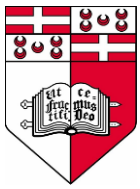


CCE2501
Modelling and Computer Simulation

Assignment

Compiled by: Marc Ferriggi (286397M)

Lecturer: Dr. Gianluca Valentino

Department of Communications and Computer Engineering
University of Malta

December 2016

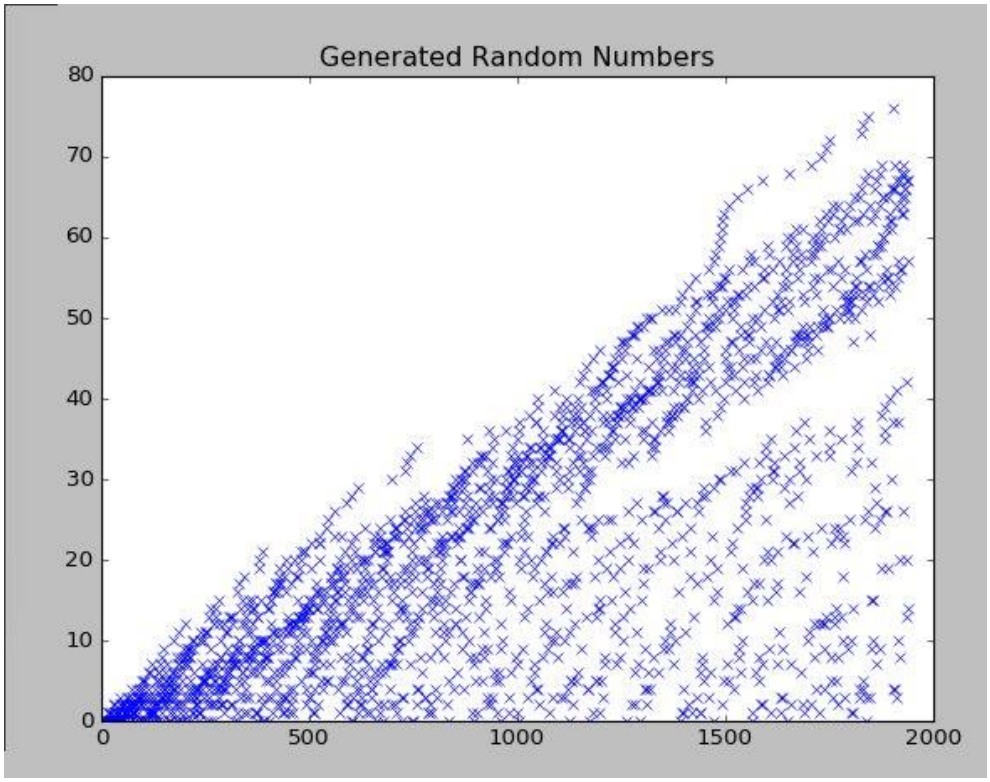
Question 1a) – Develop an algorithm in Python based on Von-Neumann's accept-reject method to generate random numbers under y(t).

To generate the desired random numbers, the following code was used:

```
import math
import random
import matplotlib.pyplot as plt
from scipy.stats import lognorm
import numpy as np

#sigma and mu of the log-normal distribtion:
sig = 0.75
mu = 0

#limits of t:
a = 0
b = 5

#from the plot of y(t), M was taken to be 1
M = 1

PI = math.pi
binRange = 5

rand_lognormal = np.zeros(100)

# Create an array of 100 bins from 0 to binRange
bins = np.linspace(0,binRange,100)

#save accepted random numbers:
generated_random_numbers = []

# Generate some uniform random numbers and convert them to a log-normal
distribution using the accept reject method
for i in range(0, 10000):

    u = a + (b-a)*random.random() #uniform
    v = M*random.random() #h(t)


    f = (1/(u*sig*math.sqrt(2*PI))*math.exp((-(math.log(u-
mu))**2)/(2*(sig**2))))


    if v < f:
        for j in range(0, len(bins)):

            if u < bins[j]:
                generated_random_numbers.append(rand_lognormal[j])
                rand_lognormal[j] += 1
                break
```

```
#show random numbers
plt.figure()
plt.plot(generated_random_numbers, 'x')
plt.title('Generated Random Numbers')
plt.show()
```

