# EEE414-DATA COMMUNICATION-LAB1

Author: Zhipeng Ye

Student ID: 1926908

Faculty: MSc Multimedia Telecommunications

Date: 12th December 2019

## Introduction

In the class, we have studied M/M/N queuing which is a theory about balancing efficiency and economy. As we all know, the more number of servers become, the more efficient the system will be. However the cost of queuing system will raise with the increase of number of servers. Therefore, scientists and engineers set up M/M/N queuing theory to research and draw up solution of this domain.

Firstly, the first M represents that the arrival rate of consumer tends to Markov process (memoryless) and the second M means that service time submits to Markov process (memoryless). Furthermore, N represents the number of service.

Suppose $N$ means steady-state number of customers, $\lambda$ means steady-state arrival rate, $T$ means steady-state customer delay.

Then $N = \lambda \cdot T$ .

The possibility of the number of costumer is $n$ in given time $t$. tends to poisson process. Here is the reason.

$$\begin{aligned} P_0(t, t+\Delta t) &= 1 - \lambda \cdot \Delta t + o(\Delta t) \\ P_1(t, t+\Delta t) &= \lambda \cdot \Delta t + o(\Delta t) \\ P_{n\geq 2}(t, t+\Delta t) &= o(\Delta t) \end{aligned}$$

We can use these formula to demonstrate that $P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$. Then the interval time submits to exponential distribution. This is because that $P\{no\ costumber\ arrived\ at\ [0,t)\} = P_0(t) = 1 - e^{-\lambda t}$. The function of possibility density is $f(t) = \lambda e^{-\lambda t}, t > 0$.

Thus, the number of costumers who leaved tend to possian process and the interval time of costumer leaving tend to the exponential distribution.

For M/M/1 system, the possibility of numbers of costumers are at system, $p_n = p_0 \cdot \rho^n,\ \rho = \frac{\lambda}{u}, p < 1$. Moreover, $p_0 = 1 - \rho, p_n = (1 - \rho) \cdot p^n$.

Then, the M/M/1 has below properties.

# M/M/1 Queueing System: Main Results

- Average number of customers in the system

$$\overline{N} = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}$$

- Average delay

$$\overline{T} = \frac{\overline{N}}{\lambda} = \frac{1}{\mu - \lambda}$$

- Average waiting time in the queue

$$\overline{W} = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = \frac{\rho}{\mu - \lambda}$$

- Average number of customers in the queue

$$\overline{N_Q} = \lambda \overline{W} = \frac{\rho^2}{1-\rho}$$

And the average delay will submit this curve.

# M/M/1 Queueing System: Main Results

- Average number of customers in the system

$$\overline{N} = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda}$$

- Average delay

$$\overline{T} = \frac{\overline{N}}{\lambda} = \frac{1}{\mu - \lambda}$$

- Average waiting time in the queue

$$\overline{W} = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = \frac{\rho}{\mu - \lambda}$$

- Average number of customers in the queue

$$\overline{N_Q} = \lambda \overline{W} = \frac{\rho^2}{1-\rho}$$

The M/M/N has below properties

- Because $\sum_{n=0}^{\infty} p_n = 1$, we obtain

$$p_0 = \frac{1}{\sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!} + \frac{(m\rho)^m}{m!} \frac{1}{1-\rho}}$$

- We first obtain $p_{m+}$, which will be used in deriving main results:

$$p_{m+} \triangleq P\{N \geq m\} = \sum_{n=m}^{\infty} p_n = \frac{(m\rho)^m}{m!(1-\rho)} p_0$$

- Average delay

$$\overline{T} = \overline{W} + \frac{1}{\mu} = \frac{\rho}{\lambda(1-\rho)} p_{m+} + \frac{1}{\mu}$$

## Simulation

Simulate 3 case , as mentioned in the given task instruction.

Our experimental value may submit to these formula.

### M/M/1

#### Code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##
# @file     new_mmN.py
# @author   Kyeong Soo (Joseph) Kim <kyeongsoo.kim@gmail.com>
# @date     2019-10-29
#
# @brief    Simulate M/M/1 queueing system
#
# @remarks  Copyright (C) 2019 Kyeong Soo (Joseph) Kim. All rights reserved.
#
# @remarks  This software is written and distributed under the GNU General
#           Public License Version 2 (http://www.gnu.org/licenses/gpl-
2.0.html).
#           You must not remove this notice, or any other, from this
software.
#

import argparse
import random

import numpy as np
import simpy


def source(env, mean_ia_time, mean_srv_time, server, delays, number, trace):
    """Generates packets with exponential interarrival time."""
    for i in range(number):
        ia_time = random.expovariate(1.0 / mean_ia_time)
        srv_time = random.expovariate(1.0 / mean_srv_time)
        pkt = packet(env, 'Packet-%d' % i, server, srv_time, delays, trace)
        env.process(pkt)
        yield env.timeout(ia_time)


def packet(env, name, server, service_time, delays, trace):
    """Requests a server, is served for a given service_time, and leaves the
server."""
    arrv_time = env.now
    if trace:
        print("t={0:.4E}s: {1:s} arrived".format(arrv_time, name))
```

