# Train station queueing system simulations

## Research as part of DATA304 and DATA474

Gale Bueno, Chad Kakau
Victoria University, Wellington, New Zealand
April - May 2022

### Model comparisons: M/M/4, Best-fit distribution, Empirical distribution

# Introduction

The data in this series of simulations was collected from the Wellington City Railway station and records service times for the public ticket counters within the station terminal. Data was collected across four separate observation sessions of two models: M/M/3 and M/M/4. M/M/C simulation is against the M/M/4 model (i.e. four servers pulling from a single queue).

Based on the data collection, and assuming a combining all sessions, the estimated parameters are:

- $\lambda = \frac{1}{23}$, or an average of one customer arriving every 23 seconds
- $\mu = \frac{1}{34}$, or an average of 34 seconds to complete each service.

As part of DATA304/DATA474 university papers, course members must observe a queuing system in action, collect data from the system in operation and compare that data against simulations based on three models:

- M1: M/M/c with parameters estimated from the data
- M2: Best-fit model, using distributions fitted from the observed data
- M3: Empirical model, using random variates drawn from a distribution based on the observations

The three simulation models are presented below. Each section describes the model being presented, states the distributions and reports three performance measures:

- W: average time spent in the system
- L: average number of customers in the queue
- B: average time servers are busy

Each model is plotted against the observed data for visual comparison.

# Model 1: M/M/c with estimated parameters

Model 1 uses the M/M/c model, where c = 4 servers, with parameters:

- $\lambda : \frac{1}{23}$
- $\mu = \frac{1}{34}$

- $\rho = \frac{\lambda}{c\mu} = \frac{\frac{1}{23}}{4*\frac{1}{34}} = \frac{1}{23} * \frac{34}{4} = \frac{34}{92} = 0.3696$

The following simulation uses these parameters, and N= 10000, R = 50.

In [2]:

```
from SimPy.Simulation import *
import random
import numpy as np
import math
import pandas as pd
import statsmodels.distributions.empirical_distribution as st
import matplotlib.pyplot as plt
```

In [3]:

```
# read in empirical data,
# read in raw data
# script in the same directory as datafile

raw_data = pd.read_csv("DATA474_Proj_data.csv",
                      parse_dates =[
                          ["Date", "Arrive"],
                          ["Date", "Serv_start"],
                          ["Date", "Serv_end"]
                      ])

# extract and store inter-arrival times for use with the draw_empirical funct
arr_data = []
for i in range(len(raw_data["Date_Arrive"])):
    if i == 0:
        arr_data.append(0)
    else:
        inter = (raw_data.loc[i, "Date_Arrive"] - raw_data.loc[i-1,"Date_Arri
        arr_data.append(inter)

# store the serving time data for use with the draw_empirical() function
serv_data = raw_data.loc[:,"Serv_time_sec"]
```

In [4]:

```
## Useful extras
def conf(L):
    """confidence interval"""
    lower = np.mean(L) - 1.96*np.std(L)/math.sqrt(len(L))
    upper = np.mean(L) + 1.96*np.std(L)/math.sqrt(len(L))
    return lower, upper
```

In [5]:

```
# Model
class Source(Process):
    """generate random arrivals"""
    def run(self, N, lamb, mu):
        for i in range(N):
            a = Arrival(str(i))
            activate(a, a.run(mu))
            t = random.expovariate(lamb)
            yield hold, self, t
```

In [6]:

```
class Arrival(Process):
    """an arrival"""
    n = 0 # class variable (number in system)
```

```
    def run(self, mu):
        # Event: arrival
        Arrival.n += 1 # number in system
        arrivetime = now()
        G.numbermon.observe(Arrival.n)
        if (Arrival.n>0):
            G.busymon.observe(1)
        else:
            G.busymon.observe(0)

        yield request, self, G.server
        # ... waiting in queue for server to be empty (delay) ...

        # Event: service begins
        t = random.expovariate(mu)

        yield hold, self, t
        # ... now being served (activity) ...

        # Event: service ends
        yield release, self, G.server

        Arrival.n-=1
        G.numbermon.observe(Arrival.n)
        if (Arrival.n>0):
            G.busymon.observe(1)
        else:
            G.busymon.observe(0)
        delay = now()-arrivetime
        G.delaymon.observe(delay)
```

In [7]:

```
class G:
    server = 'dummy'
    delaymon = 'Monitor'
    numbermon = 'Monitor'
    busymon = 'Monitor'
```