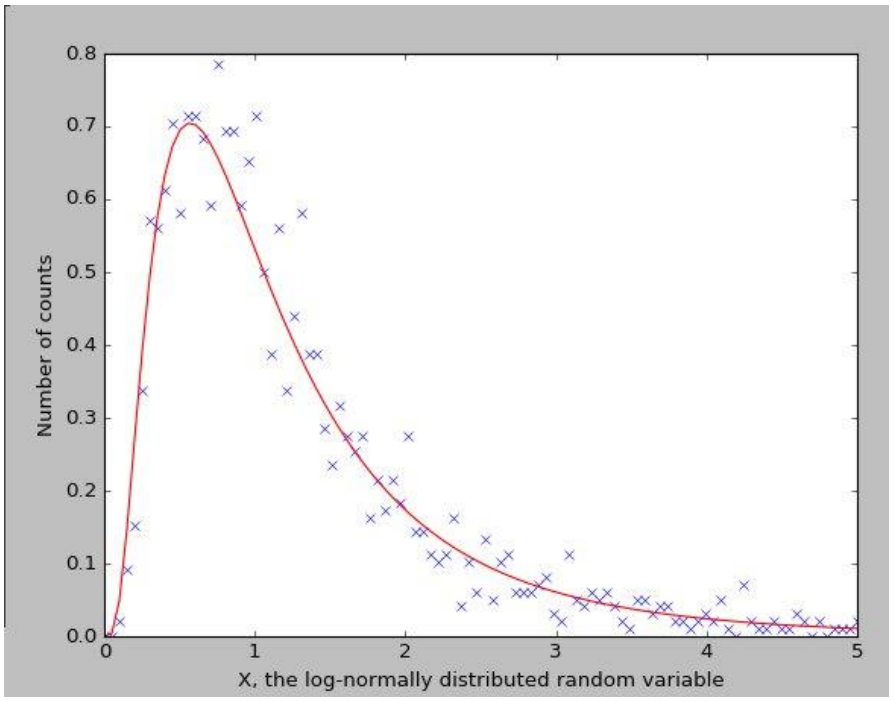
Below is the resulting plot showing the values of the generated random numbers:



Question 1b) - Generate 10000 points in order to plot the proper probability density function with blue scatter points, and superimpose a red curve of the log-normal distribution produced using the standard Python library.

The following code was added to that of question 1a to generate the plot seen below:

```
curve = np.linspace(0,binRange,100)
toPlot = lognorm.pdf(curve, sig)

#calculate area:
area = 0
for i in range(1, len(rand_lognormal)):
    area += (bins[i]-bins[i-1]) * rand_lognormal[i]

new_lognormal = [i/area for i in rand_lognormal]


plt.figure()
plt.plot(bins, new_lognormal, 'bx') #rand_lognormal was divided by area to
turn into a pdf.
plt.plot(curve,toPlot, 'r-')
plt.ylabel('Number of counts')
plt.xlabel('X, the log-normally distributed random variable')
plt.show()
```



Question 2a) - Develop a conceptual model made up of queuing systems for the traffic-lights controlled T-Junction. Explain all your reasoning.

In total, four separate simulations will be run, two for roads A and B respectively in the morning hours and two for the roads A and B respectively in the evening hours. In each simulation, the tasks shall be placed in a list. These tasks include the Arrival Event, Begin Service Event, End Service Event, Red Light Event and Green Light Event. Throughout the whole simulation, cars shall arrive at the traffic lights at an inter-arrival rate based on the Exponential distribution with parameter $\lambda=2$. This shall be normalised with a Gaussian distribution whose parameters vary according to the road and time.

When the server is free, the service time is fixed to 2 seconds. The Red Light and Green Light events are called accordingly every 30 seconds to set the server to busy when there is a red light. The queue length shall be charted every time there is a change and then this shall be represented graphically in the required plots.

Question 2b) - Write GSL code including the initialization and event routines to implement your conceptual model.

