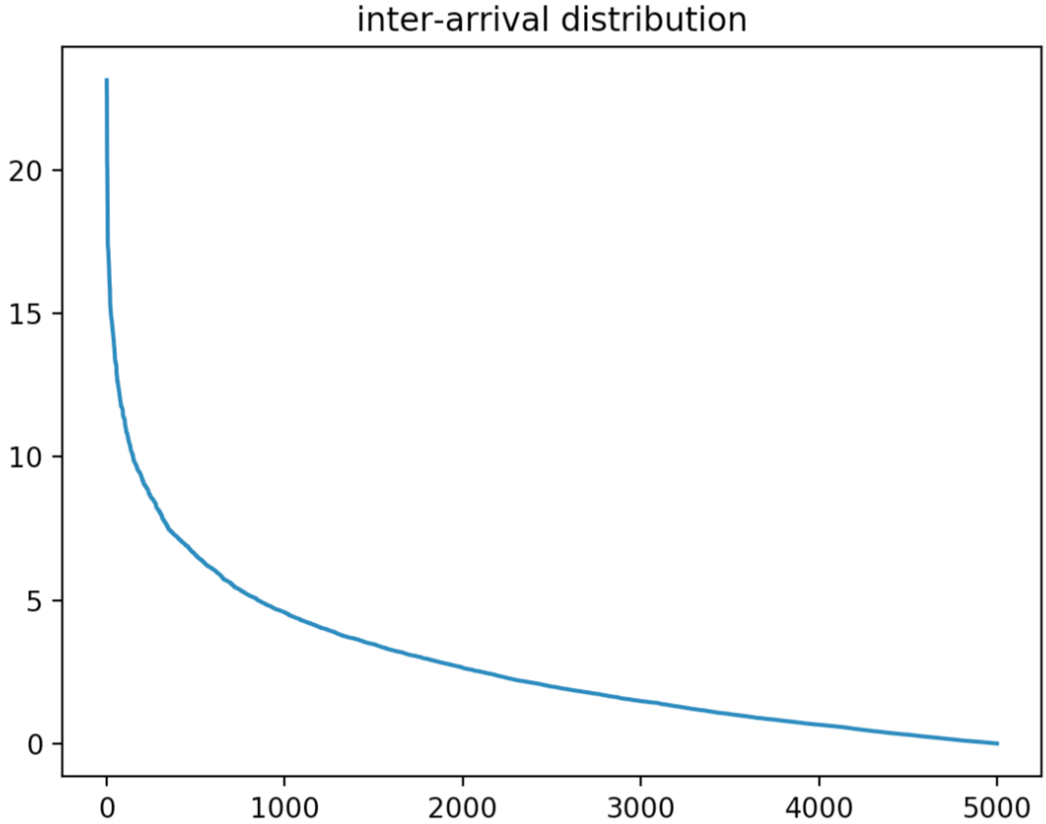
Verify the correctness of the inter-arrival probability distribution:

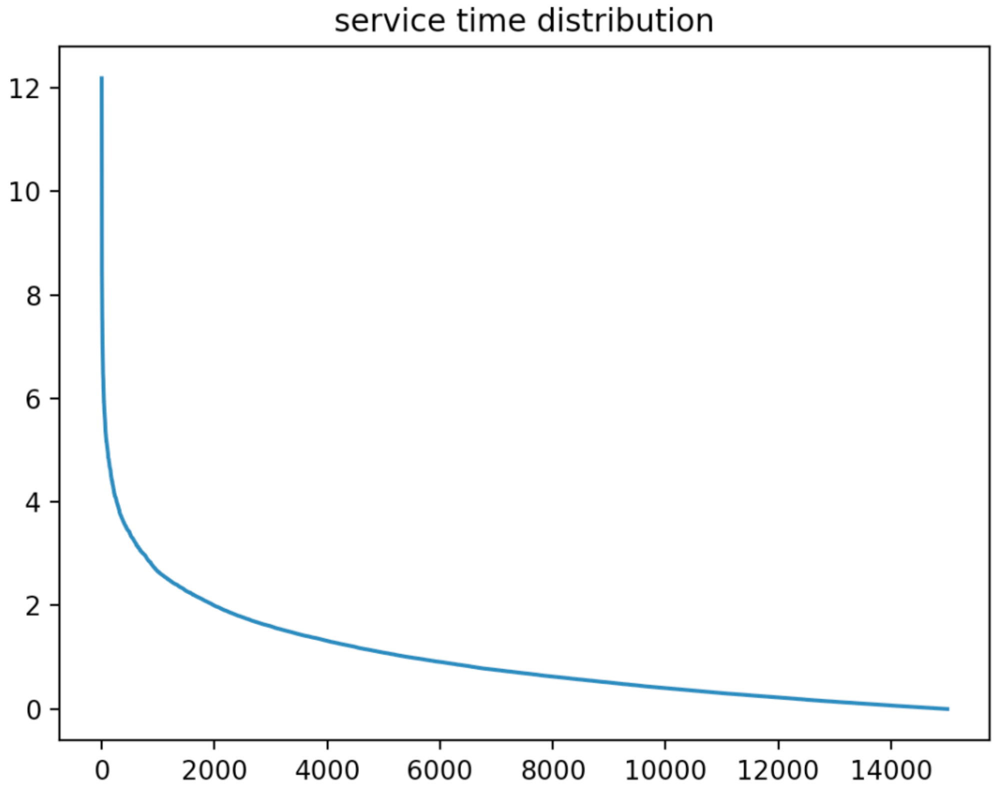
By drawing the diagram it shows how the inter-arrival looks like. From the diagram we can obtain that it's an exponential distribution

```
import random
import math
import matplotlib.pyplot as plt
Lambda=0.35
mu=1
inter_arrival=[]
random.seed(2)
x=[]
for i in range(5000):
    a = -(math.log(1-random.random()))/Lambda
    inter_arrival.append(a)
for j in range(len(inter_arrival)):
    x.append(j)
inter_arrival.sort()
inter_arrival.reverse()
plt.plot(x,inter_arrival)
plt.show()
```


inter-arrival distribution
20
15
10
5
0
0
1000
2000
3000
4000
5000

By drawing the diagram it shows how the service time looks like. From the diagram we can obtain that it’s an exponential distribution

```
import random
import math
import matplotlib.pyplot as plt
Lambda=0.35
mu=1
inter_arrival=[]
service=[]
random.seed(2)
x=[]

for i in range(5000):
    for _ in range(0,3):
        b=0
        b = b-(math.log(1-random.random()))/mu
        service.append(b)
for j in range(len(service)):
    x.append(j)
service.sort()
service.reverse()
plt.plot(x,service)
plt.show()
```


service time distribution
12
10
8
6
4
2
0
0
2000
4000
6000
8000
10000
12000
14000