In [8]:

```
def model(c, N, lamb, mu, maxtime, rvseed):
    # setup
    initialize()
    random.seed(rvseed)
    G.server = Resource(c, monitored = True)
    G.delaymon = Monitor()
    G.numbermon = Monitor()
    G.busymon = Monitor()

    Arrival.n = 0

    # simulate
    s = Source('Source')
    activate(s, s.run(N, lamb, mu))
    simulate(until=maxtime)

    # gather performance measures
    W = G.delaymon.mean()
    L = G.numbermon.timeAverage()
    B = G.busymon.mean()
#     Busy = G.server.actMon.mean()
    return(W,L,B)
```

In [9]:
```
## Experiment ----------------
allW = []
allL= []
allB = []
allLambdaEffective = []
# allBmon = []
for k in range(50):
    seed = 123*k
    result = model(c=4, N=10000, lamb=1/23.02481, mu=1/34.00496, maxtime=2000
    allW.append(result[0])
    allL.append(result[1])
    allB.append(result[2])
    allLambdaEffective.append(result[1]/result[0])
#     allBmon.append(result[3])

m1_W = np.mean(allW)
m1_L = np.mean(allL)
m1_B = np.mean(allB)
print("Estimate of W:", np.mean(allW))
print("Conf in of W:", conf(allW))
print("Estimate of L:", np.mean(allL))
print("Conf in of L:", conf(allL))
print("Estimate of B:", np.mean(allB))
print("Conf int of B:", conf(allB))
print("Estimate of LambdaEffective:", np.mean(allLambdaEffective))
print("Conf int of LambdaEffective:", conf(allLambdaEffective))
# print("Estimate from the Resource monitor: ", np.mean(allBmon))
# print("Conf int of Bmon: ", conf(allBmon))
```

```
Estimate of W: 34.92652195180287
Conf in of W: (34.7901775790929, 35.06286632451284)
Estimate of L: 1.5182172622497612
Conf in of L: (1.5099002974890192, 1.5265342270105031)
Estimate of B: 0.8871660000000001
Conf int of B: (0.8862071755323273, 0.8881248244676729)
Estimate of LambdaEffective: 0.04346742955753993
Conf int of LambdaEffective: (0.04333236956761407, 0.043602489547465796)
```

### M/M/4 expected and simulated performance measures

From an M/M/c queuing system with:

c = 4,
$\lambda = \frac{1}{23} = 0.0435$,
$\mu = \frac{1}{34} = 0.0294$,
$\frac{\lambda}{\mu} = \frac{34}{23} = 1.4783$, and
$\rho = \frac{\lambda}{c\mu} = \frac{0.0435}{4*0.0294} = 0.3699$

We expect:

$L = L_q + \frac{\lambda}{\mu}$ , where:

$$\begin{aligned} L_q &= \frac{(\frac{\lambda}{\mu})^{c+1}}{(c-1)!(c-\frac{\lambda}{\mu})^2} \\ &= \frac{(1.4783)^5}{6(4-1.4783)^2} \\ &= \frac{7.060}{38.1538} \\ &= 0.1850 \end{aligned}$$

so,

$L = 0.1850 + 1.4783 = 1.6633.$

On average, we expect 1.63 customers in the system at any given time. This is higher than the simulated value and outside of the confidence interval (1.5099, 1.5265).

According to Little's Law:

$L = \lambda W$, therefore:
$W = \frac{L}{\lambda} = \frac{1.6633}{\frac{1}{23}} = 38.2559$

On average, we expect each customer to remain in the system for 38.2559 seconds. This is higher than the simulation estimate and outside of the confidence interval (34.7901, 35.0629).

$B = \rho = \frac{\lambda}{c\mu} = 0.3699.$

On average, servers are busy for 36.99% of the time. This is well below the simulation estimate and outside the convidence interval (88.6%, 88.8%)

Comparing the estimated values for M/M/4:

L = 1.52 customers in the system on average W = 38.26 seconds for a customer in the system B = 0.37 utilisation

The simulated values, where N = 10,000 and r = 50, arrival times $Exp(\frac{1}{\lambda})$, and server times $Exp(\frac{1}{mu})$

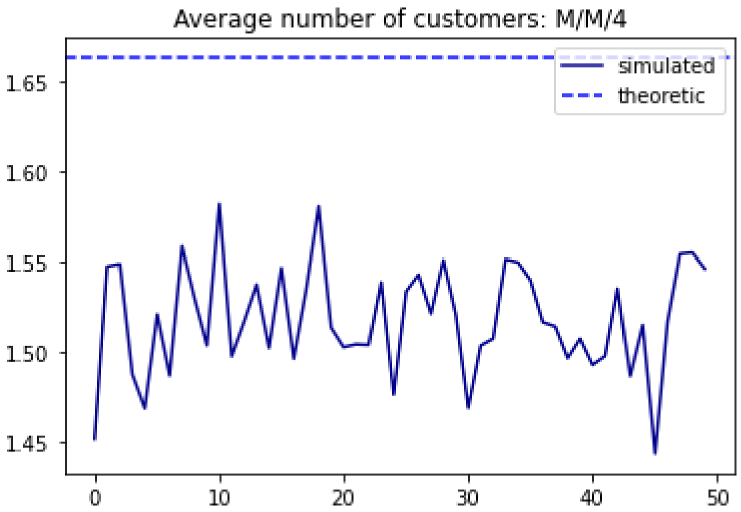
L = 1.23 customers in the system W = 34.93 seconds for a customer in the system B = 0.89 utilisation of servers

