Project 1 - Markovian Queue Implementation

Harshavardhan Nalajala

October 23 2017

Contents

1 Description

Purpose of this project is to implement and run simulation of finite population, finite capacity, finite servers continuous time markov process system. Service times of servers are exponential. Arrival rate is poisson with inter arrival times being exponential. Assumptions:

- Service times of all servers are equal and there is no priority for a request arriving.
- A terminal is blocked when there is a request corresponding to the terminal either being processed or in queue.

2 Implementation

Following is the implementation detail with L(no. of terminals), K(capacity of the system including queue size and requests in service), m(no. of servers), mu(service time of all servers)

- For lambda ranging between 0.01*mu*m to 1*mu*m
  - Generate L requests with rate lambda and add them to min heap.
  - while number of departures is not total(100000 for testing purposes)
    - get the top event from min heap i.e. min element in heap (ordered by time).
    - If the request in event is ARRIVAL:
      - If system size is K, drop the request and generate arrival event for the corresponding terminal, and add this new event to heap.
      - If system size is less than K, and greater than m, add the request to queue (requests are ordered by time of arrival).
      - If system size is less than m, generate departure event and add to min heap.
    - If the request in event is DEPARTURE:
      - If there are requests in queue, generate departure event to the first request in queue, and add to min heap.
      - If total departure is less than TOTAL, generate arrival event for the corresponding terminal and add to min heap.
      - If total departure is TOTAL, exit.

3 Data Structures

- Min Heap to store the requests ordered by time. Event that occurs in the most near future is stored at the root of the heap. Max heap size

is dependent on the requests generated. In the current project, heap size is dependent on L(no. of terminals) since each terminal cannot generate another request until current request from the terminal is processed by the system.

- Queue to store the requests that arrived at the system but servers are busy. Max queue size is order of K(Queue size).

4 Performance Statistics

Statistics collection consists of

- Expected Number in the system: Each time arrival or departure happens, EN is given by (number in the system)*(current clock - prev event clock). Expected number is given by EN/totalClock at the end of simulation.
- Expected Time in the system: Each time departure happens, ET is given by (event departure time - event arrival time). Expected time is given by ET/totalDeparture at the end of simulation.
- blocking probability: no. of drops/total no. of arrival requests
- utilization: System busy time is calculated based on the number of requests in the system. If the no. of requests in the system is greater than or equal to no. of servers, system busy time is calculated as (no. of servers)*(current time - prev event time). If the no. of requests in the system is less than no. of servers, system busy time is calculated as (no. of requests in the system) * (current time - prev event time). Utilization is set to system busy time divided by (total servers * total clock) at the end of simulation.

5 ReadMe

- main.c contains the main simulation program (run_simulation API).
- utils.c is used to store data structures and generate exponential random variables.
- utils.h contains the structures and API used by main.c and implemented in utils.c

- input.in contains the input to be given to the program. Following is the input style.
  - no. of events to be generated for each run, 'run'
  - no. of terminals, L
  - queue size, K
  - no. of servers, m
  - service time of each server, 'mu'
- plotter.py python script to generate the graphs of output obtained from simulation.
- avg_output file contains the output as lambda, expected no., theoritical expected no., expected time, theoritical expected time, blocking probability, theoritical blocking probability, utilization of the system, theoritical utilization of the system.
- q_simulator executable to be run.
- run_script shell script file to build and run the simulation. Use following method to run the simulation.
  - modify input.in as per above instructions
  - ./run_script.sh

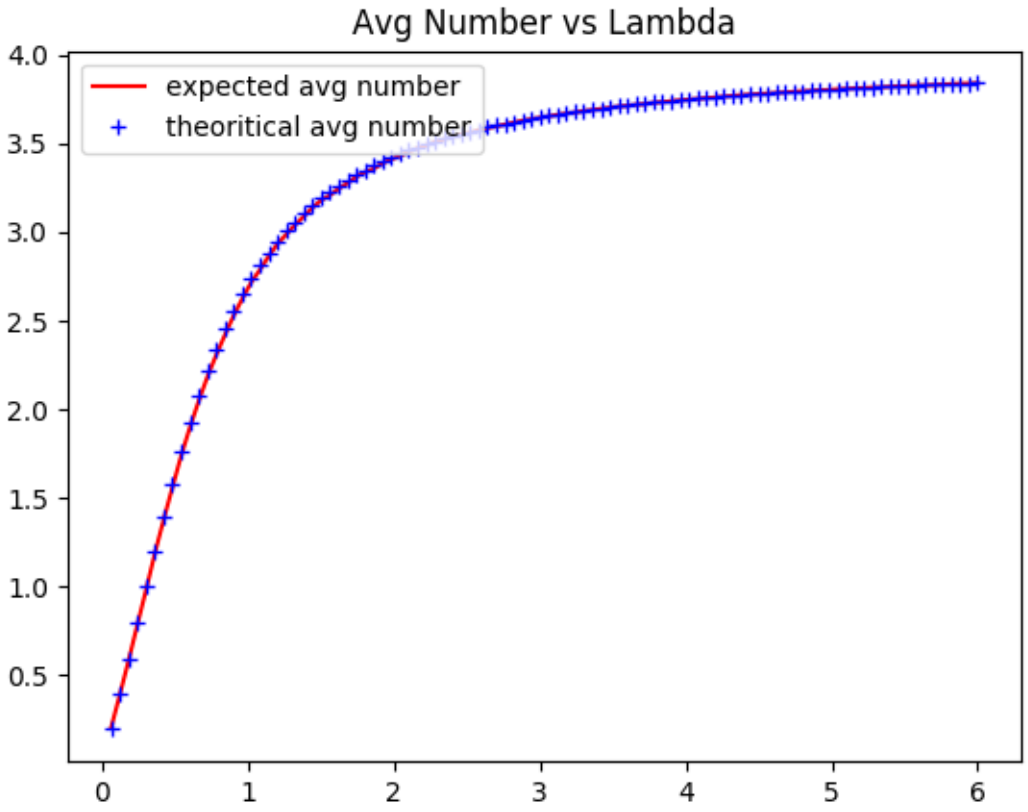
6 Dependencies

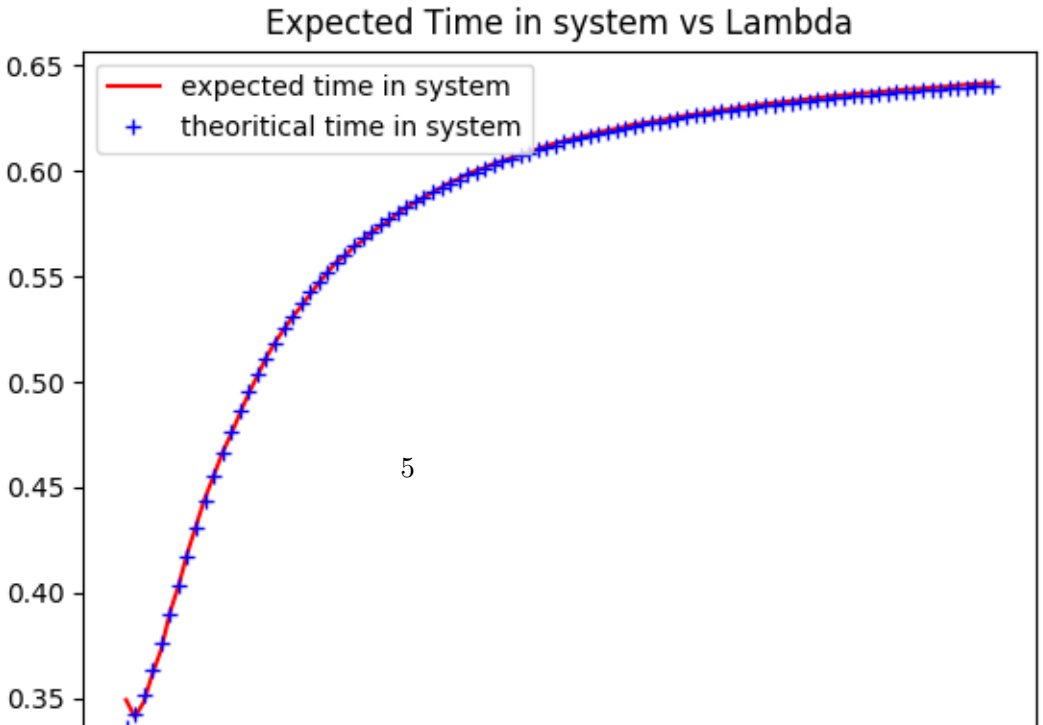
- gcc compiler
- python3. For python2 or 2.7, run_script needs to be modified to enable python2.