Routine Initialize

```
Q := 0
S := idle
Schedule arrival event at time NOW
Schedule red-light event at time NOW+30
```

Simulate

```
WHILE NOW < 14400 DO:
     Update the clock to the time on the first event notice
     Execute the event routine for the type of event on the
     first event notice.
     Discard the first event notice.
```

Event routine *arrival*

```
Schedule arrival event at time NOW + inter-arrival time
Q := Q + 1
IF S := idle THEN schedule begin service event at time NOW
```

Event routine *begin service*

```
Q := Q - 1
S := busy
Schedule end service at time NOW + service time
```

Event routine *end service*

```
S := idle
IF Q > 0
     THEN schedule begin service at time NOW
```

Event routine *red-light*

```
S := busy
Schedule green-light event at time NOW+30
Schedule red-light event at time NOW+60
```

Event routine *green-light*

```
S := idle
```

Question 2c) - Implement your model in Python and run two separate simulations for morning and evening, plotting the queue lengths as a function of time in roads A and B in both cases. Calculate the traffic intensities for both roads and both times.

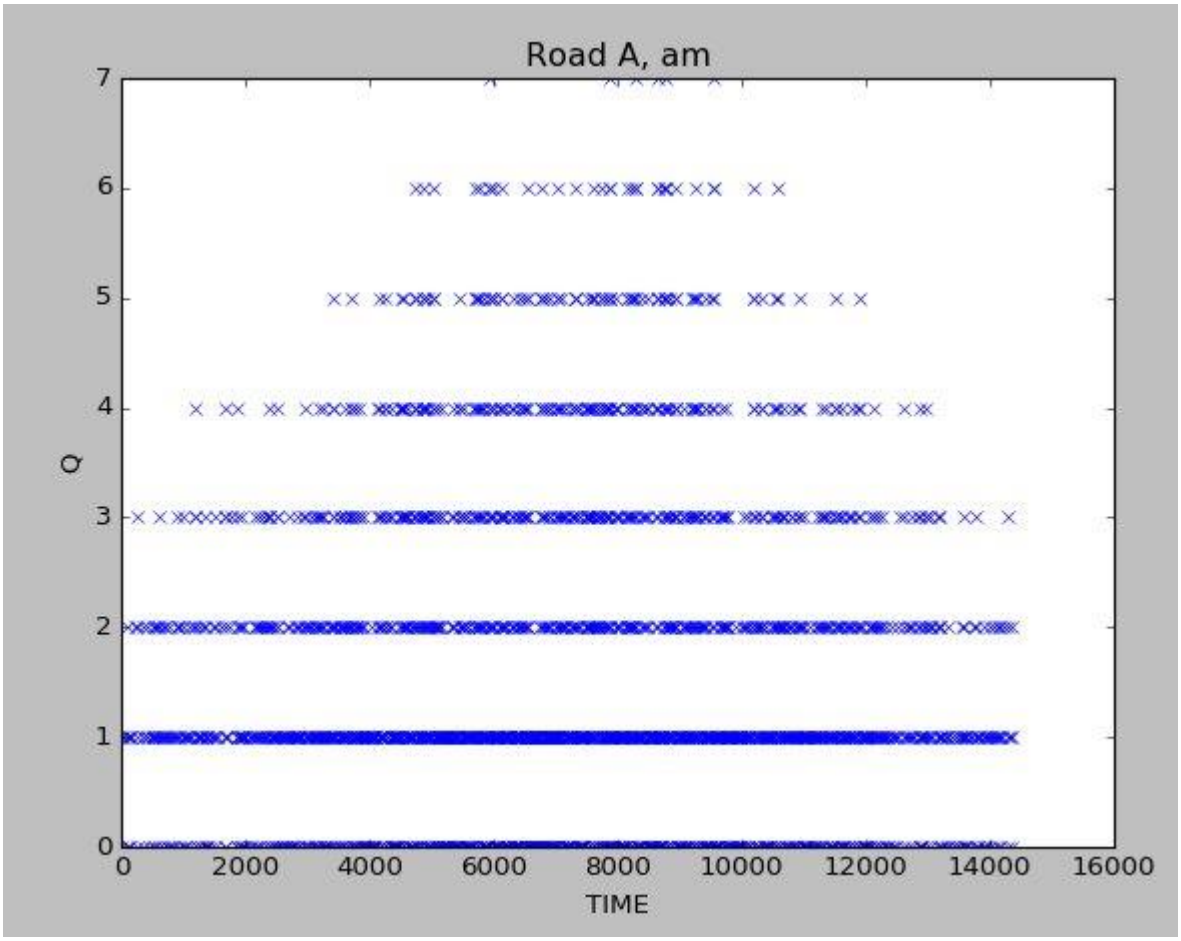
The code used to run the simulation for road A in the morning can be seen below. The parameters were then changed to simulate Road B, am; Road A, pm; and Road B, pm. The resultant plots can be seen further below.

```
import numpy as np
from operator import itemgetter
from scipy.stats import norm
from matplotlib import pyplot as plt

# State Variables:
Q = 0  # Number of cars
S = 0  # 0 - idle, 1 - busy
NOW = 0  # NOW
expBeta = 2
normMu = 7200
normSig = 60 * 60
queueArray = []

# Initialize Event list:
eventList = []
#To calculate traffic intensity:
interArrivalTimes = []
serviceTimes = []
# Schedule an arrival event:
eventList.append([0, 0])
#schedule a red light at 30:
eventList.append([3, 30])


def arrival():
    global NOW, Q, eventList, S
    global poisslambda, queueArray

    # generate a random number from an exponential distribution with bounds
at [1,10]
    rnd_exp = 0
    while rnd_exp < 1 or rnd_exp > 10:
        rnd_exp = np.random.exponential(expBeta)

    div = norm.pdf(NOW/3600,normMu/3600,normSig/3600)
    iat = rnd_exp / div

    interArrivalTimes.append(iat) #for traffic intensities

    eventList.append([0, NOW + iat])
    Q = Q + 1
    queueArray.append([Q, NOW])
    if S == 0:
        eventList.append([1, NOW])
```

```
def beginervice():
    global Q, S, expBeta, eventList, NOW, queueArray
    S = 1
    st = 2
    serviceTimes.append(st) # for traffic intensities
    eventList.append([2, NOW + st])
    Q = Q - 1
    queueArray.append([Q, NOW])


def endService():
    global S, NOW, eventList, Q
    S = 0
    if Q > 0:
        # Schedule begin service event at time NOW
        eventList.append([1, NOW])

def redLight():
    global S, NOW, eventList
    S = 1 # set server to busy
    eventList.append([4, NOW + 30]) #schedule green light in 30s
    eventList.append([3,NOW+60]) #schedule stop in 1 minute

def greenLight():
    global S, NOW, eventList
    S = 0 #set server to idle

while NOW < 14400:
    # sort list based on second column:
    eventList.sort(key=itemgetter(1))
    firstEvent = eventList[0]
    NOW = firstEvent[1]

    if firstEvent[0] == 0:  # if event type is arrival
        arrival()
    elif firstEvent[0] == 1:  # if event type is begin service
        beginervice()
    elif firstEvent[0] == 2:  # event type is end Serviece
        endService()
    elif firstEvent[0] == 3:
        redLight()
    elif firstEvent [0] == 4:
        greenLight()
    # Remove first element from list
    del eventList[0]


npQueue = np.array(queueArray)
x = npQueue[:, 1]
y = npQueue[:, 0]

plt.figure()
plt.plot(x, y, 'x')
plt.xlabel("TIME")
plt.ylabel("Q")
plt.title("Road A, am")
plt.show()
```