```
    with server.request() as request:
        yield request
        yield env.timeout(service_time)
        delay = env.now - arrv_time
        delays.append(delay)
        if trace:
            print("t={0:.4E}s: {1:s} served for {2:.4E}s".format(env.now,
name, service_time))
            print("t={0:.4E}s: {1:s} delayed for {2:.4E}s".format(env.now,
name, delay))


def run_simulation(num_servers, mean_ia_time, mean_srv_time,
num_packets=1000, random_seed=1234, trace=True):
    """Runs a simulation and returns statistics."""
    if trace:
        print('M/M/' + str(num_servers) + ' queue\n')
    random.seed(random_seed)
    env = simpy.Environment()

    # start processes and run
    server = simpy.Resource(env, capacity=num_servers)
    delays = []
    env.process(source(env, mean_ia_time,
                       mean_srv_time, server, delays, number=num_packets,
trace=trace))
    env.run()

    # return mean delay
    return np.mean(delays)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-M",
        "--num_servers",
        help="number of servers; default is 1",
        default=1,
        type=int)  # for extension to M/M/m
    parser.add_argument(
        "-A",
        "--arrival_rate",
        help="packet arrival rate [packets/s]; default is 1.0",
        default=1.0,
        type=float)
    parser.add_argument(
        "-S",
        "--service_rate",
        help="packet service rate [packets/s]; default is 10.0",
        default=0.1,
        type=float)
    parser.add_argument(
        "-N",
        "--num_packets",
```

```
        help="number of packets to generate; default is 1000",
        default=1000,
        type=int)
    parser.add_argument(
        "-R",
        "--random_seed",
        help="seed for random number generation; default is 1234",
        default=1234,
        type=int)
    parser.add_argument('--trace', dest='trace', action='store_true')
    parser.add_argument('--no-trace', dest='trace', action='store_false')
    parser.set_defaults(trace=True)
    args = parser.parse_args()

    # set variables using command-line arguments
    num_servers = args.num_servers  # for extension to M/M/m
    mean_ia_time = 1.0 / args.arrival_rate
    mean_srv_time = 1.0 / args.service_rate
    num_packets = args.num_packets
    random_seed = args.random_seed
    trace = args.trace

    # run a simulation
    mean_delay = run_simulation(num_servers, mean_ia_time, mean_srv_time,
                                num_packets, random_seed,
                                trace)

    # print arrival rate and mean delay
    print("{0:.4E}\t{1:.4E}".format(args.arrival_rate, mean_delay))
```

Here is my command line code, because I use Mac OS, I choose shell code to run simulation batchly.

```
for i in $(seq 5 5 95);
do echo $i ;
python new_mmN.py -M 1 -A $i -S 100 -N 10000 --no-trace >> mm1.out
done
```

Result

```
5.0000E+00  1.0511E-02
1.0000E+01  1.1095E-02
1.5000E+01  1.1725E-02
2.0000E+01  1.2452E-02
2.5000E+01  1.3263E-02
3.0000E+01  1.4141E-02
3.5000E+01  1.5148E-02
4.0000E+01  1.6342E-02
4.5000E+01  1.7738E-02
5.0000E+01  1.9411E-02
5.5000E+01  2.1465E-02
6.0000E+01  2.4243E-02
6.5000E+01  2.7676E-02
7.0000E+01  3.2535E-02
```

```
7.5000E+01  3.9691E-02
8.0000E+01  4.8997E-02
8.5000E+01  6.4200E-02
9.0000E+01  9.7699E-02
9.5000E+01  2.0845E-01
```

Plot diagram

```
import argparse

import numpy as np
import matplotlib.pyplot as plt
import math

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-M",
        "--num_servers",
        help="number of servers; default is 1",
        default=1,
        type=int)
    parser.add_argument(
        "-F",
        "--file_path",
        help="file_path",
        default='',
        type=str)
    parser.add_argument(
        "-A",
        "--arrival_rate",
        help="packet arrival rate [packets/s]; default is 1.0",
        default=1.0,
        type=float)
    parser.add_argument(
        "-S",
        "--service_rate",
        help="packet service rate [packets/s]; default is 10.0",
        default=0.1,
        type=float)

    args = parser.parse_args()

    file_path = args.file_path
    arrival_rate = args.arrival_rate
    service_rate = args.service_rate
    num_servers = args.num_servers

    file_prefix = file_path.split('.')[0]

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)