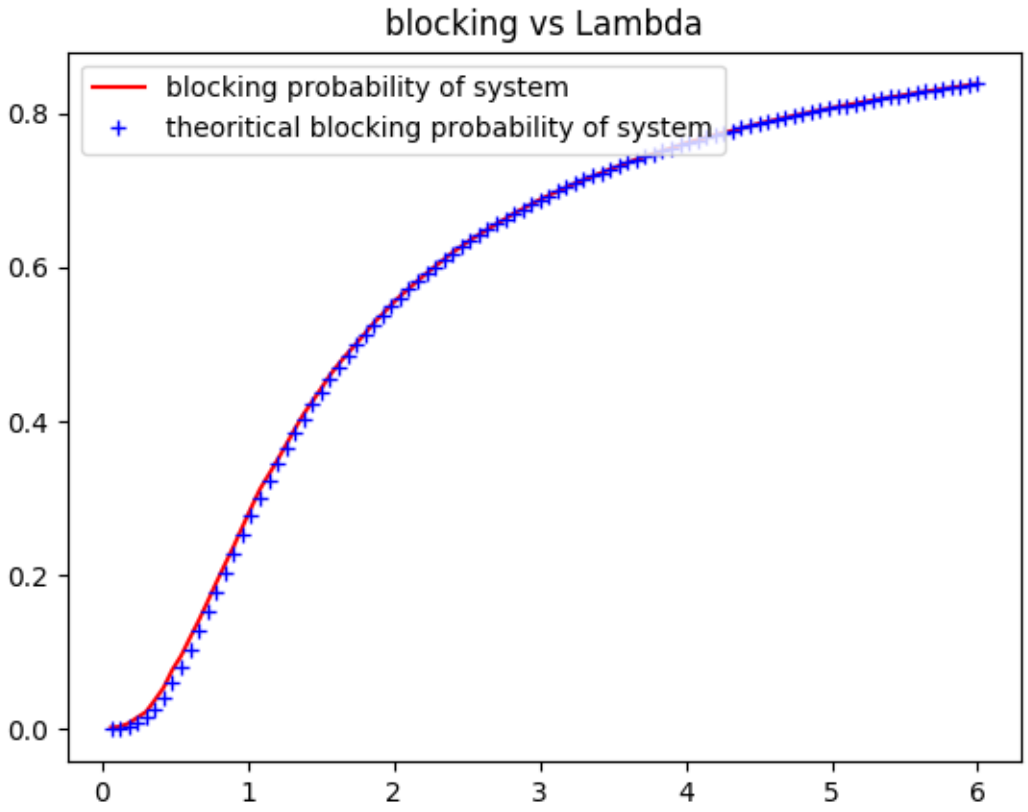
7 OUTPUTS

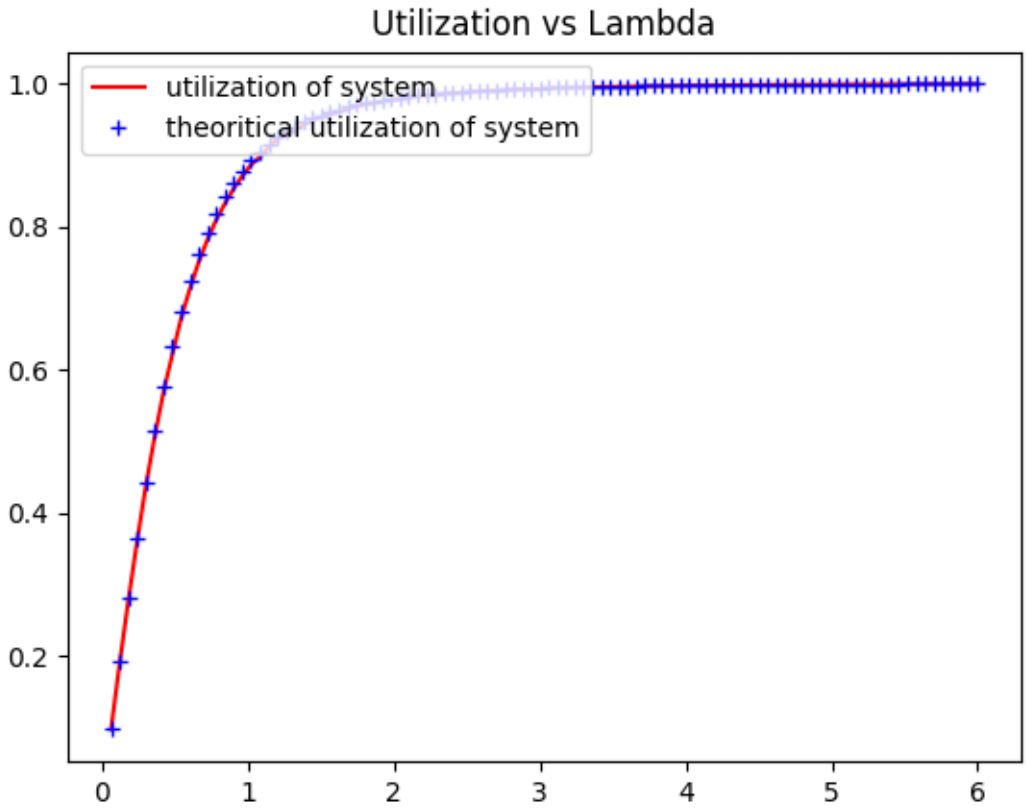
Performance measures obtained using simulation are compared to measures calculated theoritically in code. Following are the list of experiments done.