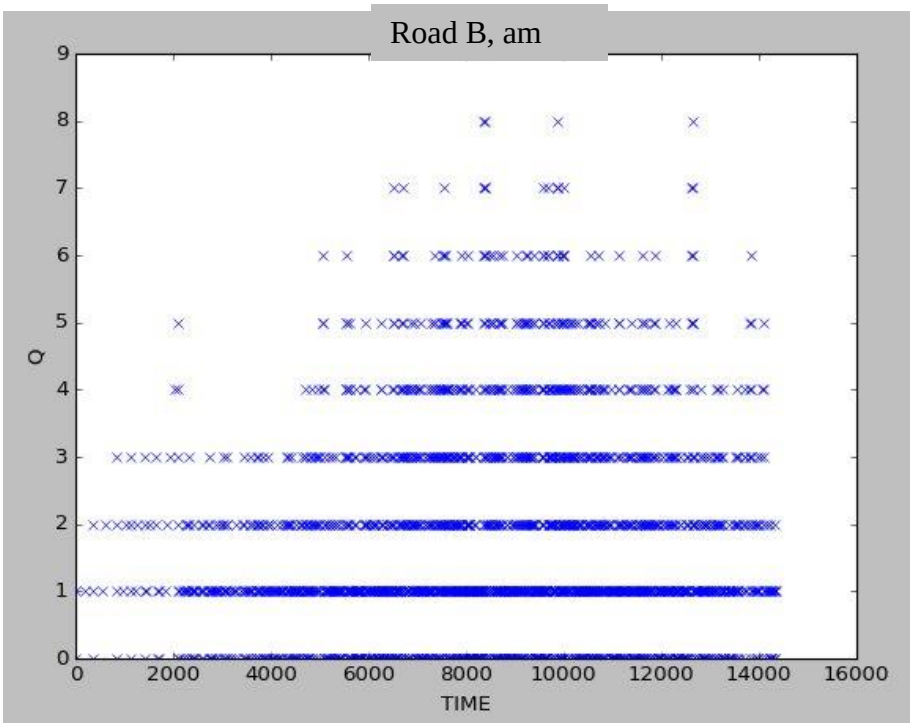
```
#Traffic Intensity:
avg_iat = sum(interArrivalTimes)/len(interArrivalTimes)
avg_st = sum(serviceTimes)/len(serviceTimes)

trafficIntensity = avg_st/avg_iat
print("Traffic Intensity: ", trafficIntensity)
```