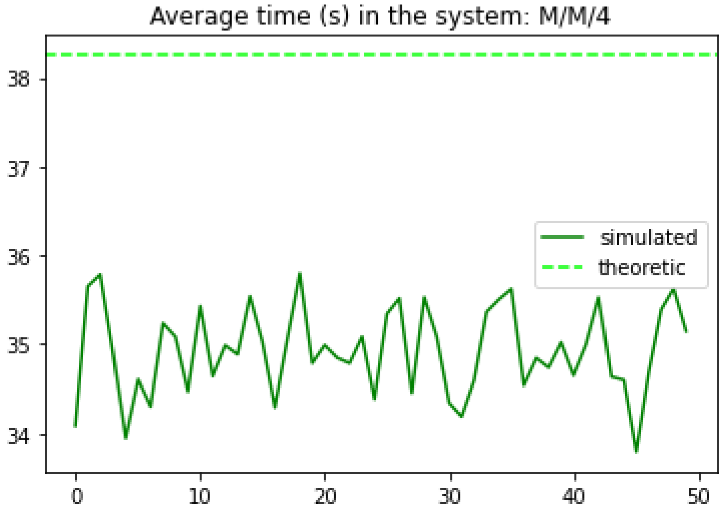
The plots below provide visualisations of the simulated performance against the estimated performance. Clearly, the simulation performance was below estimates for average number of customers in the system and average time in the system, but server utilisation was more than twice as high in the simulation.

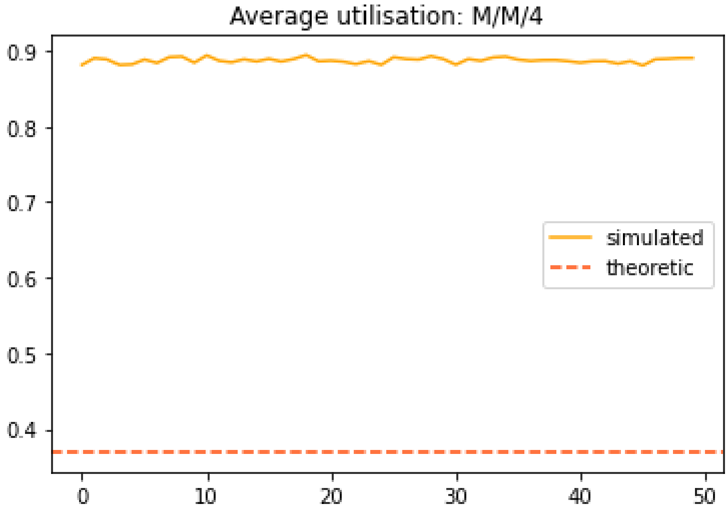
In [10]:

```
plt.plot(allL, color = "darkblue", label = "simulated") # average number of c
plt.axhline(1.663, color = "blue", label = "theoretic", ls = "--")
plt.title("Average number of customers: M/M/4")
plt.legend()
plt.show()

plt.plot(allW, color = "green", label = "simulated") # average time in the sy
plt.axhline(38.2559, color = "lime", label = "theoretic", ls = "--")
```

```
plt.title("Average time (s) in the system: M/M/4")
plt.legend()
plt.show()

plt.plot(allB, color = "orange", label = "simulated") # average utilisation i
plt.axhline(0.3699, color = "orangered", label = "theoretic", ls = "--")
plt.title("Average utilisation: M/M/4")
plt.legend()
plt.show()
```




# Model 2: best-fit distributions

Model 2 - should use the interarrival times best fit (Gamma(2, 2/$\lambda$)) and the service times best fit

(Exp($1/\mu$)) distributions to simulate the performance of the service unit (the best fit model).