7.1 L10, K4, M2, U3

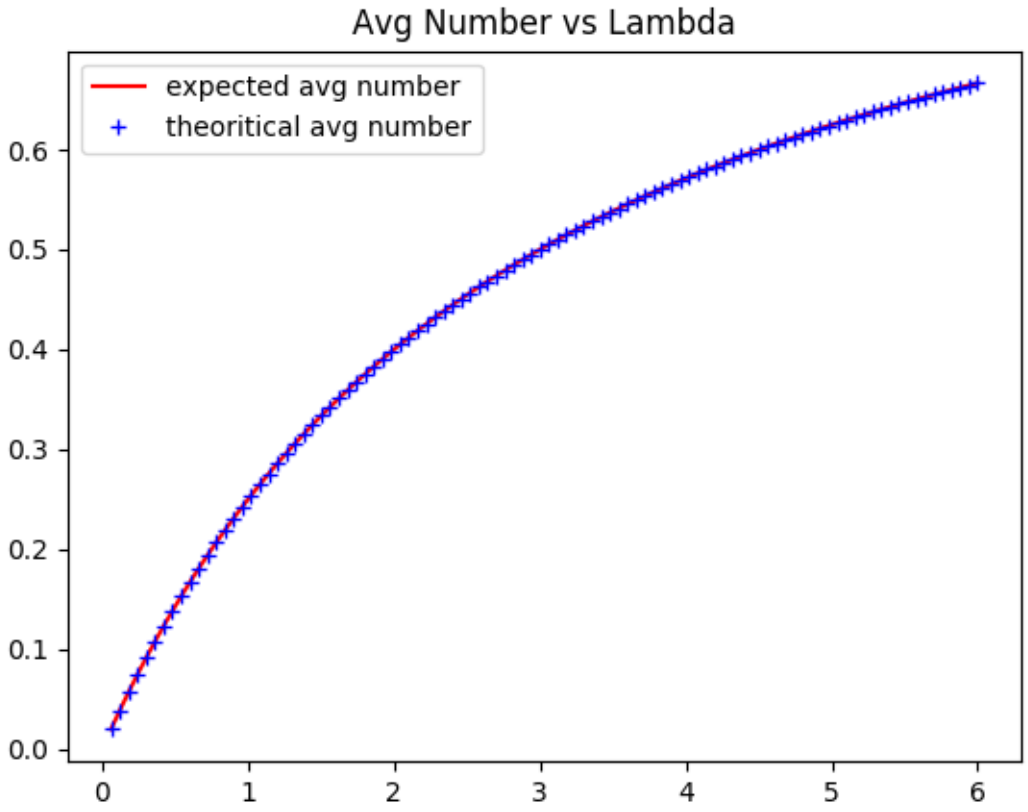


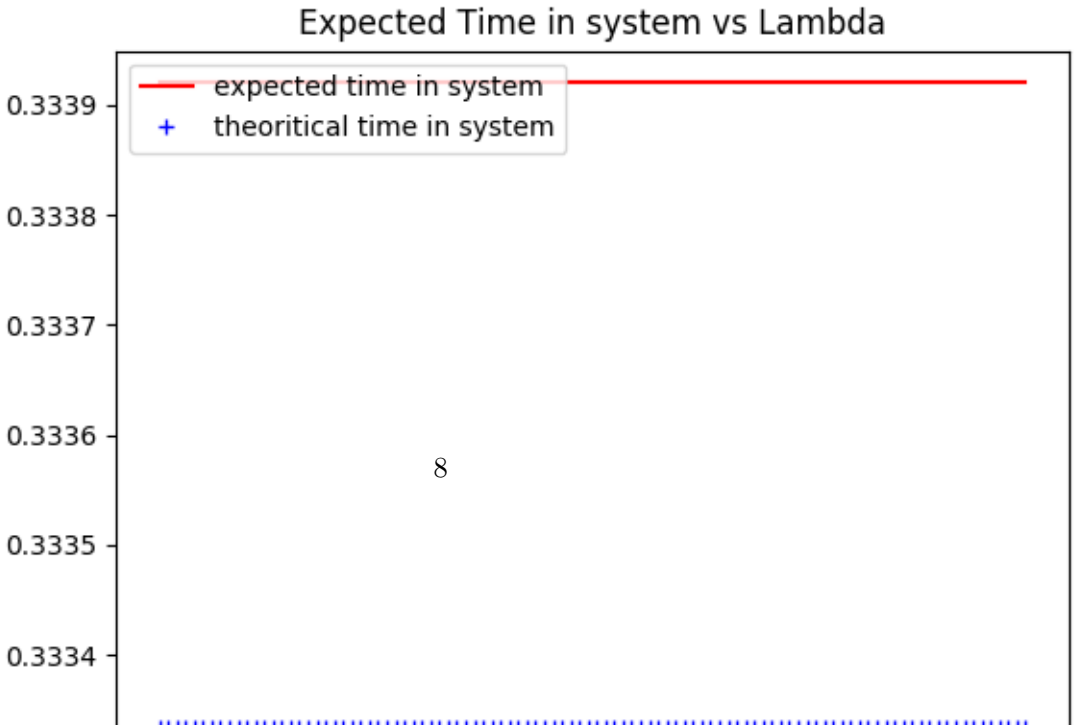


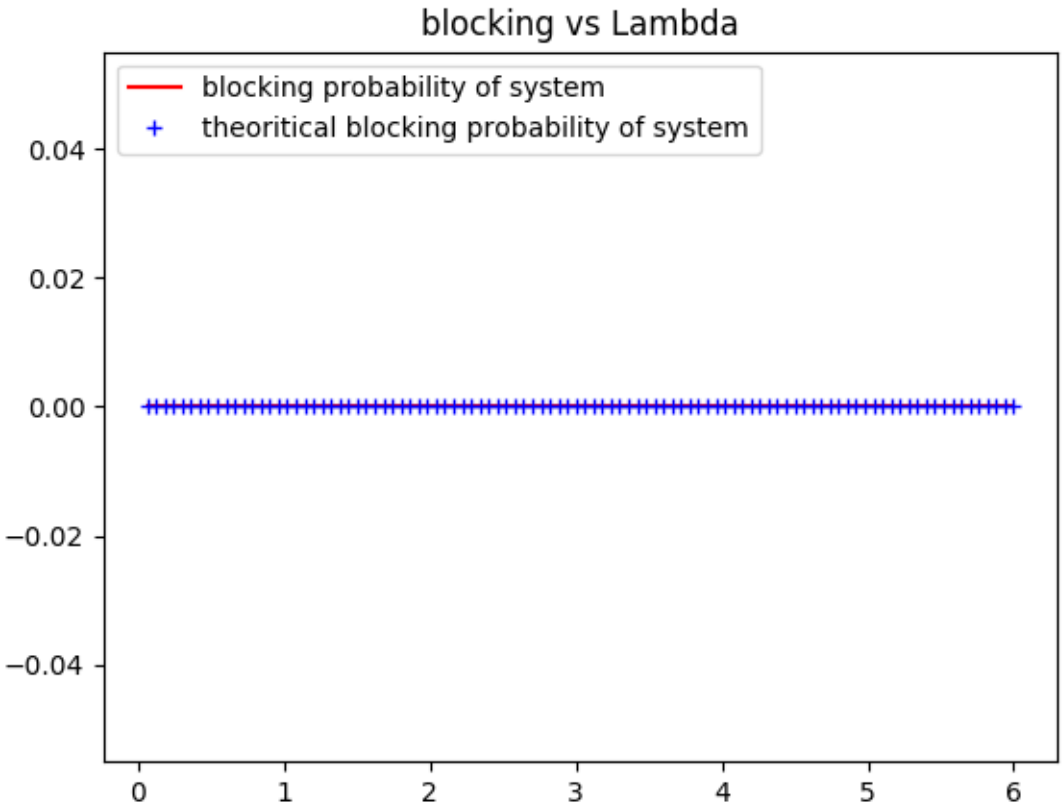

blocking vs Lambda
blocking probability of system
theoritical blocking probability of system
0.0
0.2
0.4
0.6
0.8
0
1
2
3
4
5
6

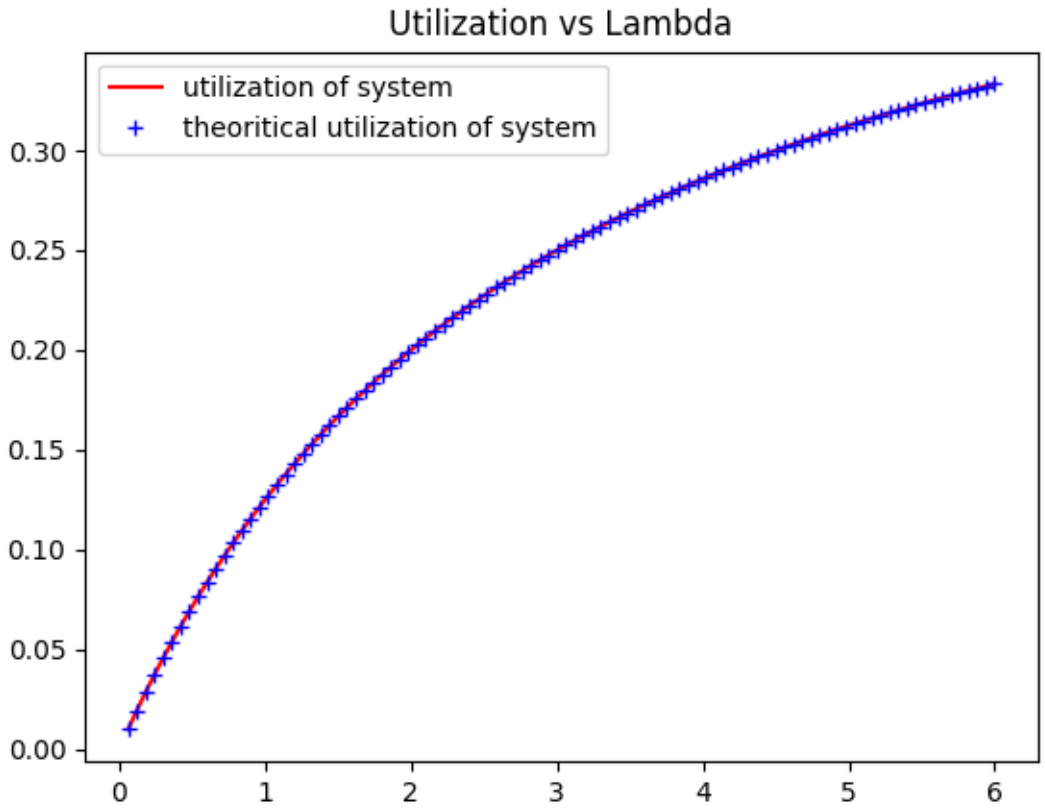

Utilization vs Lambda
utilization of system
theoritical utilization of system
1.0
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6