Road A, am:



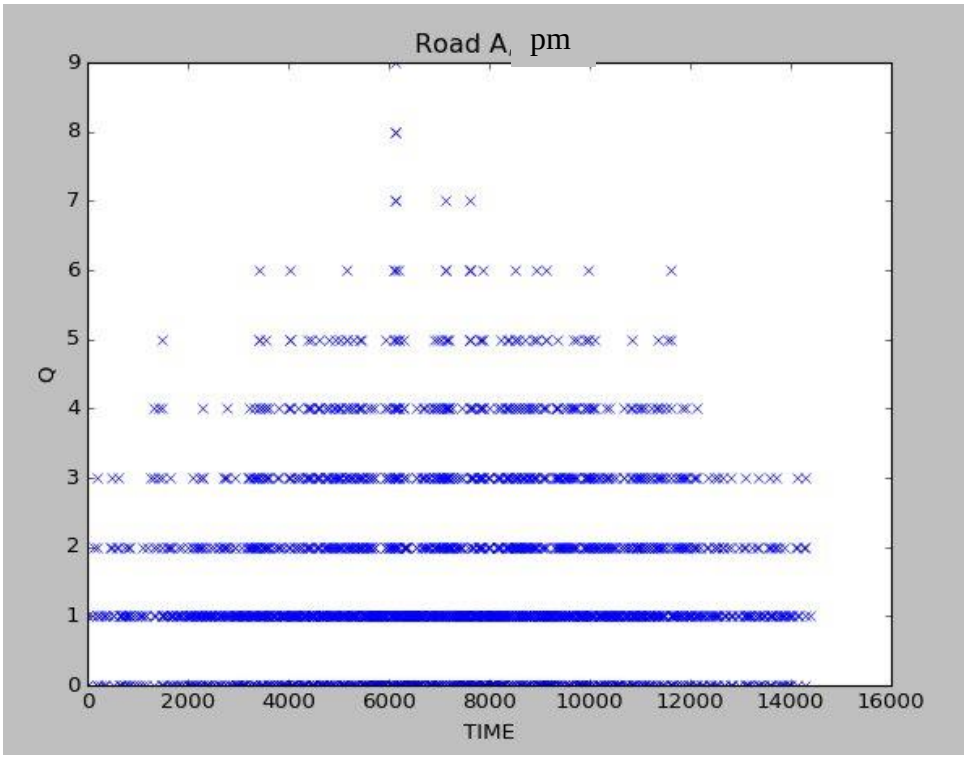
Traffic Intensity: 0.170246274641

Road B, am:



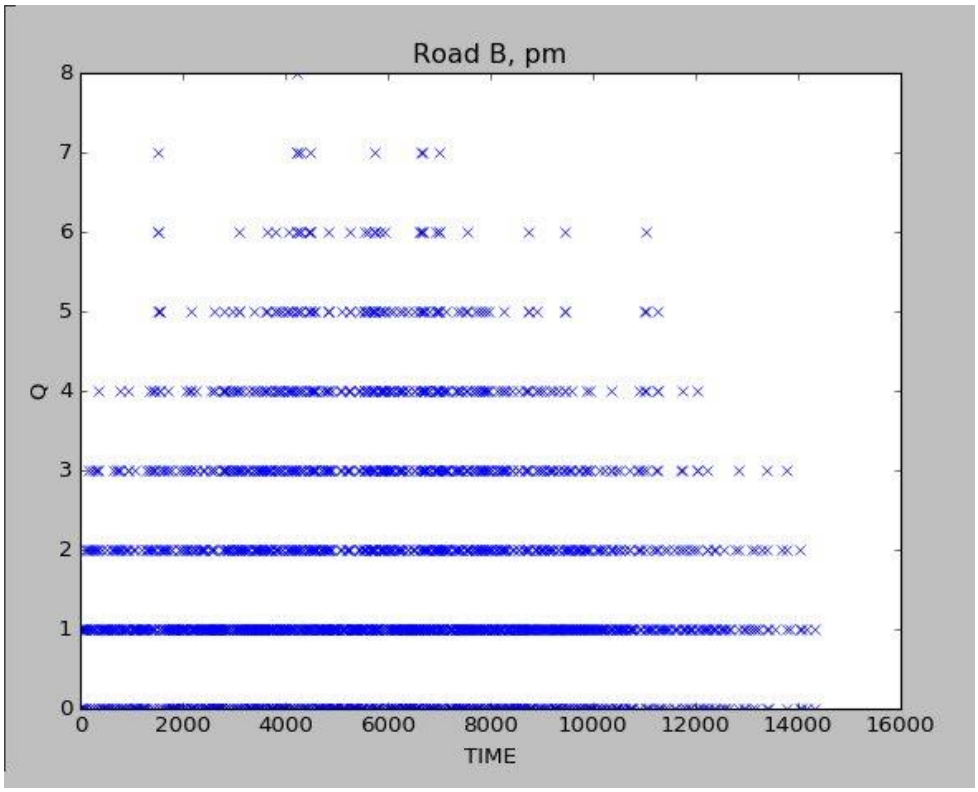
Traffic Intensity: 0.156066456296

Road A, pm:



Traffic Intensity: 0.16189704296

Road B, pm:



Traffic Intensity: 0.159396307019

Question 2d) - What difference do you notice between roads A and B in the morning and evening, and what action can the transport authorities take to reduce the overall traffic?

From the plots shown above, it seems like there seems to be longer traffic queues in road B than road A in both the morning and the evenings, however one can also note that it seems like the longest queues happen at different times for roads A and B.

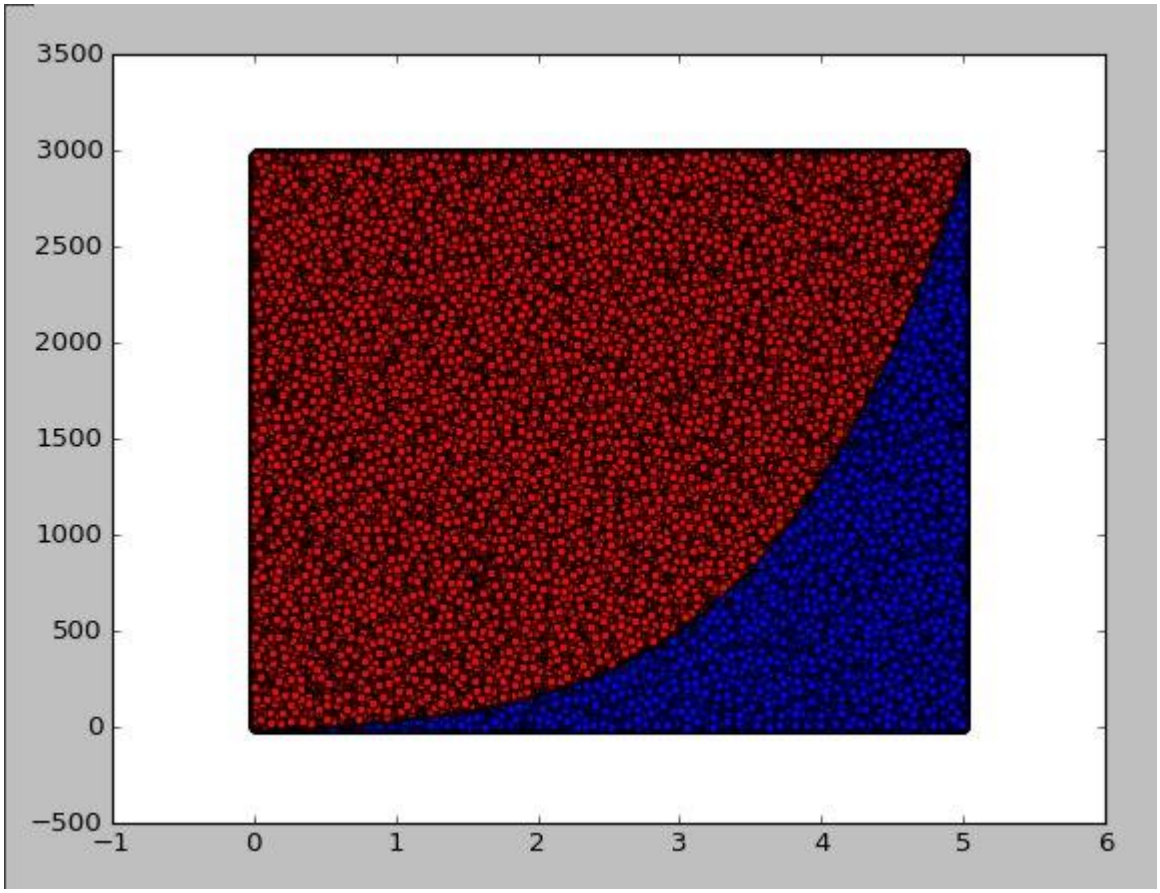
The transport authorities could reduce the length of the red light event when the queue seems to be at its longest for each road respectively.