    # Major ticks every 10, minor ticks every 5 for x axis
```

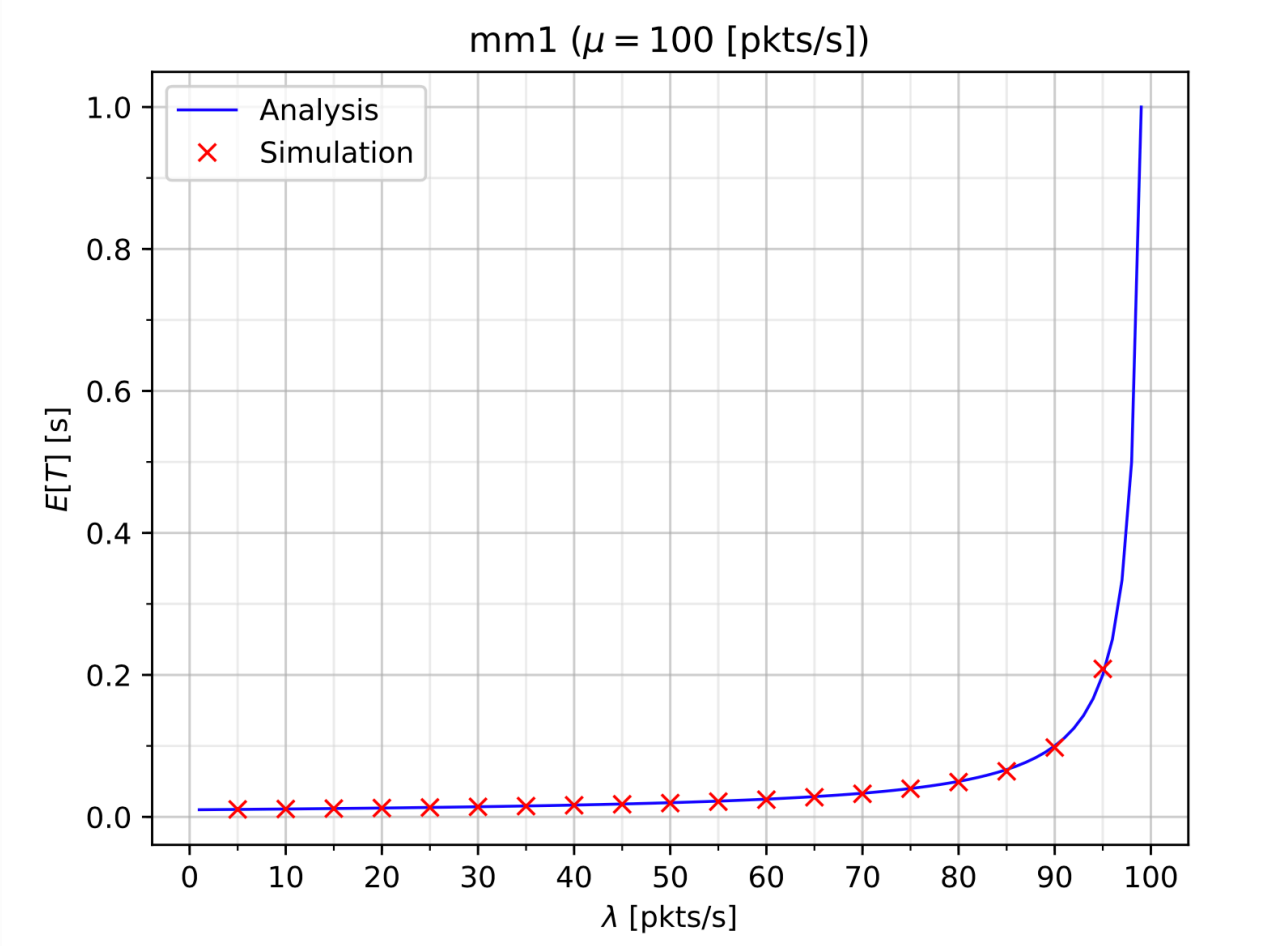
```
    x_major_ticks = np.arange(0, 101, 10)
    x_minor_ticks = np.arange(0, 101, 5)

    # Major ticks every 0.1, minor ticks every 0.1 for y axis
    y_major_ticks = np.arange(0, 1.1, 0.2)
    y_minor_ticks = np.arange(0, 1.1, 0.1)

    ax.set_xticks(x_major_ticks)
    ax.set_xticks(x_minor_ticks, minor=True)
    ax.set_yticks(y_major_ticks)
    ax.set_yticks(y_minor_ticks, minor=True)

    # Or if you want different settings for the grids:
    ax.grid(which='minor', alpha=0.2)
    ax.grid(which='major', alpha=0.5)

    # calculate and plot analytical results

    # m/m/1 formula
    if num_servers == 1:
        x1 = np.arange(1, 100)
        y1 = 1 / (100 - x1)
        plt.plot(x1, y1, 'b-', label="Analysis", linewidth=1)

    else:

        # m/m/n formula
        m = num_servers

        x1 = np.arange(1, 100)
        p = x1 / service_rate / m

        n = np.arange(0, m)

        denominator = np.array([(m * p) ** num for num in n])

        division = np.array([math.factorial(num) for num in n])

        p0 = 1 / (sum(np.array([denominator[num] / division[num] for num in
range(m)]))
                  + (m * p) ** m / math.factorial(m)
                  * 1 / (1 - p))

        pm = (m * p) ** m / (math.factorial(m) * (1 - p)) * p0

        y1 = p / (x1 * (1 - p)) * pm + 1 / service_rate

        plt.plot(x1, y1, 'b-', label="Analysis", linewidth=1)

    # load and plot simulation results
    # change delimiter '/t' into ','
    x2, y2 = np.loadtxt(file_path, delimiter='\t', unpack=True)
    plt.plot(x2, y2, 'rx', label="Simulation")

    # add labels, legend, and title
```

```
    plt.xlabel(r'$\lambda$ [pkts/s]')
    plt.ylabel(r'$E[T]$ [s]')
    plt.legend()
    plt.title(r'' + file_prefix + ' ($\mu=100$ [pkts/s])')

    plt.savefig(file_prefix + '.pdf')
    plt.show()
```

Shell code

```
python plot_mmN.py -M 1 -F mm1.out -S 100
```

Result




Analysis

As we can see from result, the simulation is consistent to theoretical curve.

$$\bar{T} = \frac{1}{u - \lambda}$$

Refer to previous process, here is my result of M/M/2 and M/M/5. Because the formula of M/M/N is different from M/M/1, I change the analysis expression in python file.

## M/M/2

Main change in python file