7.2 L1, K4, M2, U3

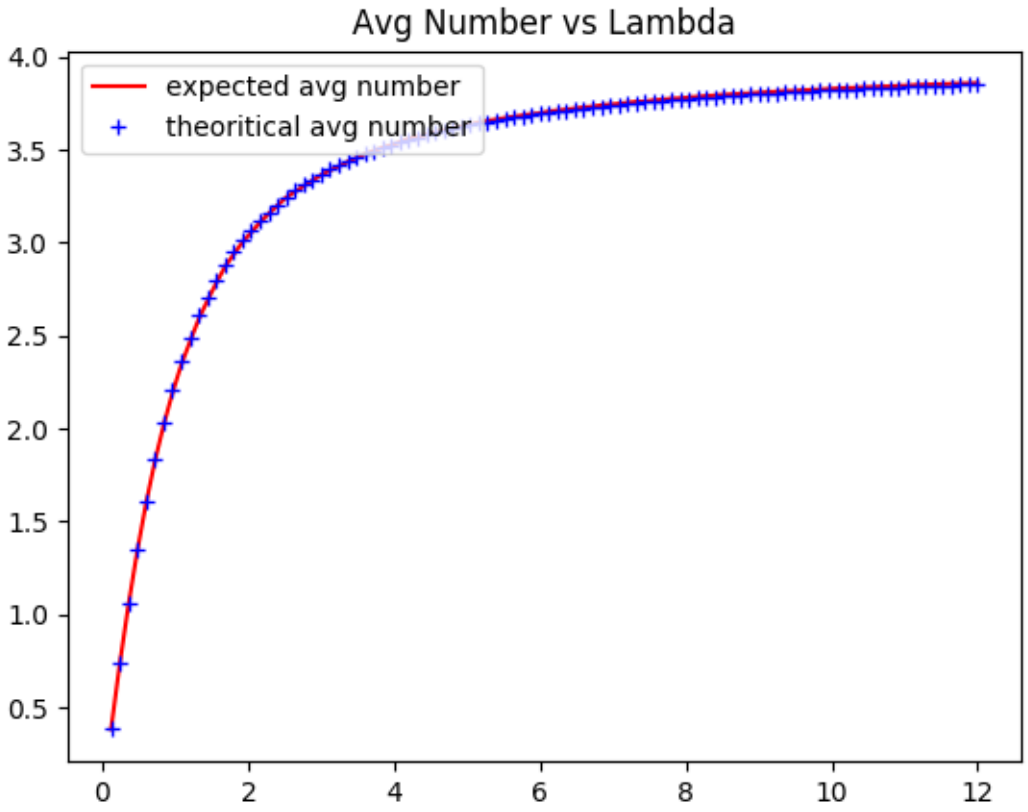

Avg Number vs Lambda
expected avg number
theoritical avg number
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
1
2
3
4
5
6

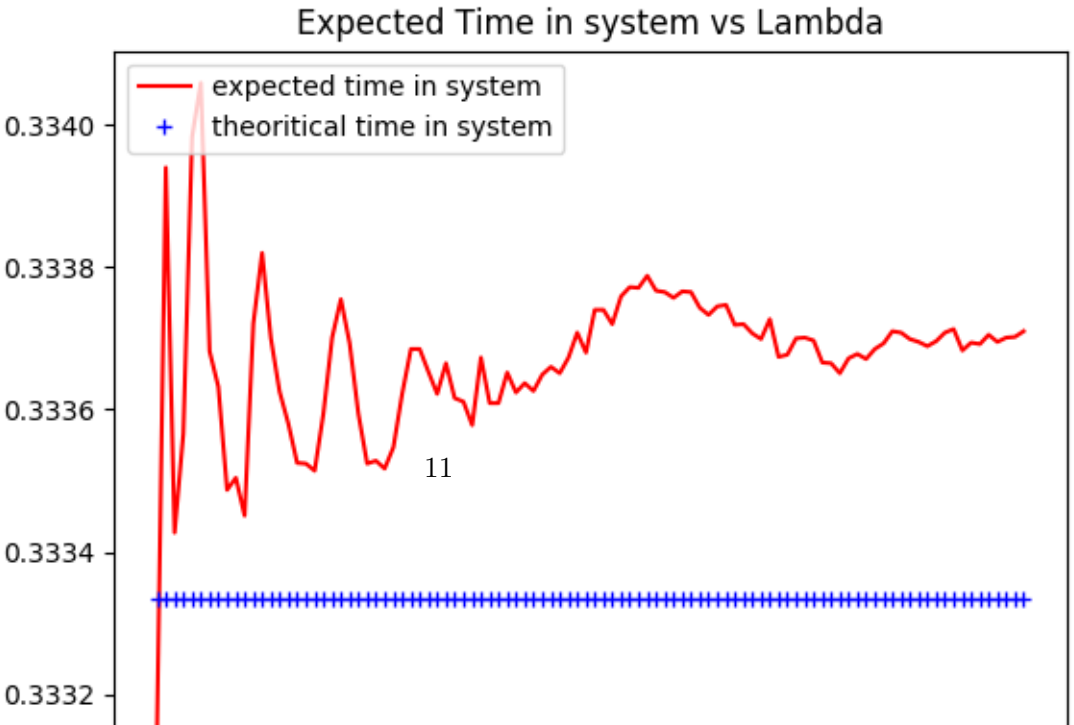

Expected Time in system vs Lambda
expected time in system
theoritical time in system
0.3339
0.3338
0.3337
0.3336
0.3335
0.3334

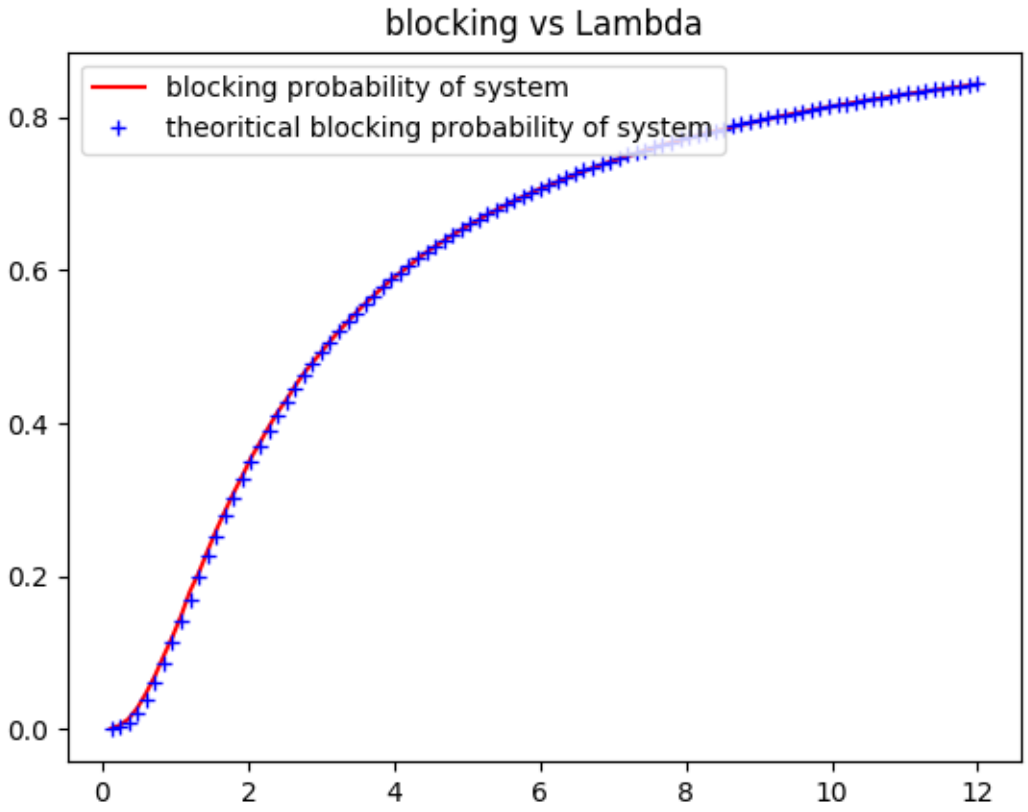

blocking vs Lambda
blocking probability of system
theoritical blocking probability of system
0.04
0.02
0.00
−0.02
−0.04
0
1
2
3
4
5
6