Question 3a) - Compare your answer to the area achieved analytically.

The code used to run the Monte Carlo Simulation can be seen below with the following results.

```
import random
import matplotlib.pyplot as plt
import numpy as np
import scipy.integrate as integrate

N_iter = 1000000
#width = 10
#radius = width / 2

#CENTER = [width / 2, width / 2]
rangey = 5*(5**4)-8.7*(5**3)+33*(5**2)+21*5+10.8
actual_area = integrate.quad(lambda x: 5*(x**4)-
8.7*(x**3)+33*(x**2)+21*x+10.8, 1, 5)
#print(actual_area)
areaVal = actual_area[0]
print("Actual Area: ", areaVal)


def under_curve(point):
    x = point[0]
    y = point[1]
    #center_x = CENTER[0]
    #center_y = CENTER[1]
    return y < (5*x**4)-(8.7*x**3)+(33*x**2)+(21*x)+10.8


points_under_curve = []
points_out_curve = []

count = count_in = 0

for i in range(N_iter):

    point = [5 * random.random(), rangey * random.random()]

    if under_curve(point):
        points_under_curve.append(point)
        count_in += 1
    else:
        points_out_curve.append(point)

    count += 1

bounds_area = 5*rangey
area = (count_in/count)*bounds_area

print("Area: ", area)

np_under_curve = np.array(points_under_curve)
np_out_curve = np.array(points_out_curve)
```