```
# m/m/n formula
        m = num_servers

        x1 = np.arange(1, 100)
        p = x1 / service_rate / m

        n = np.arange(0, m)

        denominator = np.array([(m * p) ** num for num in n])

        division = np.array([math.factorial(num) for num in n])

        p0 = 1 / (sum(np.array([denominator[num] / division[num] for num in
range(m)]))
                  + (m * p) ** m / math.factorial(m)
                  * 1 / (1 - p))

        pm = (m * p) ** m / (math.factorial(m) * (1 - p)) * p0

        y1 = p / (x1 * (1 - p)) * pm + 1 / service_rate
```

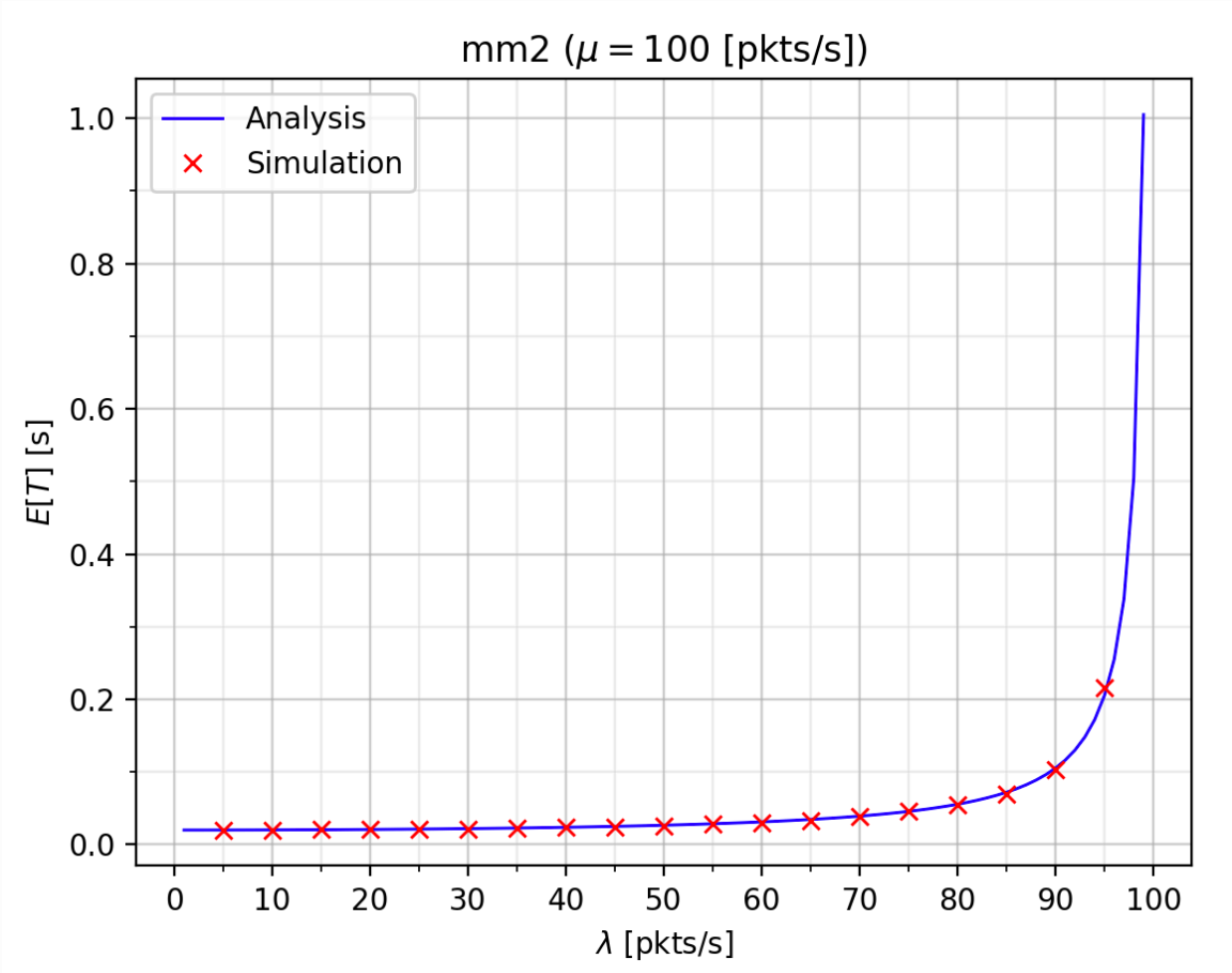
Shell code

```
for i in $(seq 5 5 95);
do echo $i ;
python new_mmN.py -M 2 -A $i -S 50 -N 10000 --no-trace >> mm2.out
done
```