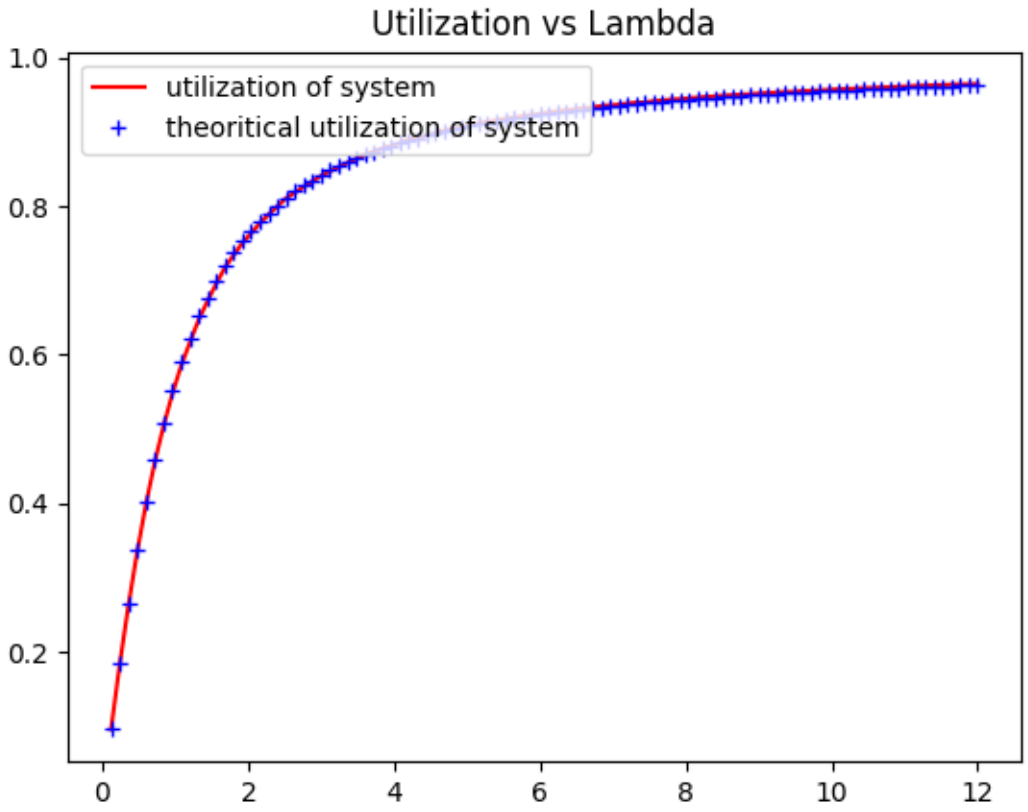

Utilization vs Lambda
utilization of system
theoritical utilization of system
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6

7.3 L10, K4, M4, U3

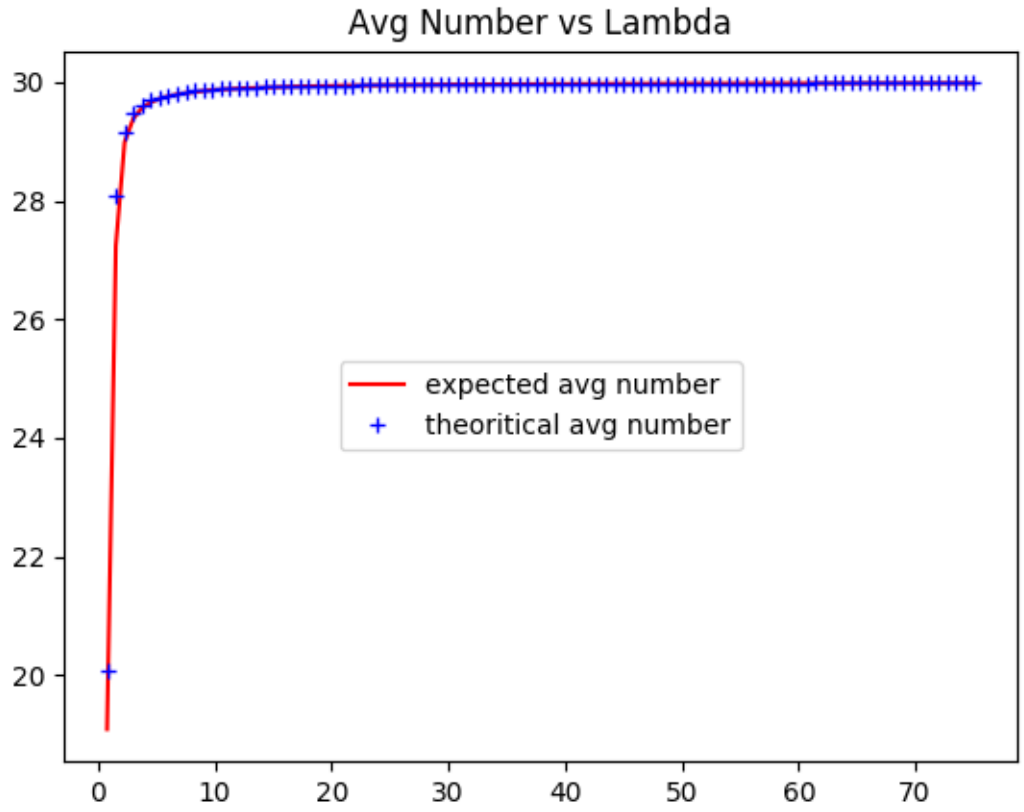


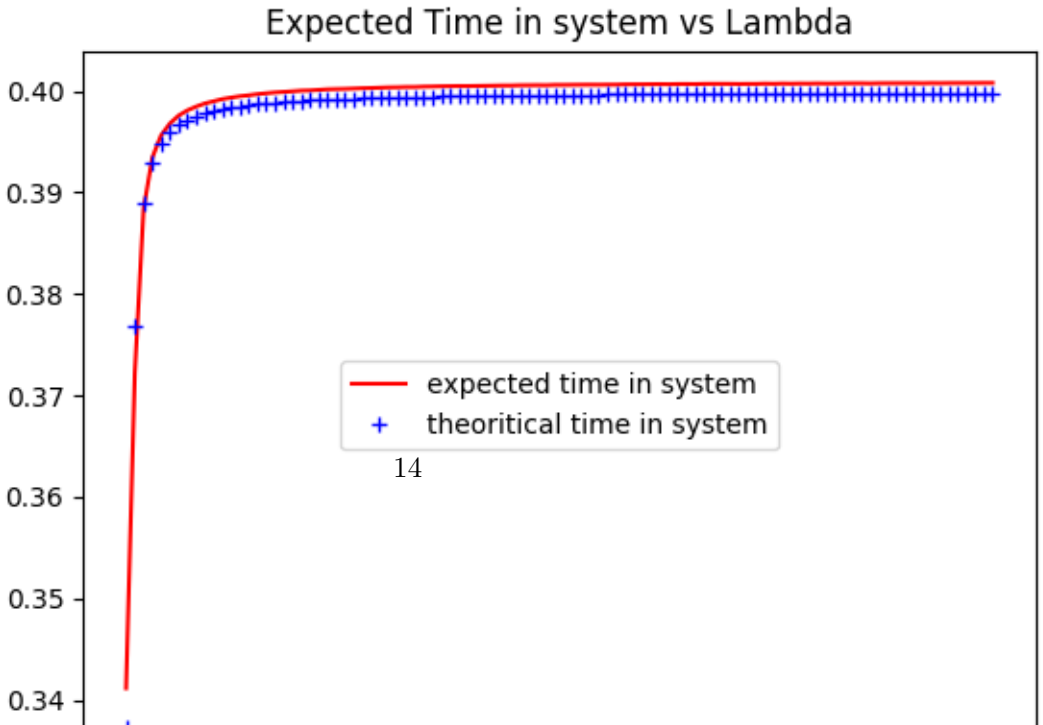


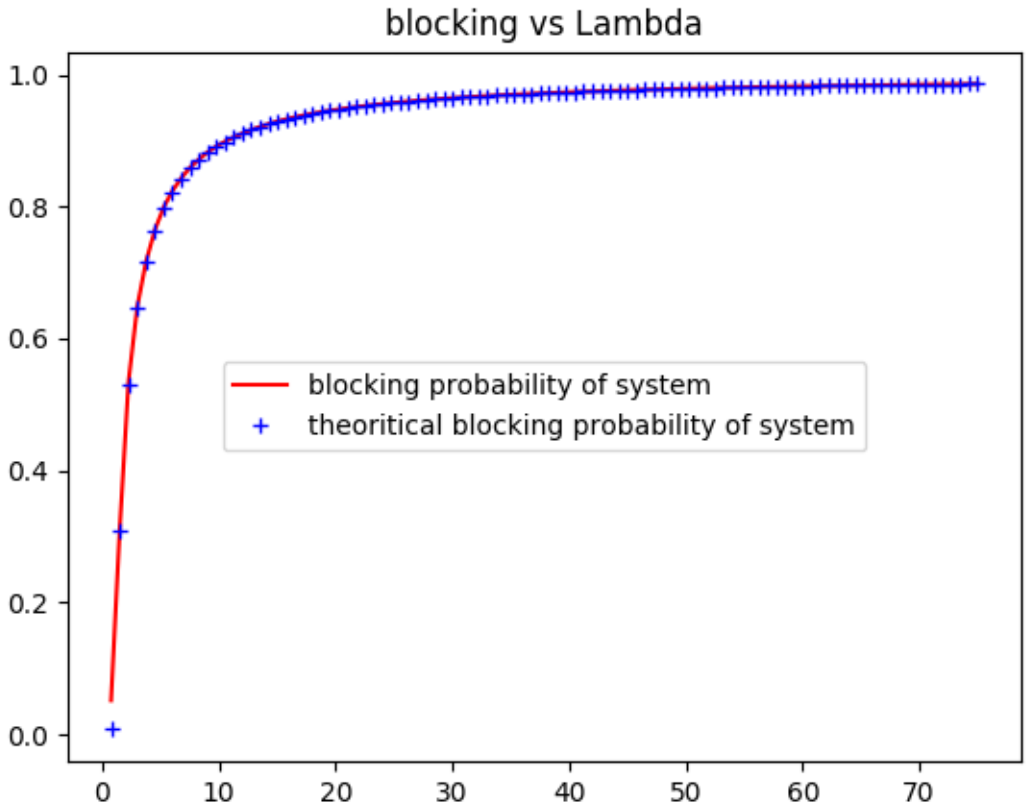

blocking vs Lambda
blocking probability of system
theoritical blocking probability of system
0.0
0.2
0.4
0.6
0.8
0
2
4
6
8
10
12

