.0414316742155822, 2.0709016591177516]	Inside
10.5	[2.0456863913633687, 2.074580275303298]	Inside
10.55	[2.0491184581957897, 2.07828154180421]	Inside
10.6	[2.051558884175642, 2.080507782491024]	Inside
10.65	[2.0548450712650554, 2.0844882620682768]	Inside
10.7	[2.058489174781377, 2.0885774918852893]	Inside
10.75	[2.061889405022355, 2.091377261644312]	Inside
10.8	[2.0652659954102583, 2.0954006712564075]	Inside
10.85	[2.0693942944577834, 2.099605705542217]	Inside
10.9	[2.0732243589441173, 2.1042423077225494]	Right
10.95	[2.0775148930205822, 2.108551773646085]	Outside
11	[2.08030419121978, 2.1110958087802216]	Outside

Figure 13: valuelist

Reference

[1] Anshul Gandhi, Sherwin Doroudi, Mor Harchol-Balter, Alan Scheller-Wol. *Exact analysis of the M/M/k/setup class of Markov-chains via recursive renewal reward.* Queueing Systems, 2014, Volume 77, Number 2, Page 177

[2] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design.* Measurement, Simulation, and Modeling, Wiley, 1991.

[3] Dimitri Bertsekas and Robert Gallager *Data Networks.* Prentice Hall, Second Edition, 1992.

[4] Averill M. Law and W. David Kelton *Simulation Modeling and Analysis.* McGraw-Hill, Second Edition, 1991.

[5] Wayne L. Winston *Operations Research: Applications and Algorithms.* Duxbury Press. Third Edition, 1994.

[6] Chun Tung Chou: Capacity Planning, `https://webcms3.cse.unsw.edu.au/COMP9334/18s1/`