In [11]:

```
# Model 2
class Source2(Process):
    """generate random arrivals"""
    def run(self, N, lamb, mu):
        for i in range(N):
            a = Arrival2(str(i))
            activate(a, a.run(mu))
#             t = random.expovariate(lamb)
            t = np.random.gamma(2, 2/lamb)
            yield hold, self, t
```

In [12]:

```
class Arrival2(Process):
    """an arrival"""
    n = 0 # class variable (number in system)

    def run(self, mu):
        # Event: arrival
        Arrival2.n += 1 # number in system
        arrivetime = now()
        G.numbermon.observe(Arrival2.n)
        if (Arrival2.n>0):
            G.busymon.observe(1)
        else:
            G.busymon.observe(0)

        yield request, self, G.server
        # ... waiting in queue for server to be empty (delay) ...

        # Event: service begins
        t = random.expovariate(mu)

        yield hold, self, t
        # ... now being served (activity) ...

        # Event: service ends
        yield release, self, G.server

        Arrival2.n-=1
        G.numbermon.observe(Arrival2.n)
        if (Arrival2.n>0):
            G.busymon.observe(1)
        else:
            G.busymon.observe(0)
        delay = now()-arrivetime
        G.delaymon.observe(delay)
```

In [13]:

```
def model2(c, N, lamb, mu, maxtime, rvseed):
    # setup
    initialize()
    random.seed(rvseed)
    G.server = Resource(c)
    G.delaymon = Monitor()
    G.numbermon = Monitor()
    G.busymon = Monitor()

    Arrival2.n = 0

    # simulate
```

```
    s = Source2('Source')
    activate(s, s.run(N, lamb, mu))
    simulate(until=maxtime)

    # gather performance measures
    W = G.delaymon.mean()
    L = G.numbermon.timeAverage()
    B = G.busymon.timeAverage()

    return(W,L,B)
```

In [14]:

```
## Experiment ----------------
allW2 = []
allL2= []
allB2 = []
allLambdaEffective2 = []
for k in range(50):
    seed = 123*k
    result = model2(c=4, N=10000, lamb=1/23, mu=1/34, maxtime=2000000, rvseec
    allW2.append(result[0])
    allL2.append(result[1])
    allB2.append(result[2])
    allLambdaEffective2.append(result[1]/result[0])

m2_W = np.mean(allW2)
m2_L = np.mean(allL2)
m2_B = np.mean(allB2)
m2_Leff = np.mean(allLambdaEffective2)

print("Estimate of W2:", np.mean(allW2))
print("Conf in of W2:", conf(allW2))
print("Estimate of L2:", np.mean(allL2))
print("Conf in of L2:", conf(allL2))
print("Estimate of B2:", np.mean(allB2))
print("Conf int of B2:", conf(allB2))
print("Estimate of LambdaEffective:", np.mean(allLambdaEffective2))
print("Conf int of LambdaEffective:", conf(allLambdaEffective2))
```

```
Estimate of W2: 33.96332372506943
Conf in of W2: (33.87209711092889, 34.05455033920997)
Estimate of L2: 0.3686446681294052
Conf in of L2: (0.36752164797532444, 0.369767688283486)
Estimate of B2: 0.33034064116484196
Conf int of B2: (0.32943121492163907, 0.33125006740804486)
Estimate of LambdaEffective: 0.010854329985812714
Conf int of LambdaEffective: (0.010832698661550608, 0.01087596131007482)
```

Comparing to estimated values for:

L = 1.66 customers in the system on average W = 38.26 seconds for a customer in the system B = 0.37 utilisation

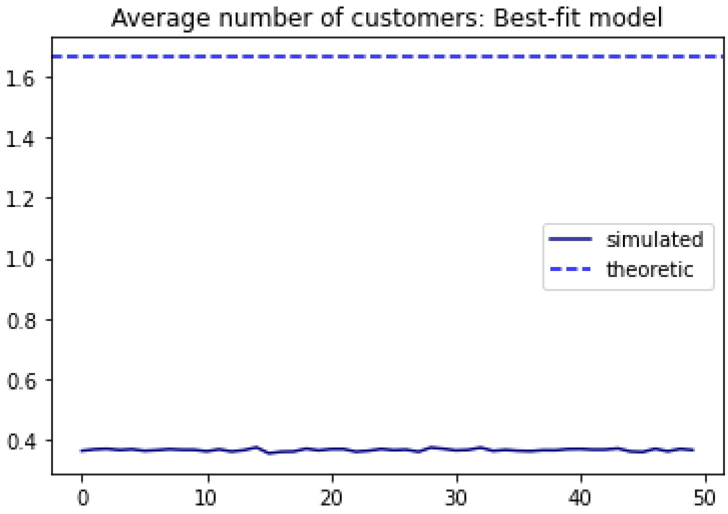
The simulated values, where N = 10,000 and r = 50, arrival times $Gamma(2, \frac{2}{\lambda})$, and server times $Exp(\frac{1}{\mu})$