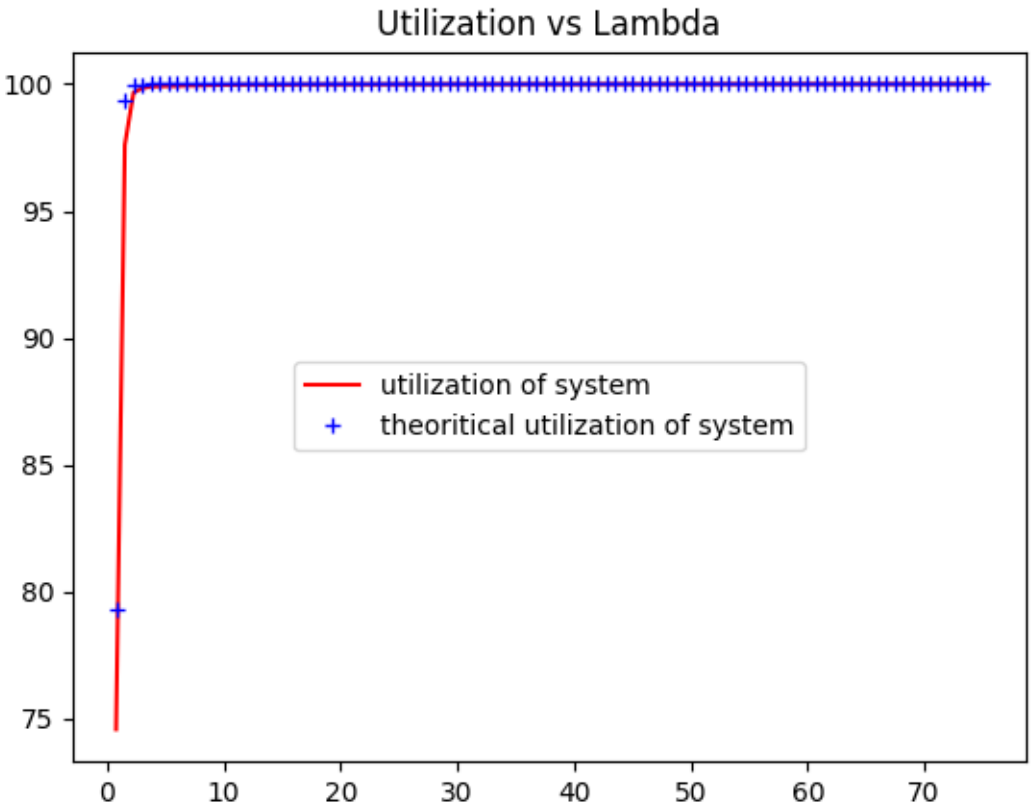

Utilization vs Lambda
utilization of system
theoritical utilization of system
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8
10
12