Verify the correctness of the simulation code:

```python
import ass
### Obtain the number of test
with open('num_tests.txt','r') as num:
    number_of_test = num.readline()

###  Obtain what the mode will be
for i in range(1,int(number_of_test)+1):
    with open('mode_'+str(i)+'.txt','r') as arrival:
        mode = arrival.readline()
    

    with open('para_'+str(i)+'.txt','r') as paramental:
        paramental_list = [line.strip() for line in paramental]
         ## when mode is trace, obtain the parameters
        if len(paramental_list) == 3:
            m = int(paramental_list[0])
            setup_time = float(paramental_list[1])
            delayoff_time = float(paramental_list[2])
         ## when mode is random, obtain the parameters
        if len(paramental_list) == 4:
            m = int(paramental_list[0])
            setup_time = float(paramental_list[1])
            delayoff_time = float(paramental_list[2])
            Time_end = float(paramental_list[3])
    ## Obtain the arrival time and service time when trace mode
    if mode == "trace":
        with open('arrival_'+str(i)+'.txt','r') as arrival:
            Arrival_time_before = [line.strip()for line in arrival]
            Arrival_time = list(map(eval,Arrival_time_before))
        with open ('service_'+str(i)+'.txt','r') as service:
            Service_time_before = [line.strip()for line in service]
            Service_time = list(map(eval,Service_time_before))
```

```python
## Obtain the lambda and mu when random mode
if mode =="random":
    with open('arrival_'+str(i)+'.txt','r') as arrival:
        Lambda = float(arrival.readline())
    with open('service_'+str(i)+'.txt','r') as service:
        mu = float(service.readline())
## create the mrt.txt and departure.txt when trace
if mode=="trace":
    ## use Simulation function in ass.py
    mrt,arrrival_and_departure=ass.Simulation(Arrival_time,Service_time,m,setup_time,delayoff_time)
    ## create mrt file to write the mean response time
    with open('mrt_'+str(i)+'.txt','w') as f:
        f.write(str('%.3f' % mrt))
    ## crete departure file to write departure time and coresponding arrival time
    with open('departure_'+str(i)+'.txt','w') as w:
        for key,value in arrrival_and_departure.items():

            w.write(str('%.3f' % value)+'    '+str('%.3f' % key))
            w.write('\n')

## create the mrt.txt and departure.txt when random
if mode =="random":
    ## use Simulation_random function in ass.py
    mrt,arrrival_and_departure,response_time_every_job = ass.Simulation_rendom(Lambda,mu,m,setup_time,delayoff_time,Time_end)
    ## create mrt file to write the mean response time
    with open('mrt_'+str(i)+'.txt','w') as f:
        f.write(str('%.3f' % mrt))
    ## crete departure file to write departure time and coresponding arrival time
    with open('departure_'+str(i)+'.txt','w') as w:
        for key,value in arrrival_and_departure.items():
            w.write(str('%.3f' % value)+'    '+str('%.3f' % key))
            w.write('\n')
```

In the wrapper.py, it read the number_tests.txt file to determine the number of test, and then read all the configuration files one by one to determine all the parameters it needed, then call different function according to the different mode

For example:

mode_1.txt

```
trace
```

para_1.txt — 已编辑

```
3
50
100
```

service_1.txt

```
1
2
3
4
```

arrival_1.txt

```
10
20
32
33
```

Since the mode is trace, the Simulation() function will be called, which will return the mean response time and departure time with it corresponding arrival time

```python
def Simulation(Arrival_time,Service_time,m,setup_time,delayoff_time):
    #systm [0:OFF 1:SetUp 2:BUSY 3:DelayedOFF]
    response_time_cumlative = 0
    num_customer_served = 0

    Master_Clock = 0

    server_state=[0 for _ in range(m)]
    next_departure_arrival_time=[0 for _ in range(m)]
    Dispatcher = []

    arrrival_and_departure ={}


    next_departure_time = [float("inf") for _ in range(m)]

    Set_up_finish_time = [float("inf") for _ in range(m)]

    Expiry_time = [float("inf") for _ in range(m)]

    UNMARKED=[]

    while Arrival_time.count(float("inf")) != len(Arrival_time) or  Set_up_fini

    avg_response_time = response_time_cumlative / num_customer_served


    return avg_response_time, arrrival_and_departure
```

Then the returned result will be given to mrt and arrrival_and_departure, which will be used to create mrt.txt and departure.txt respectively:

```
mrt,arrrival_and_departure=ass.Simulation(Arrival_time,Service_time,m,setup_time,delayoff_time)
## create mrt file to write the mean response time
with open('mrt_'+str(i)+'.txt','w') as f:
    f.write(str('%.3f' % mrt))
## crete departure file to write departure time and coresponding arrival time
with open('departure_'+str(i)+'.txt','w') as w:
    for key,value in arrrival_and_departure.items():

        w.write(str('%.3f' % value)+'    '+str('%.3f' % key))
        w.write('\n')
```

mrt_1.txt

```
41.250
```

departure_1.txt

```
10.000    61.000
20.000    63.000
32.000    66.000
33.000    70.000
```

In general, when the trace mode, number of service =3, set_up_time = 50, Tc=100, Arrival time = [10,20,32,33] and Service time=[1,2,3,4], the Mean response time = 41.250 and the departure time according to the arrival time is shown as departure_1.txt, which are all correct according to the Section 3.2 test.