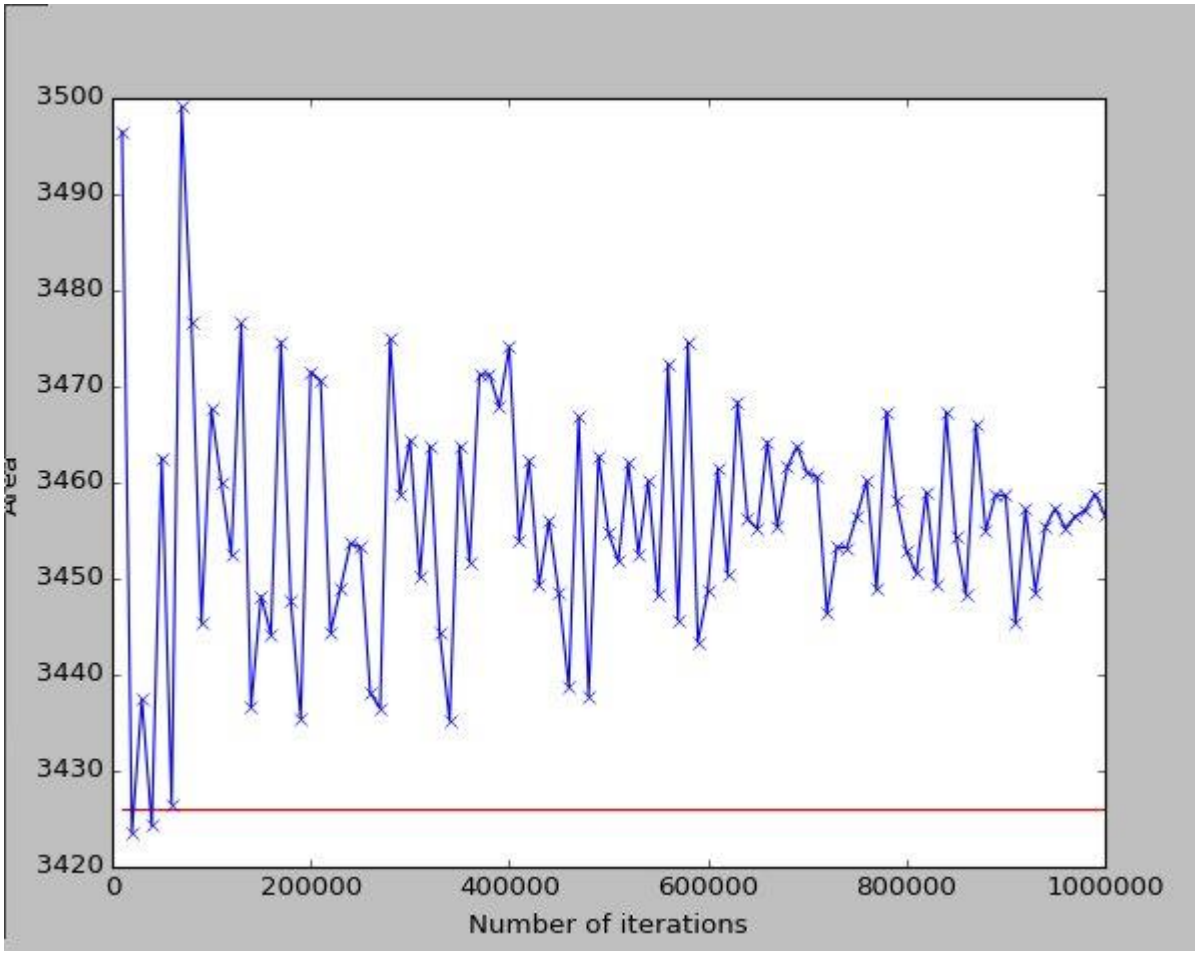
```
plt.figure()
plt.scatter(np_under_curve[:, 0], np_under_curve[:, 1], c='b')
plt.scatter(np_out_curve[:, 0], np_out_curve[:, 1], c='r')
plt.show()


area_bins = []
actual_area_arr = []
area_values = []
x = 10000
step = 10000

N_iters = []

while x <= 1000000:
    N_iters.append(x)
    print(x)
    count = count_in = 0

    for i in range(x):

        point = [5 * random.random(), rangey * random.random()]

        if under_curve(point):
            count_in += 1

        count += 1

    bounds_area = 5 * rangey
    area = (count_in / count) * bounds_area
    area_bins.append([x])
    actual_area_arr.append(areaVal)
    area_values.append(area)

    x = x+step


plt.figure()
plt.plot(area_bins,actual_area_arr, '-r')
plt.plot(N_iters, area_values, '-bx')
plt.xlabel("Number of iterations")
plt.ylabel("Area")
plt.show()
```



Actual area: 3426
Area achieved: 3449.11

As one can note, unfortunately the area achieved in this experiment was not entirely accurate. This must be due to some errors occurring in the experiment.

Question 3b) - Run the simulation for increasing number of iterations in steps of 10,000 up to 1,000,000 and plot the area of the curve achieved through MC simulation versus number of iterations. Superimpose a constant line (in red colour) representing the area achieved analytically. What is the minimum number of iterations required to achieve a value for the area which beyond which it remains within 1% of the analytical value?



Due to the error mentioned in question 3a, unfortunately, it seems that the plot hasn't converged to the actual area. However, assuming that the actual area was in fact 3449.11, then the minimum number of iterations required would be around 9000.