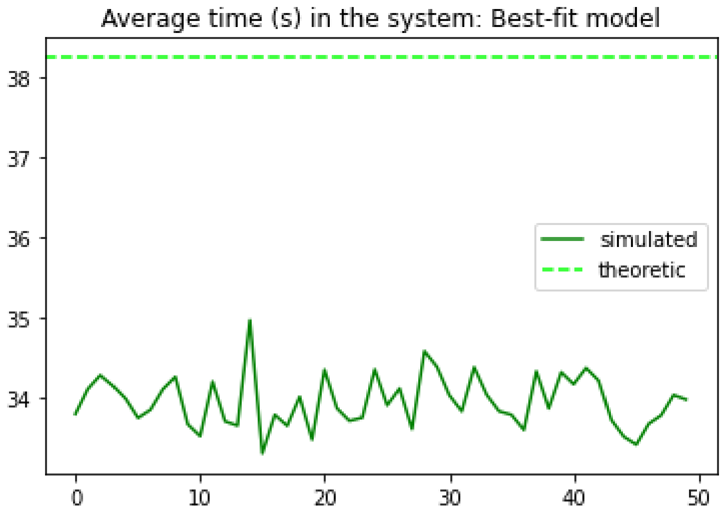
L = 0.37 customers in the system W = 33.96 seconds for a customer in the system B = 0.33 utilisation of servers

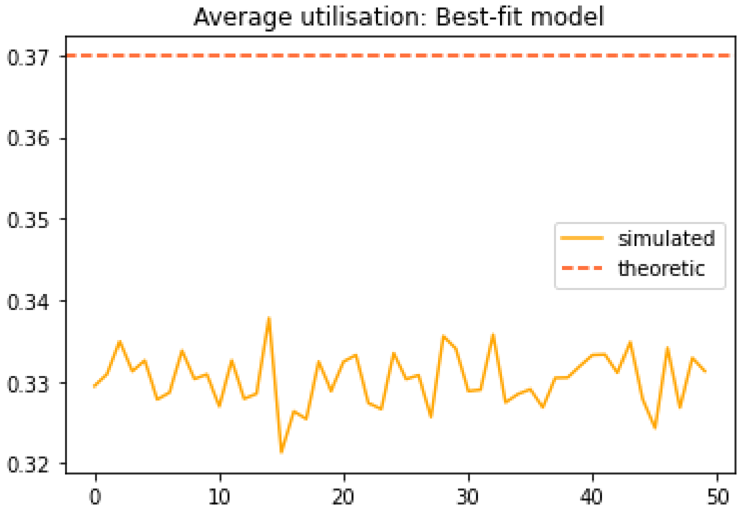
The plots below provide visualisations of the simulated performance against the estimated performance. Clearly, the simulation performance was below estimates for average number of customers in the system and average time in the system, but server utilisation was around twice as high in the simulation.

In [15]:

```
plt.plot(allL2, color = "darkblue", label = "simulated") # average number of
plt.axhline(1.663, color = "blue", label = "theoretic", ls = "--")
plt.title("Average number of customers: Best-fit model")
plt.legend()
plt.show()

plt.plot(allW2, color = "green", label = "simulated") # average time in the s
plt.axhline(38.2559, color = "lime", label = "theoretic", ls = "--")
plt.title("Average time (s) in the system: Best-fit model")
plt.legend()
plt.show()

plt.plot(allB2, color = "orange", label = "simulated") # average utilisation
plt.axhline(0.3699, color = "orangered", label = "theoretic", ls = "--")
plt.title("Average utilisation: Best-fit model")
plt.legend()
plt.show()
```






# Model 3: empirical distribution

Model 3 - should use the empirical distribution of the interarrival time and the empirical distribution the service time to simulate the performance of your service unit (the empirical model).

In [16]:

```
""" draw from an empirical distribution, uses the inverse
    transformation method and linear interpolation"""

import numpy as np
import random
import scipy
import pylab

def draw_empirical(data, r):
    """one draw (for given r ~ U(0,1)) from the
    empirical cdf based on data"""

    d = {x: data.count(x) for x in data}
    obs_values, freq = zip( *sorted( zip(d.keys(), d.values())))
    obs_values = list(obs_values)
    freq = list(freq)
    empf = [x*1.0/len(data) for x in freq]
    ecum = np.cumsum(empf).tolist()
    ecum.insert(0, 0)
    obs_values.insert(0,0)

    for x in ecum:
         if r <= x:
            rpt = x
            break
    r_end = ecum.index(rpt)
    y = obs_values[r_end] - 1.0*(ecum[r_end]-r)*(obs_values[r_end]-
        obs_values[r_end-1])/(ecum[r_end]-ecum[r_end-1])
    return y
```

In [17]:

```
# Replace previous Model 3 to use draw_empirical function rather than actual
class Source3(Process):
    """generate random arrivals"""
    def run(self, N, arr_data, serv_data):
        for i in range(N):
            a = Arrival3(str(i))
            activate(a, a.run(list(serv_data)))
```

```
            r = random.random()
            t = draw_empirical(list(arr_data), r)
            yield hold, self, t
```