Besides, when the mode is random, the Simulation_rendom() function will be called, which will also return the mean response time and departure time with it corresponding arrival time

For example:

mode_4.txt

```
random
```

para_4.txt

```
5
5
0.1
15000
```

service_4.txt

```
1
```

arrival_4.txt

```
0.35
```

```python
def Simulation_rendom(Lambda,mu,m,setup_time,delayoff_time,Time_end):
    #systm [0:OFF 1:SetUp 2:BUSY 3:DelayedOFF]
    response_time_cumlative = 0
    num_customer_served = 0
    Master_Clock = 0
    server_state=[0 for _ in range(m)]
    next_departure_arrival_time=[0 for _ in range(m)]
    Dispatcher = []
    arrrival_and_departure ={}
    response_time_every_job=[]
    next_departure_time = [float("inf") for _ in range(m)]
    Service_time =[]
    Arrival_time =[]
    Set_up_finish_time = [float("inf") for _ in range(m)]
    Expiry_time = [float("inf") for _ in range(m)]

    UNMARKED=[]

    #################
    random.seed(2)

    a = -(math.log(1-random.random()))/Lambda
    b=0
    for _ in range(0,3):
        b = b-(math.log(1-random.random()))/mu

    Arrival_time.append(a)
    Service_time.append(b)

    while Master_Clock < Time_end:⋯

    avg_response_time = response_time_cumlative / num_customer_served
    return avg_response_time, arrrival_and_departure,response_time_every_job
```

Then the returned result will be given to mrt and arrrival_and_departure, which will be used to create mrt.txt and departure.txt respectively:

```python
## create the mrt.txt and departure.txt when random
if mode =="random":
    ## use Simulation_random function in ass.py
    mrt,arrrival_and_departure,response_time_every_job = ass.Simulation_rendom(Lambda,mu,m,setup
    ## create mrt file to write the mean response time
    with open('mrt_'+str(i)+'.txt','w') as f:
        f.write(str('%.3f' % mrt))
    ## crete departure file to write departure time and coresponding arrival time
    with open('departure_'+str(i)+'.txt','w') as w:
        for key,value in arrrival_and_departure.items():
            w.write(str('%.3f' % value)+'    '+str('%.3f' % key))
            w.write('\n')
```

mrt_4.txt

```
6.070
```

departure_4.txt

```
0.412    9.030
2.367    10.573
2.648    10.051
6.753    10.900
6.827    14.109
7.524    11.401
8.230    11.879
22.226    29.725
24.286    33.809
25.182    33.674
29.556    35.422
31.564    35.203
37.202    47.094
39.861    46.598
40.313    47.976
43.552    51.347
44.988    49.336
46.033    53.105
52.060    58.130
59.749    68.473
60.937    69.228
69.754    77.883
70.103    77.582
82.778    88.631
83.470    94.125
86.800    90.795
87.849    91.492
95.923    102.181
100.847    103.401
```

In general, when the random mode, number of service =5, set_up_time = 5, Tc=0.1, Arrival time is according to Lambda=0.35 , Service time is according to mu = 1, and Time_end =15000, the Mean response time = 6.070 and the departure time according to the arrival time is shown as departure_4.txt

Demonstrate reproducible:

To prove the random simulation is reproducible, I write a program which will write the mean response time and the system time to a file, it demonstrate that when the seed remain the same, the simulation function will obtain the same result

```python
import ass
import datetime

mrt=ass.Simulation_rendom(0.35,1,5,5,0.1,15000)[0]
time_now= datetime.datetime.now()
with open("reproducible_test.txt","w") as w:
    w.write(str('%.3f' % mrt)+'     '+str( time_now))
```