7.4 L100, K30, M25, U3

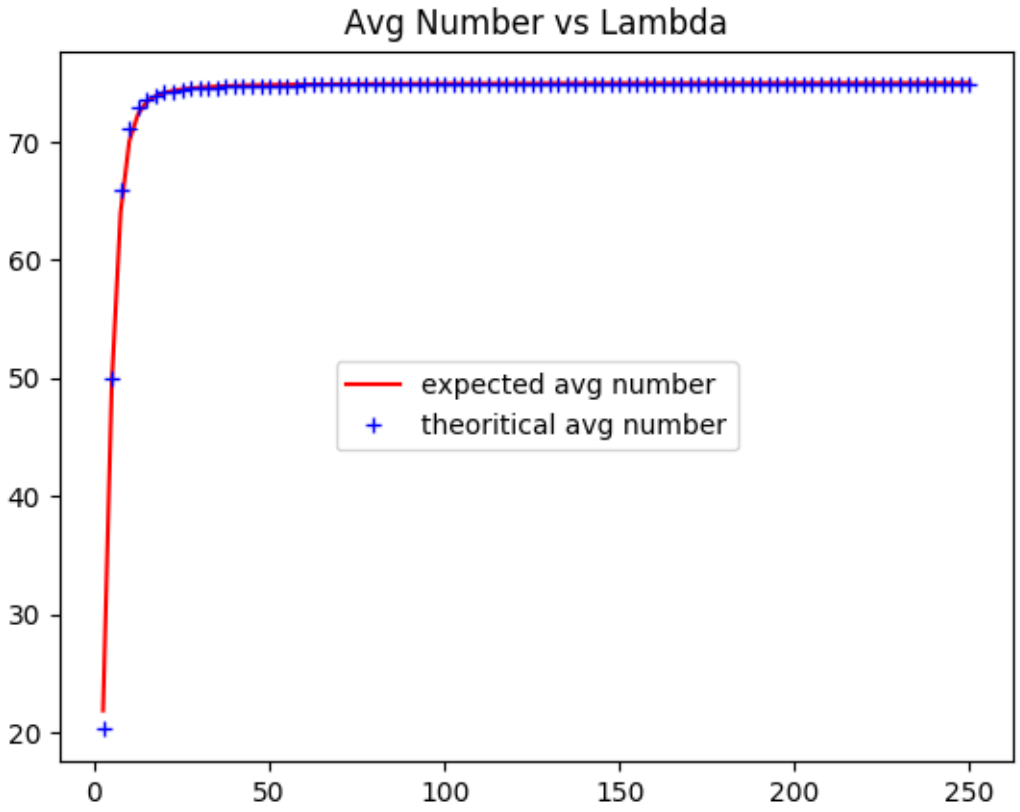


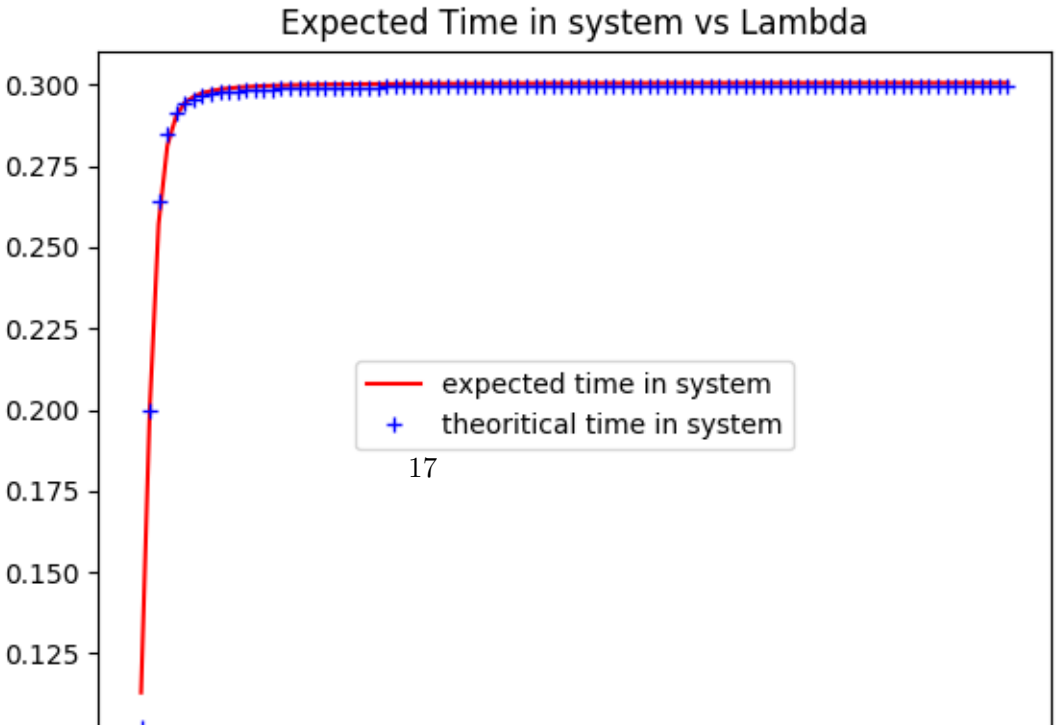


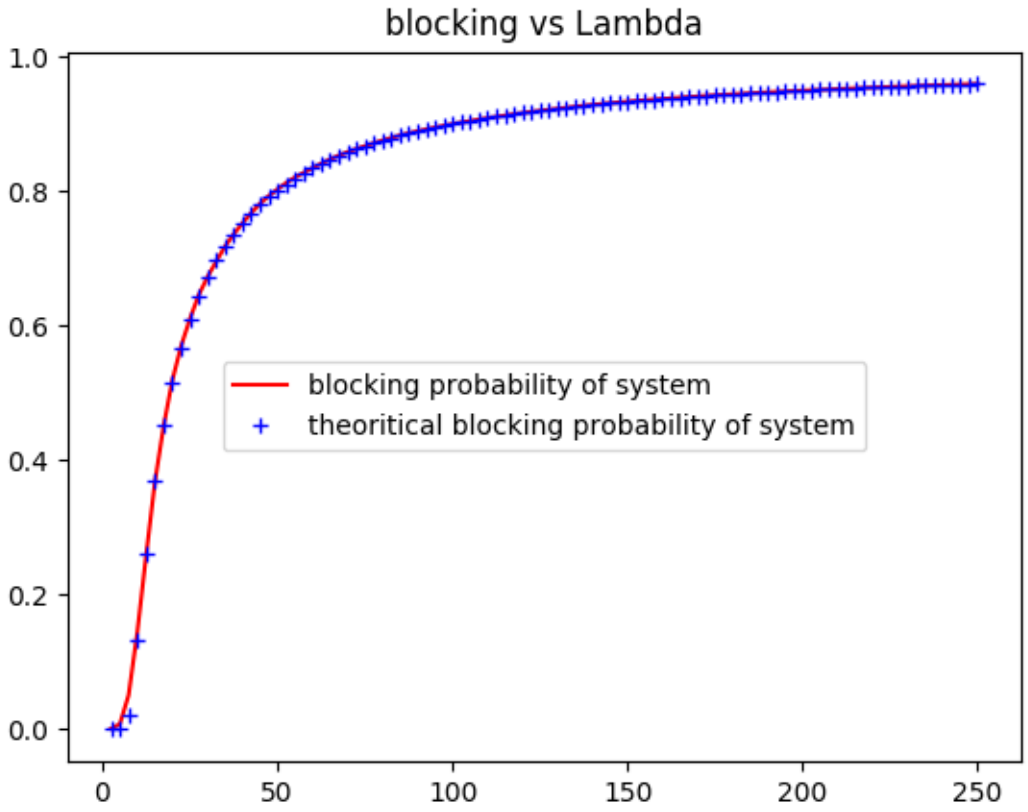

blocking vs Lambda
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70
blocking probability of system
theoritical blocking probability of system

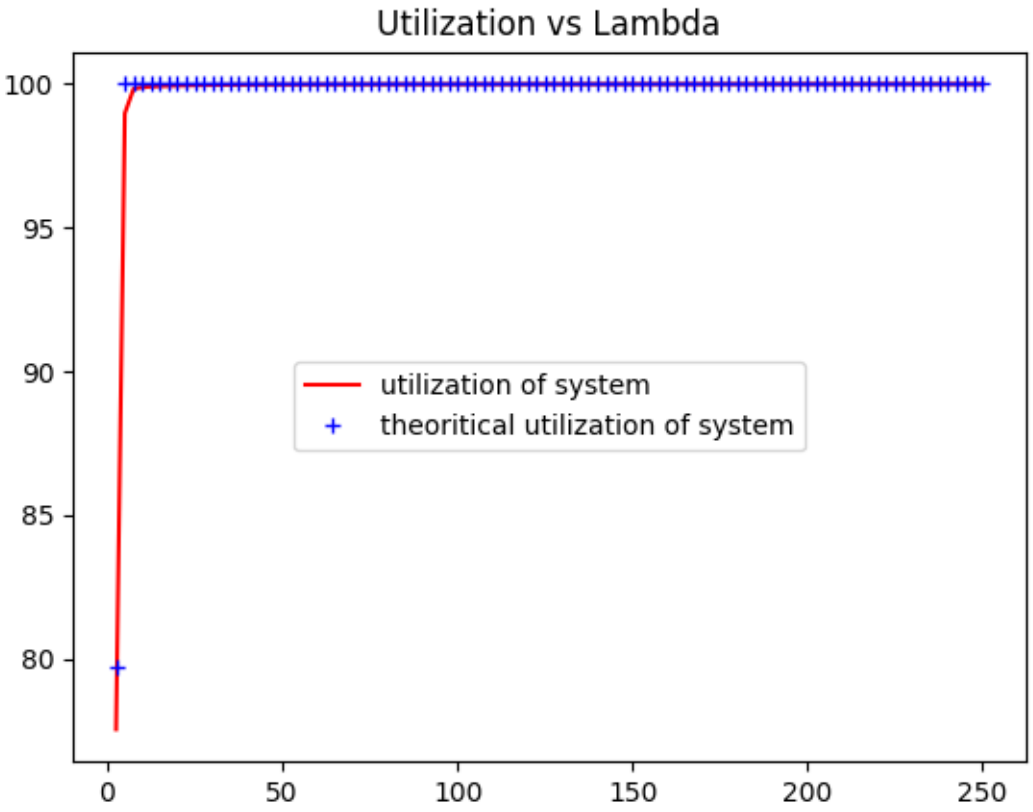

Utilization vs Lambda
100
95
90
85
80
75
0
10
20
30
40
50
60
70
utilization of system
theoritical utilization of system