In [18]:

```
class Arrival3(Process):
    """an arrival"""
    n = 0 # class variable (number in system)

    def run(self, serv_data):
        # Event: arrival
        Arrival3.n += 1 # number in system
        arrivetime = now()
        G.numbermon.observe(Arrival3.n)

        if (Arrival3.n>0):
            G.busymon.observe(1)
        else:
            G.busymon.observe(0)

        yield request, self, G.server
        # ... waiting in queue for server to be empty (delay) ...

        # Event: service begins
        r = random.random()
        t = draw_empirical(list(serv_data), r)

        yield hold, self, t
        # ... now being served (activity) ...

        # Event: service ends
        yield release, self, G.server

        Arrival3.n-=1

        G.numbermon.observe(Arrival3.n)
        if (Arrival3.n>0):
            G.busymon.observe(1)

        else:
            G.busymon.observe(0)

        delay = now()-arrivetime
        G.delaymon.observe(delay)
```

In [19]:

```
def model3(c, N, maxtime, rvseed, arr_data, serv_data):
    # setup
    initialize()
    random.seed(rvseed)
    G.server = Resource(c)
    G.delaymon = Monitor()
    G.numbermon = Monitor()
    G.busymon = Monitor()

    Arrival3.n = 0

    # simulate
    s = Source3('Source')
    activate(s, s.run(N, arr_data, serv_data))
    simulate(until=maxtime)
```

```
    # gather performance measures
    W = G.delaymon.mean()
    L = G.numbermon.timeAverage()
    B = G.busymon.timeAverage()
    B_2 = G

    return(W,L,B)
```

In [20]:

```
## Experiment ----------------
allW3 = []
allL3= []
allB3 = []
allLambdaEffective3 = []

for k in range(50):
    seed = 123*k
    result = model3(c=4,
                    N=10000,
                    arr_data = arr_data,
                    serv_data = serv_data,
                    maxtime=20000,
                    rvseed=seed)
    allW3.append(result[0])
    allL3.append(result[1])
    allB3.append(result[2])
    allLambdaEffective3.append(result[1]/result[0])

m3_W = np.mean(allW3)
m3_L = np.mean(allL3)
m3_B = np.mean(allB3)
m3_Leff = np.mean(allLambdaEffective3)

print("Estimate of W3:", np.mean(allW3))
print("Conf in of W3:", conf(allW3))
print("Estimate of L3:", np.mean(allL3))
print("Conf in of L3:", conf(allL3))
print("Estimate of B3:", np.mean(allB3))
print("Conf int of B3:", conf(allB3))
print("Estimate of LambdaEffective3:", np.mean(allLambdaEffective3))
print("Conf int of LambdaEffective3:", conf(allLambdaEffective3))
```

```
Estimate of W3: 35.027567545535085
Conf in of W3: (33.26826603193133, 36.78686905913884)
Estimate of L3: 0.15657527321208314
Conf in of L3: (0.1143735645199232, 0.1987769819042431)
Estimate of B3: 0.06603462976458066
Conf int of B3: (0.048934897615249745, 0.08313436191391158)
Estimate of LambdaEffective3: 0.004260000000000001
Conf int of LambdaEffective3: (0.003132238413848034, 0.005387761586151968)
```

Comparing the estimated values for M/M/4:

L = 1.52 customers in the system on average

W = 38.26 seconds for a customer in the system

B = 0.37 utilisation

The simulated values, where N = 10,000 and r = 50, arrival times and server times are sampled from the empirical distribution