```
python plot_mmN.py -M 2 -F mm2.out -S 50
```

Result

```
5.0000E+00  2.0092E-02
1.0000E+01  2.0258E-02
1.5000E+01  2.0511E-02
2.0000E+01  2.0871E-02
2.5000E+01  2.1329E-02
3.0000E+01  2.1893E-02
3.5000E+01  2.2615E-02
4.0000E+01  2.3504E-02
4.5000E+01  2.4626E-02
5.0000E+01  2.6058E-02
5.5000E+01  2.7842E-02
6.0000E+01  3.0358E-02
6.5000E+01  3.3583E-02
7.0000E+01  3.8385E-02
7.5000E+01  4.5473E-02
8.0000E+01  5.4823E-02
8.5000E+01  7.0142E-02
9.0000E+01  1.0392E-01
9.5000E+01  2.1546E-01
```




## M/M/5

### Shell code

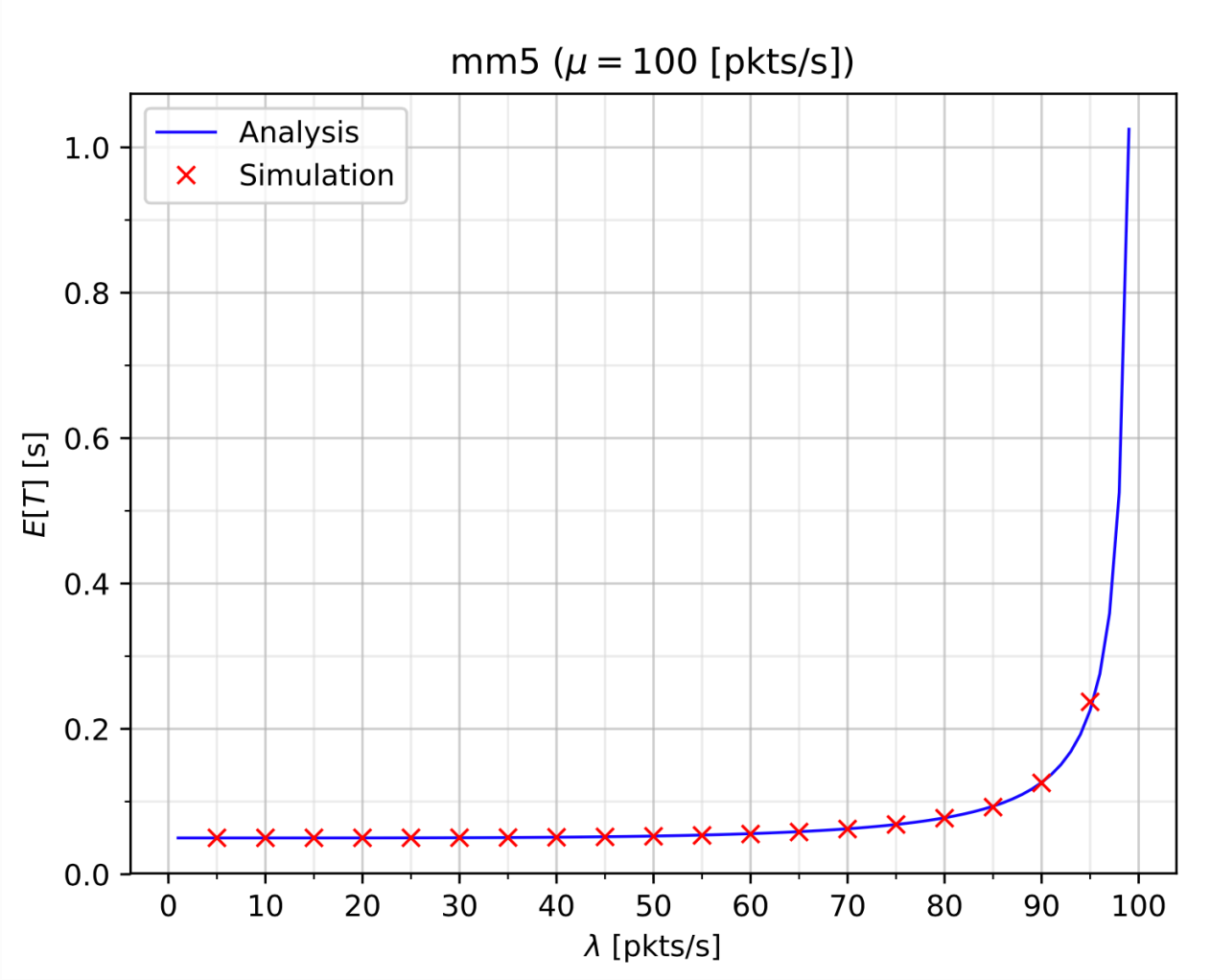
```
for i in $(seq 5 5 95);
do echo $i ;
python new_mmN.py -M 5 -A $i -S 20 -N 10000 --no-trace >> mm5.out
done
```

```
python plot_mmN.py -M 5 -F mm5.out -S 20
```

### Result

```
5.0000E+00  5.0127E-02
1.0000E+01  5.0127E-02
1.5000E+01  5.0132E-02
2.0000E+01  5.0149E-02
2.5000E+01  5.0197E-02
3.0000E+01  5.0312E-02
3.5000E+01  5.0520E-02
4.0000E+01  5.0868E-02
4.5000E+01  5.1402E-02
5.0000E+01  5.2227E-02
5.5000E+01  5.3460E-02
6.0000E+01  5.5348E-02
6.5000E+01  5.8098E-02
```

```
7.0000E+01  6.2226E-02
7.5000E+01  6.8487E-02
8.0000E+01  7.7166E-02
8.5000E+01  9.2385E-02
9.0000E+01  1.2569E-01
9.5000E+01  2.3718E-01
```




---

## Comparision and conclusion

Comparing with 3 cases the average delay will increase with the augment of number of servers.

Although this phenomenon is opposite to reality. But I found the reason for this. Here are reasons.

According to formula, $\rho = \frac{\lambda}{mu}$, $\rho$ is the same in 3 case, because of the parameters in 3 cases, $\{(1, 100), (2, 50), (5, 20)\}$.

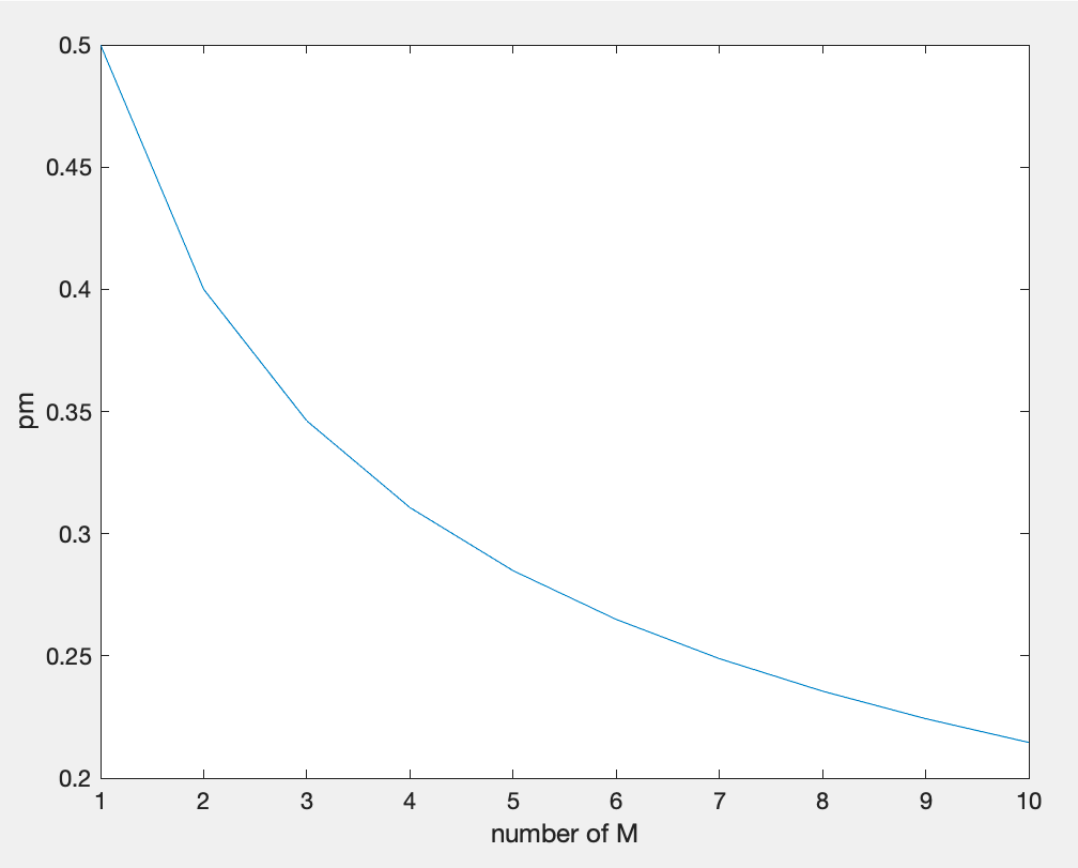
Assume $\rho$ is a constant number. With the increase of $m$, $p_0 = \frac{1}{\sum_{n=0}^{m-1} \frac{(mp)^n}{n!} + \frac{(m\rho)^m}{m!} \frac{1}{1-\rho}}$, $p_0$ will reduce.

$$p_m = \frac{(m\rho)^m}{m!(1-\rho)} p_0 = \frac{(m\rho)^m}{m!(1-\rho)} \frac{1}{\sum_{n=0}^{m-1} \frac{(mp)^n}{n!} + \frac{(m\rho)^m}{m!} \frac{1}{1-\rho}} = \frac{1}{1+\frac{\sum_{n=0}^{m-1} \frac{(mp)^n}{n!}}{\frac{(m\rho)^m}{m!} \frac{1}{1-\rho}}}.$$

I simulate this formula in Matlab to see the tendency.

```
clear
clc

m = 1:10

y_1 = m.^m./factorial(m)

y_2 = []
for k = 1:10
    y_2_value = 0
    for n = 0:k-1
        y_2_value = y_2_value + k^n/factorial(n)
    end
    y_2 = [y_2 y_2_value]
end
pm = 1./(1+y_2./y_1)

plot(m,pm)

xlabel('number of M')

ylabel('pm')
```

Result




**The result shows that $p_m$ will reduce with the increase of m**.

The **average waiting time** is the queue is

$\bar{W} = \frac{\rho}{\lambda(1-\rho)} p_m.$

**Therefore, the average waiting time will decrease with the increase of the number of servers.**

**However, for the formula of**

$\bar{T} = \frac{\rho}{\lambda(1-\rho)} p_m + \frac{1}{u}$

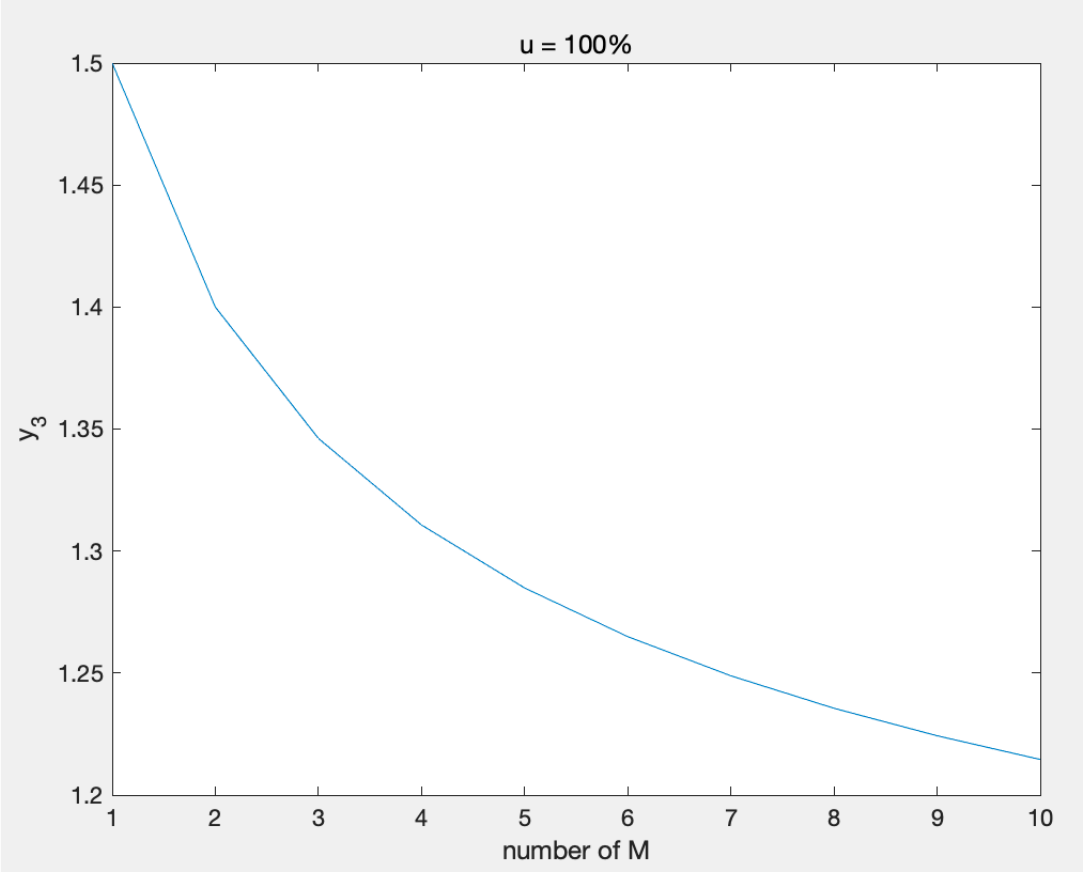
**We must consider of factor of $\frac{1}{u}$**, if we want to compare average delay in 3 cases.

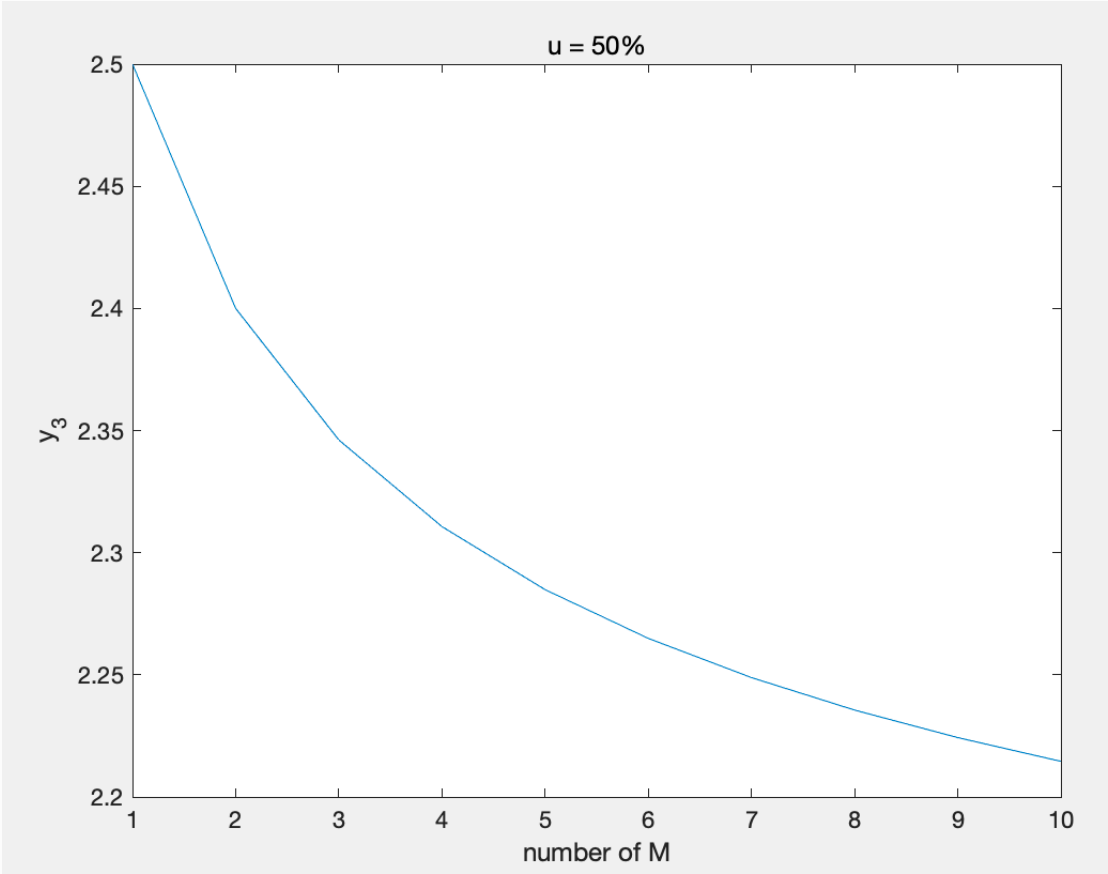
I simulate it in Matlab.

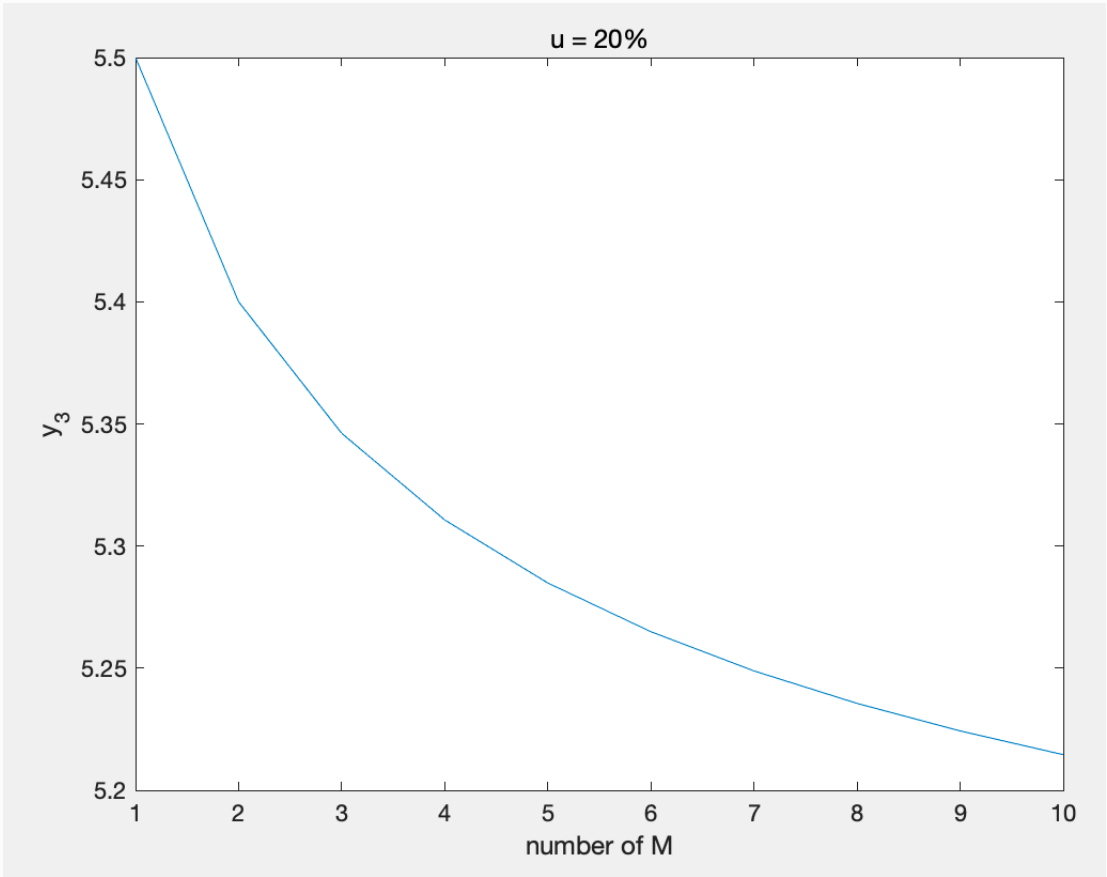
```
clear
clc

m = 1:10

y_1 = m.^m./factorial(m)

y_2 = []
for k = 1:10
    y_2_value = 0
    for n = 0:k-1
        y_2_value = y_2_value + k^n/factorial(n)
```

```
    end
    y_2 = [y_2 y_2_value]
end
pm = 1./(1+y_2./y_1)

u = [1 0.5 0.2]

y_3 = pm + 1/u(1)

plot(m,y_3)

xlabel('number of M')

ylabel('y_3')
```

Result






We can see the average delay will increase with the augment of $u$ .

Although the $\rho = \frac{\lambda}{mu}$ is a constant number, but we must consider the impact of $\frac{1}{u}$.

$$\bar{T} = \frac{\rho}{\lambda(1-\rho)} p_m + \frac{1}{u}$$

Therefore, the simulation is expected and submit to these formula.

$$\rho = \frac{\lambda}{mu}$$

$$p_0 = \frac{1}{\sum_{n=0}^{m-1} \frac{(mp)^n}{n!} + \frac{(m\rho)^m}{m!} \frac{1}{1-\rho}}$$

$$p_m = \frac{(m\rho)^m}{m!(1-\rho)} p_0$$

$$\bar{T} = \frac{\rho}{\lambda(1-\rho)} p_m + \frac{1}{u}$$