7.5 L100, K75, M25, U10

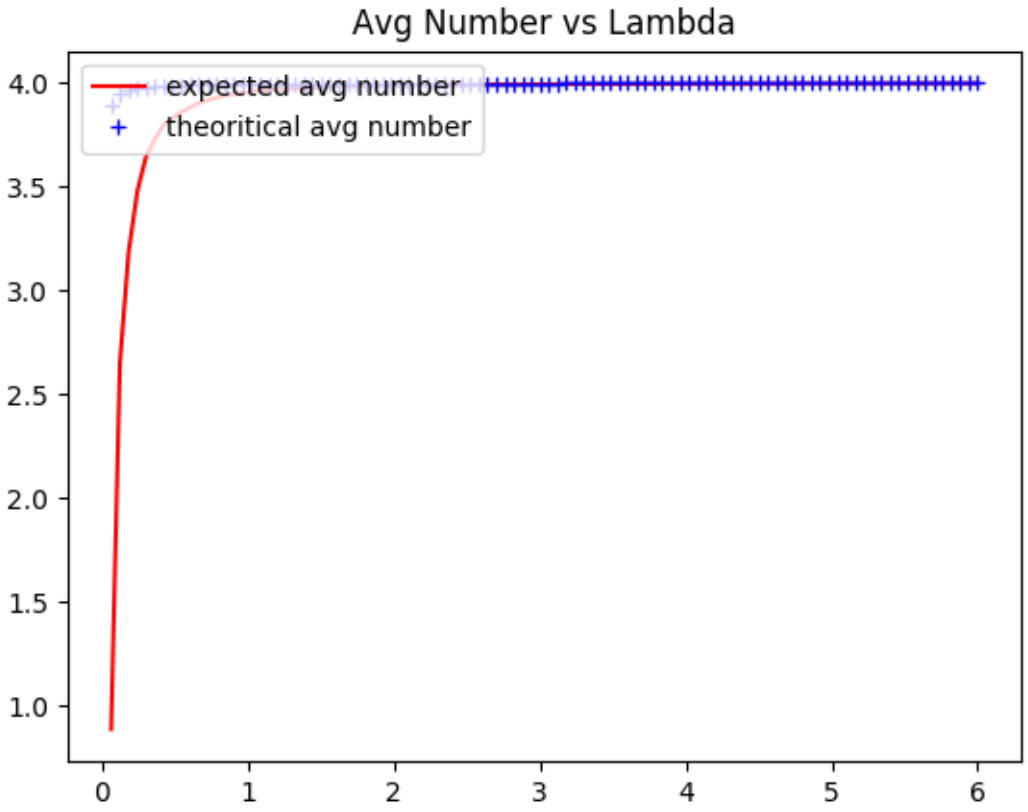


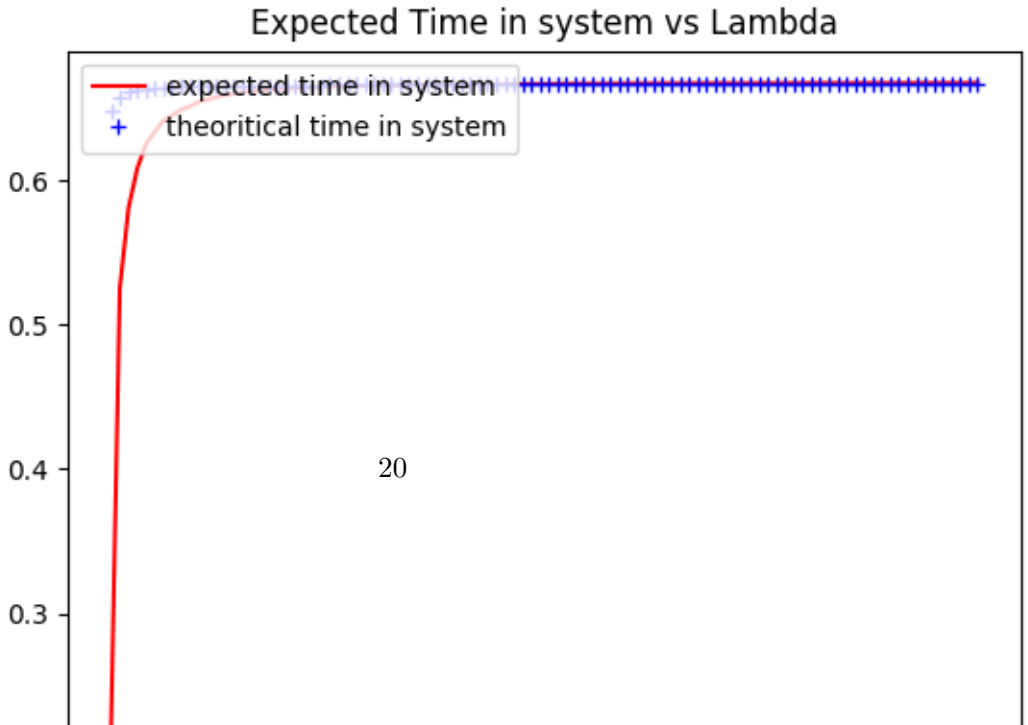


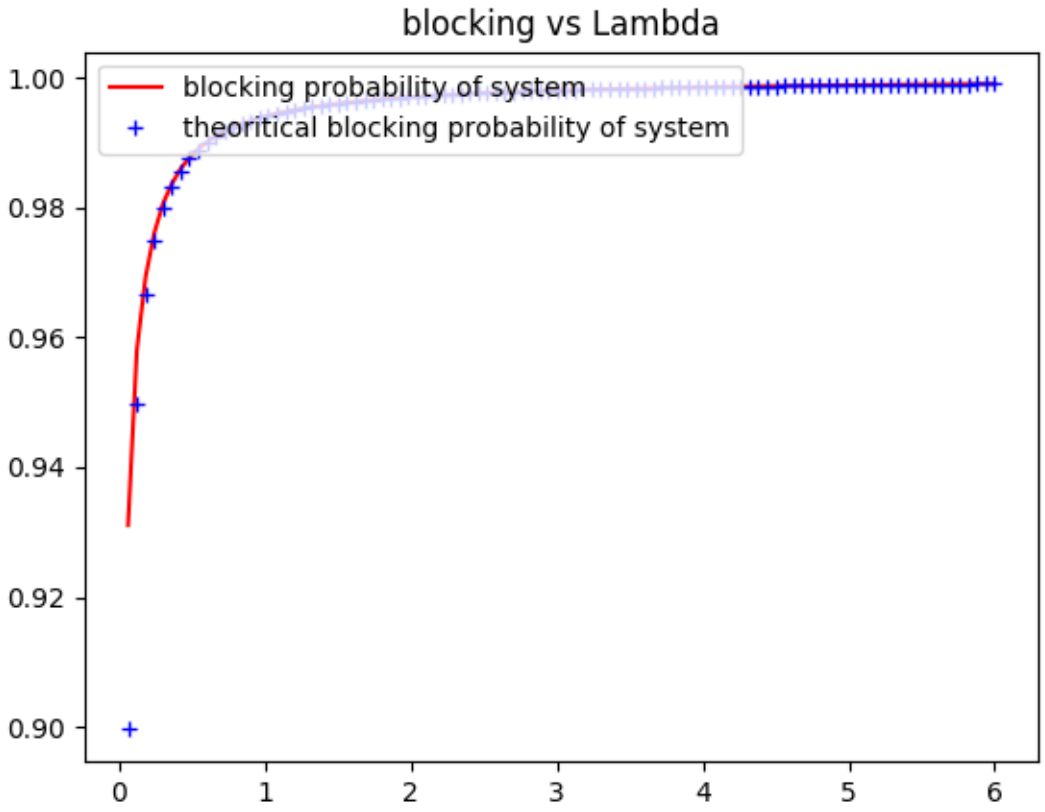

blocking vs Lambda
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
blocking probability of system
theoritical blocking probability of system

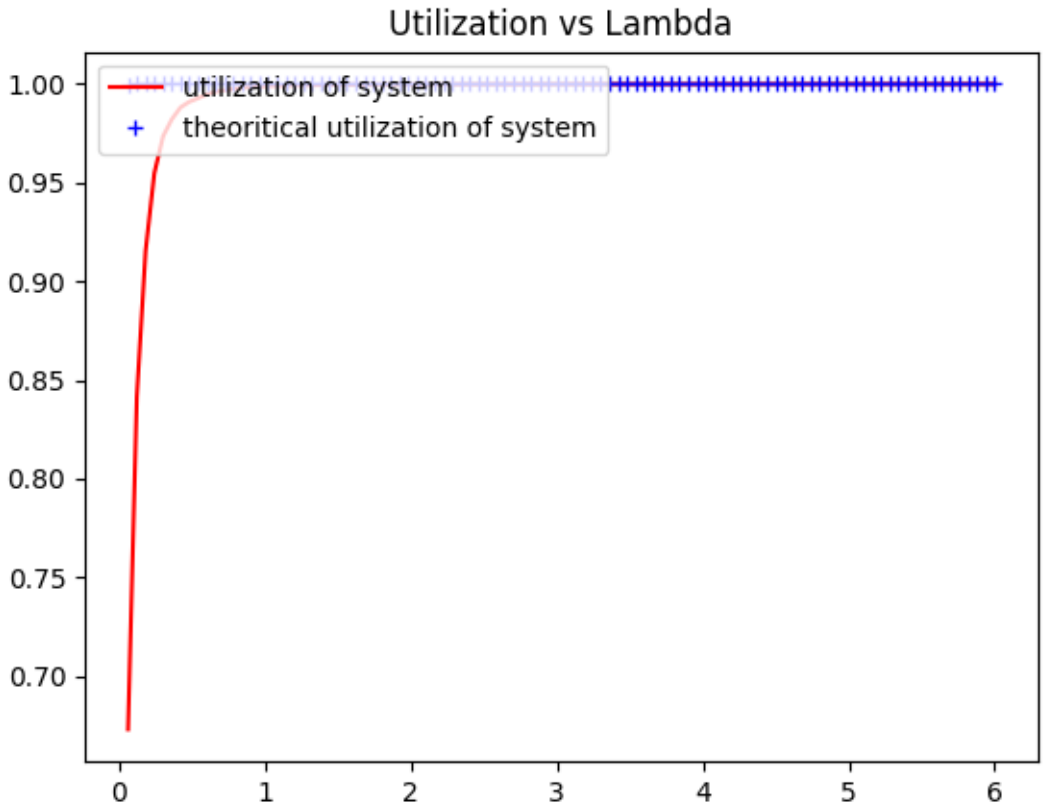

Utilization vs Lambda
100
95
90
85
80
0
50
100
150
200
250
utilization of system
theoritical utilization of system

7.6 L1000, K4, M2, U3






blocking vs Lambda
blocking probability of system
theoritical blocking probability of system
1.00
0.98
0.96
0.94
0.92
0.90
0
1
2
3
4
5
6



8 Limitations

Heap size is set to 10000 in utils.h. As long as number of events at any point of time is not more than 10000, simulator is fine. However we can modify the value and run the script again. Script builds the code and runs the simulation again.