reproducible_test.txt

```
6.086     2018-05-16 02:44:21.911277
```

reproducible_test.txt

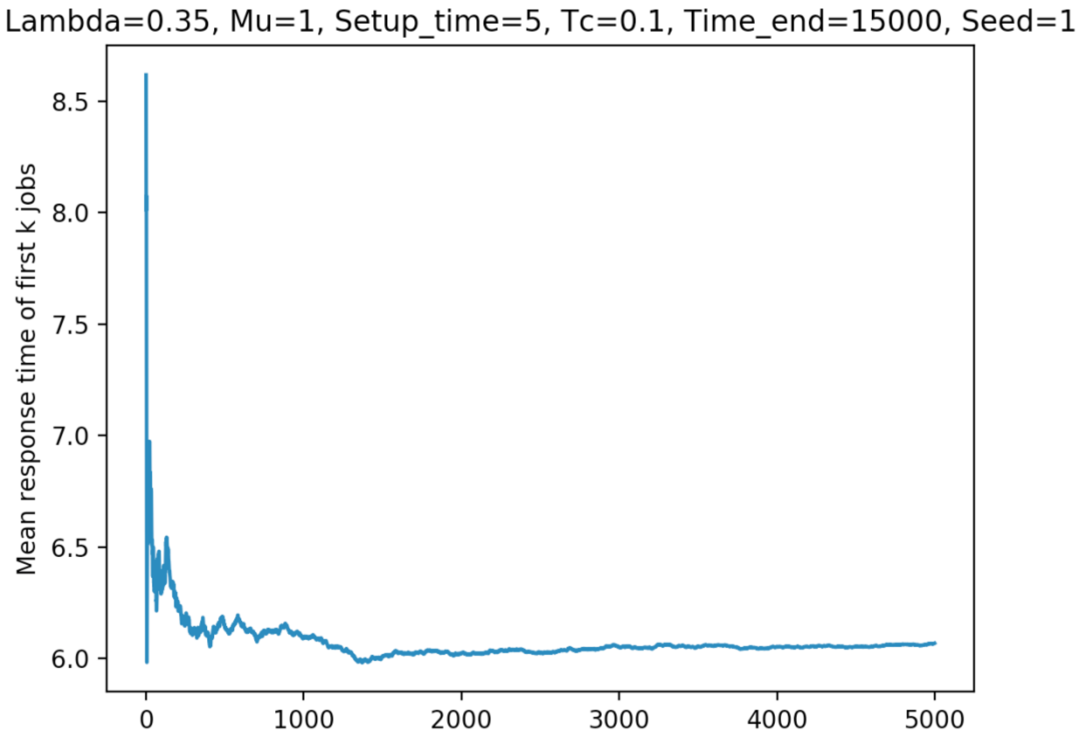
```
6.086     2018-05-16 02:51:22.007828
```

Evidence of using statistical sound methods to analyze simulation result:

In the random simulation in the baseline given in the project, with the Time_end =15000, I obtain the response time of every job, then calculate the mean response time of first 5000 jobs since the number of departure jobs are slightly bigger than 5000 with different seed. the code as below:

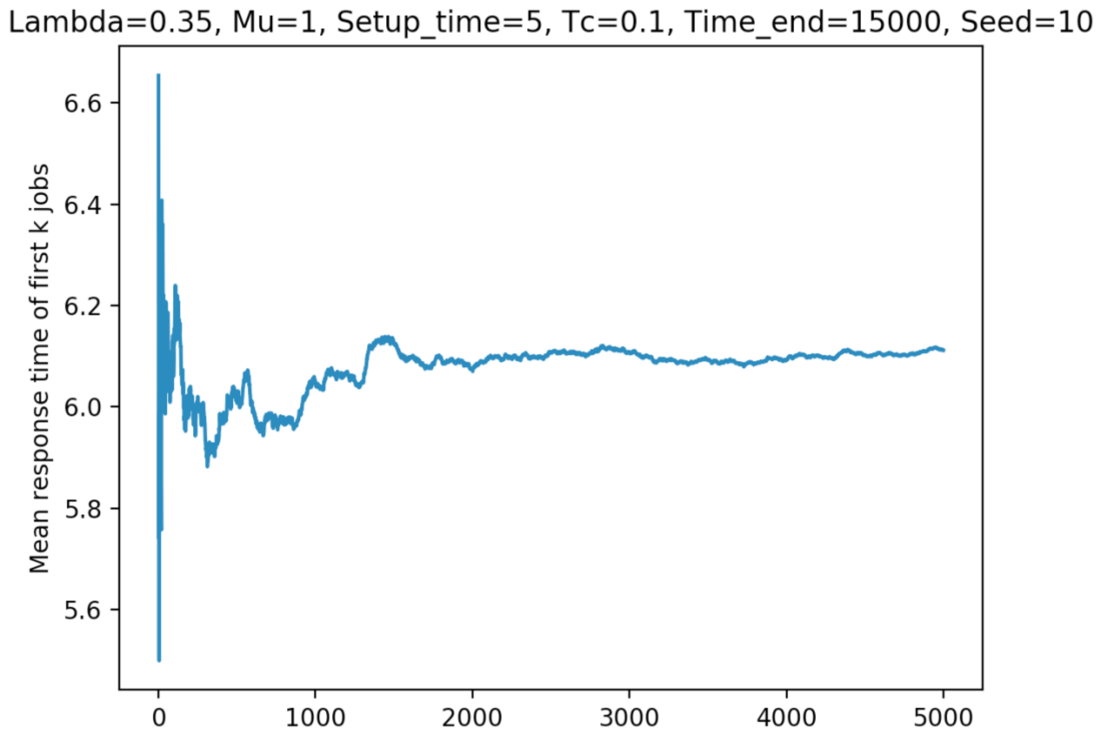
```
import math
import ass
import numpy as np
import matplotlib.pyplot as plt
Lambda=0.35
mu=1
setup_time = 5
#Tc=[0.1,10.1]
Tc=0.1
time_end = 15000
L1=[]
L2=[]
response_time_every_job=[]
k_response_time=[]
ass.random.seed(1)
response_time_every_job=ass.Simulation_rendom(Lambda,mu,5,setup_time,Tc,time_end)[2]
with open ("response.txt",'w') as respon:
    for i in response_time_every_job:
        respon.write(str(i))
        respon.write('\n')
b=0
x=[]
with open("response.txt",'r') as trace:
    response_time= trace.readlines()[0:5000]
for i in range(0,5000):
    b=b+float(response_time[i])
    k_response_time.append(b/(i+1))
    x.append(i)
plt.ylabel("Mean response time of first k jobs")
plt.title("Lambda="+str(Lambda)+ ", Mu="+str(mu)+ ", Setup_time="+str(setup_time)+", Tc="+str

plt.plot(x,k_response_time)
plt.show()
```

Then I draw the diagram to show the mean response time of the first 5000 jobs (running mean) and the diagram is as below:

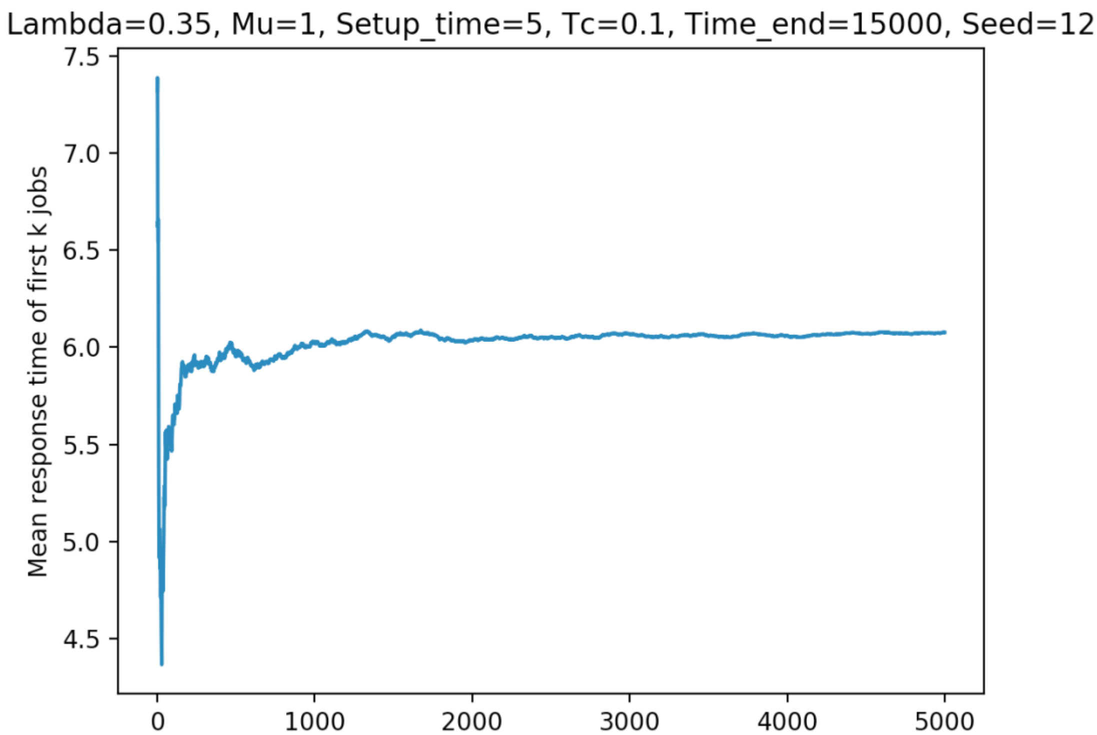


Form the diagram we can observe that the transient behavior occur at the first 2000 jobs, then after that is the state behavior.

And when seed = 10 , the diagram looks like below:



and when seed = 12:



From the diagram we can observe that the first 2000 jobs should be removed to get a steady state.

Then I choose the new Tc=10.1, also remove the first 2000 jobs to get a steady state, choose the seed from 1 to 20, which means the number of replications is 20, because when I star with 5 replications, the accuracy is low, with increasing the number of replications, it reach the desired level of accuracy in 20. then compute the 95% confidence interval of the response time of old system minus new system

```
import math
import ass
import numpy as np
import matplotlib.pyplot as plt
Lambda=0.35
mu=1
setup_time = 5
#Tc=[0.1,10.1]
Tc=[0.1,10.1]
time_end = 15000
L1=[]
L2=[]
for i in range(1,21):
    b=0
    k_response_time=[]
    response_time=[]
    ass.random.seed(i)
    response_time_every_job=ass.Simulation_rendom(Lambda,mu,5,setup_time,Tc[0],time_end)[2]
    with open ("response.txt",'w') as respon:
        for i in response_time_every_job:
            respon.write(str(i))
            respon.write('\n')


    with open("response.txt",'r') as trace:
        response_time= trace.readlines()[2000:5000]

    for j in range(0,len(response_time)):
        b=b+float(response_time[j])
    mrt=b/len(response_time)
    L1.append(float(mrt))
```

```python
for i in range(1,21):
    b=0
    k_response_time=[]
    response_time=[]
    ass.random.seed(i)
    response_time_every_job=ass.Simulation_rendom(Lambda,mu,5,setup_time,Tc[1],time_end)[2]
    with open ("response.txt",'w') as respon:
        for i in response_time_every_job:
            respon.write(str(i))
            respon.write('\n')


    with open("response.txt",'r') as trace:
        response_time= trace.readlines()[2000:5000]

    for j in range(0,len(response_time)):
        b=b+float(response_time[j])
    mrt=b/len(response_time)
    L2.append(float(mrt))
D=[]
for k in range(20):
    D.append(L1[k]-L2[k])
avrg=sum(D)/20
SD=0
for u in range(20):
    SD+=(avrg-D[u])**2
SD=math.sqrt(SD/19)
cr1=avrg-1.729*SD/math.sqrt(20)
cr2=avrg+1.729*SD/math.sqrt(20)
print((cr1,cr2))
```

```
appledeMacBook-Pro-2:9334 apple$ python c2.py
(2.0021699078530433, 2.054207417343943)
```

The output (cr1,cr2) here indicates that 95% probability that new system(with Tc=10.1) can be two units less than the baseline system. I choose Tc=10.1 is because 10.1 is the smallest it can get, when Tc=10, the output (cr1,cr2) would be like below:

```
(1.9979712435497303, 2.05116903482896)
```

Is not two units less than the baseline system in this interval.