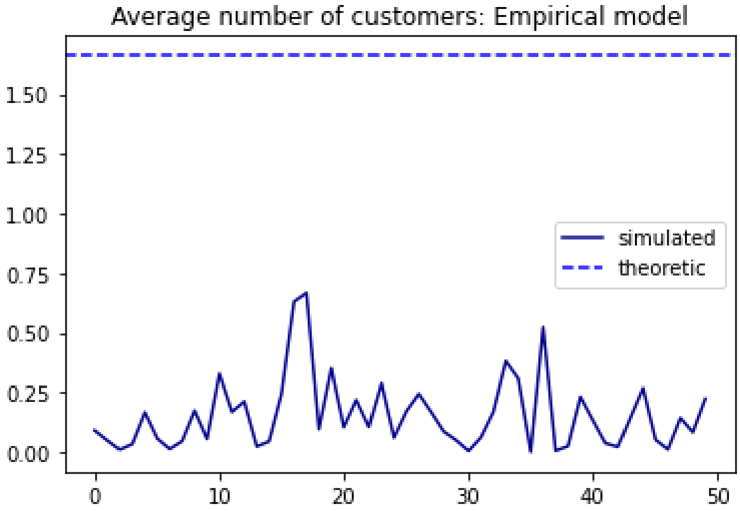
L = 0.15 customers in the system

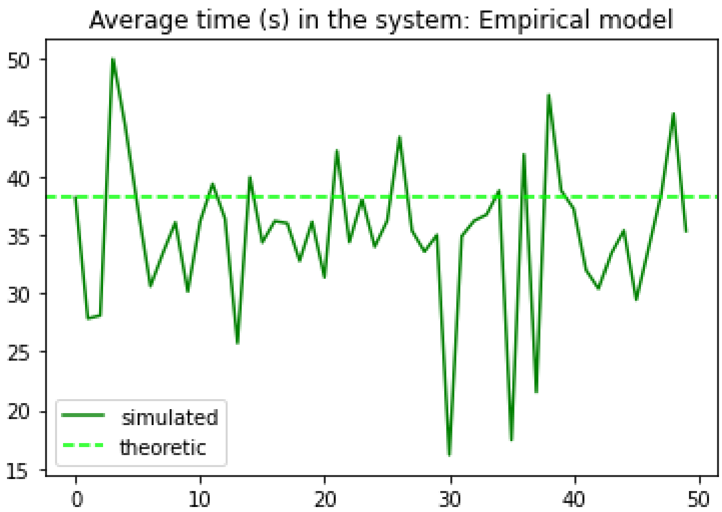
W = 35.02 seconds for a customer in the system

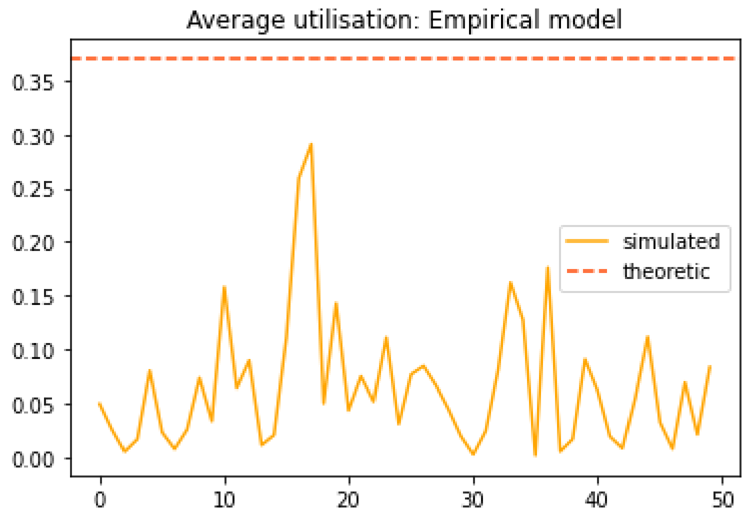
B = 0.06 utilisation of servers

The plots below provide visualisations of the simulated performance against the estimated performance. Clearly, the simulation performance was just below estimates for average time in the system, but was under the estimated number of customers by an order of magnitude and at just one sixth of the utilisation rate.

In [21]:
```
plt.plot(allL3, color = "darkblue", label = "simulated") # average number of
plt.axhline(1.663, color = "blue", label = "theoretic", ls = "--")
plt.title("Average number of customers: Empirical model")
plt.legend()
plt.show()

plt.plot(allW3, color = "green", label = "simulated") # average time in the s
plt.axhline(38.2559, color = "lime", label = "theoretic", ls = "--")
plt.title("Average time (s) in the system: Empirical model")
plt.legend()
plt.show()

plt.plot(allB3, color = "orange", label = "simulated") # average utilisation
plt.axhline(0.3699, color = "orangered", label = "theoretic", ls = "--")
plt.title("Average utilisation: Empirical model")
plt.legend()
plt.show()
```



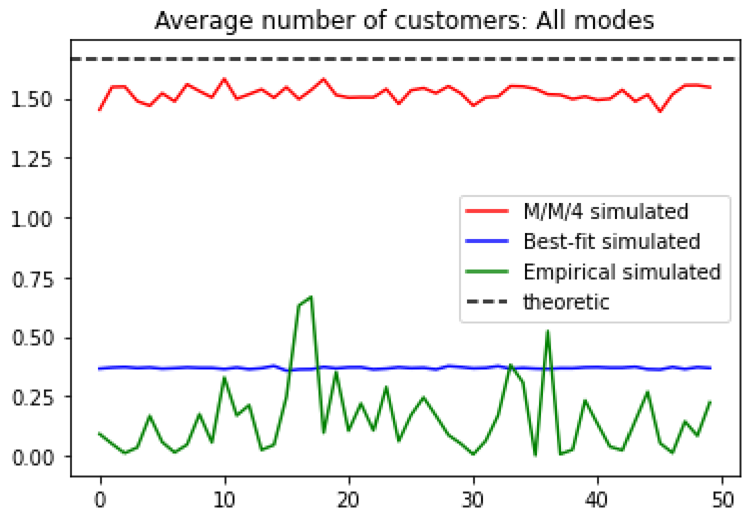



# Comparison of performance across each model

When comparing all plots against the estimated model, the plots show that all models under-estimated the average number of customers in the system. The empirical model simulated a much lower average number of customers than the other two models.

In [22]:

```
plt.plot(allL, color = "red", label = "M/M/4 simulated") # average number of
plt.plot(allL2, color = "blue", label = "Best-fit simulated") # average numbe
plt.plot(allL3, color = "green", label = "Empirical simulated") # average num
plt.axhline(1.663, color = "black", label = "theoretic", ls = "--")
plt.title("Average number of customers: All modes")
plt.legend()
plt.show()
```

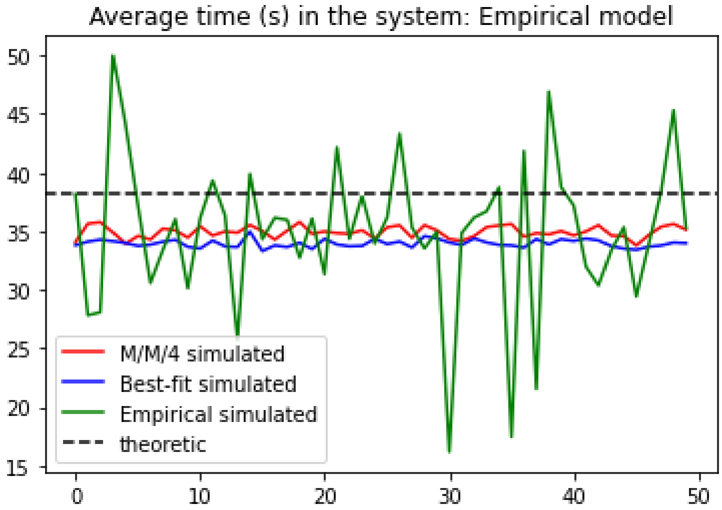



All simulations under-estimated the amount of time spent in the system. Of all the performance measures, this is the only measure where models overlapped, with most overlap between M1 (M/M/4) and M3 (Empirical) models. The Empirical mode also had some overlap with M2 (Best-fit).

In [23]:

```
plt.plot(allW, color = "red", label = "M/M/4 simulated") # average number of
plt.plot(allW2, color = "blue", label = "Best-fit simulated") # average numbe
plt.plot(allW3, color = "green", label = "Empirical simulated") # average num
plt.axhline(38.2559, color = "black", label = "theoretic", ls = "--")
plt.title("Average time (s) in the system: Empirical model")
```

```
plt.legend()
plt.show()
```

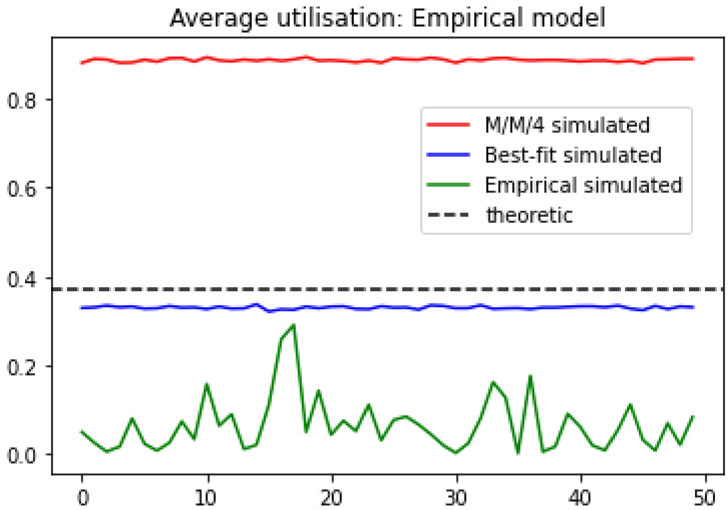



M1 simulation reported higher utilisation rates than expected, but M3 and M2 reported lower utilisation than estimated.

Of the three simulations, the performance across all three measures was more stable (less variability) for M1 and M2, with much larger variance across the 50 replications for M3.

In [36]:

```
plt.plot(allB, color = "red", label = "M/M/4 simulated") # average number of
plt.plot(allB2, color = "blue", label = "Best-fit simulated") # average numbe
plt.plot(allB3, color = "green", label = "Empirical simulated") # average nu
plt.axhline(0.3699, color = "black", label = "theoretic", ls = "--")
plt.title("Average utilisation: Empirical model")
plt.legend(loc = (0.55, 0.55))
plt.show()
```




In [ ]: