# Simulation

## 1. Problem 1

(a) The true value of $\int_a^b \exp(-x^2/2)dx$:

a = 0, b = 1: 0.85562

a = 0, b = 4: 1.25323

(b) Estimations of the integral:

a = 0, b = 1: n = 20, 0.8425.

n = 200, 0.8510.

n = 2000, 0.8532.

a = 0, b = 4: n = 20, 1.2600.

n = 200, 1.2350.

n = 2000, 1.2485.

(c) Comments:

The results above are the averages of 20 times runs. In each setting, the accuracy of estimation increases as n increases. With a larger n, the points generated are more likely to be equally spaced. The differences between estimation accuracies of the integrals for the two combinations with the same n seem not significant.

## 2. Problem 2

1. Codes:

```
import random
import numpy as np
from random import expovariate
import pandas as pd
import matplotlib.pyplot as plt


def simulation(time, rate, mean, k):
    arrival_time = 0
    arrival_time_list = [0]
    arrival_headway_list = [0]
```

```
    service_time = (mean-0.25)+ round(np.random.random()*0.5, 2) # generate the first
service time
    service_time_list = [service_time]
    departure_time = service_time
    departure_time_list = [departure_time]
    delay_time = service_time
    delay_time_list = [delay_time]
    for i in range(0, 1000):
        if arrival_time <time:
            arrival_headway = round(random.expovariate(rate), 2) # generate arrival time
interval
            arrival_headway_list.append(arrival_headway)
            arrival_time += arrival_headway
            arrival_time_list.append(arrival_time)
            service_time = (mean-0.25)+ round(np.random.random()*0.5,2) # generate
service time
            service_time_list.append(service_time)
            if arrival_time>= departure_time: # determine the departure time
                departure_time = arrival_time + service_time
                delay_time = service_time
                departure_time_list.append(departure_time)
                delay_time_list.append(delay_time)
            else:
                departure_time = departure_time + service_time
                delay_time = departure_time - arrival_time
                departure_time_list.append(departure_time)
                delay_time_list.append(delay_time)
    index = np.arange(len(arrival_headway_list))+1
    departure_list = []
    for i in range(0, len(arrival_headway_list)):
        if departure_time_list[i] <1000:
```

```
            departure_list.append(departure_time_list[i])
            n = i
    index_2 = np.arange(n+1)
    ar, = plt.plot(arrival_time_list, index)
    de, = plt.plot(departure_list, index_2)
    plt.title("Arrivals and Departures")
    plt.legend([ar, de], ["Arrivals", "Departures"], loc = 2)
    plt.show()
    average_delay = (np.array(delay_time_list)).sum()/len(delay_time_list)
    output = pd.DataFrame({'CUSTOMER':index,'ARRIVAL
HEADWAY':arrival_headway_list, 'ARRIVAL TIME':arrival_time_list, 'SERVICE
TIME':service_time_list, 'DEPARTURE TIME':departure_time_list, 'DELAY TIME':
delay_time_list})
    output.to_csv('Output'+ str(k+1)+ '.txt',"\t", index = False, header = True,cols =
["CUSTOMER", "ARRIVAL HEADWAY", "ARRIVAL TIME", "SERVICE TIME",
"DEPARTURE TIME", "DELAY TIME"] )
    return (average_delay, len(arrival_time_list), delay_time_list)

if __name__ == '__main__':
    total_delay = 0
    total_times = 0
    delay_list = []
    all_list = []
    n = 1 # times of simulation
    time = 1000 # time period of the simulation
    rate = 0.495 # arrival rate
    mean = 2 # service time mean
    for i in range(0, n):
        delay, times, list = simulation( time, rate, mean, i)
        total_delay += delay*times
        total_times += times
```

```
        delay_list.append(delay)
        all_list += list
    average_delay = total_delay/total_times
    std = np.std(np.array(all_list))
    print "Average of all delays: " + str(average_delay)
    print "Stardard deviation of all delays: "+ str(std)
```

The output is a flow chart that saved as “Outputn.txt”, and n indicates in which number of simulation the output is made.

2. The result of one-time run: total average delay for a 1000-second simulation is 32.139.

3. The mini delays for 100-second periods are 2.09, 24.06, 17.18, 21.81, 21.11, 20.35, 16.79, 12.5, 27.18, and 50.87.

4. The standard deviation of delay is 13.835.

5. Arrivals and departures plot of the 1000-second simulation:

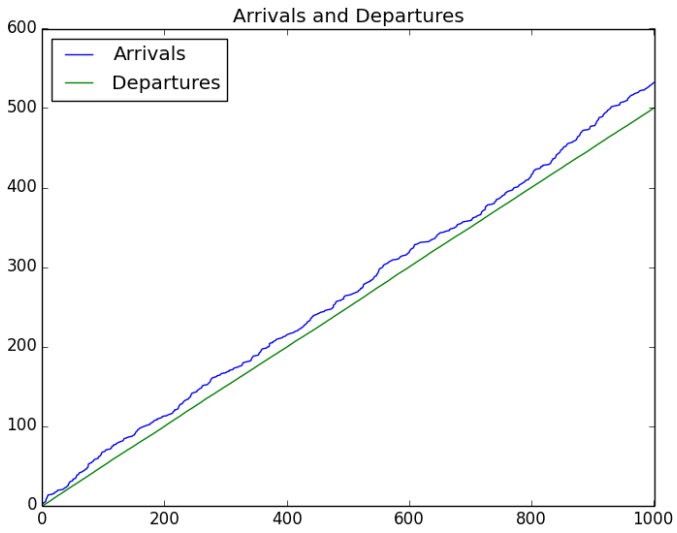


Figure 1, Arrivals and departures versus time

6. The result of one-time run: total average delay for a 1000-second simulation is 174.954, the mini delays for 100-second periods are 1.8, 32.41, 67.98, 118.31, 141.67, 172.69, 202.18, 238.64, 287.26 and 307.03. The standard deviation of delay is 99.304.

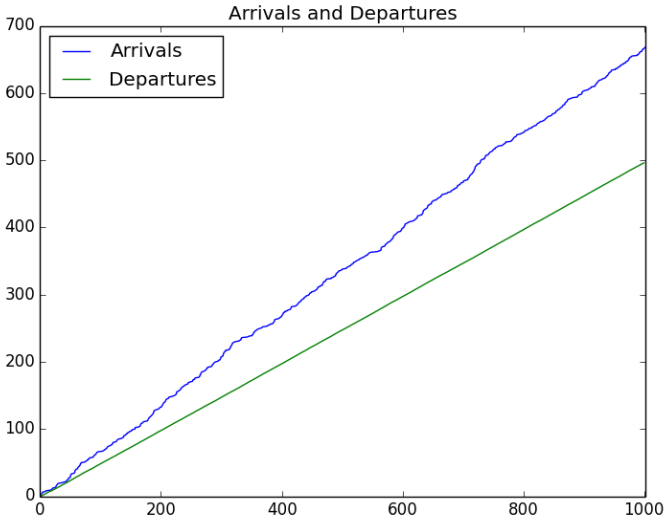


Figure 2, Arrivals and departures versus time

7. The result of one-time run: total average delay for a 1000-second simulation is 1.597, the mini delays for 100-second periods are 0.78, 0.78, 0.77, 0.75, 0.87, 0.77, 0.76, 0.84, 0.75 and 0.77. The standard deviation of delay is 0.832.

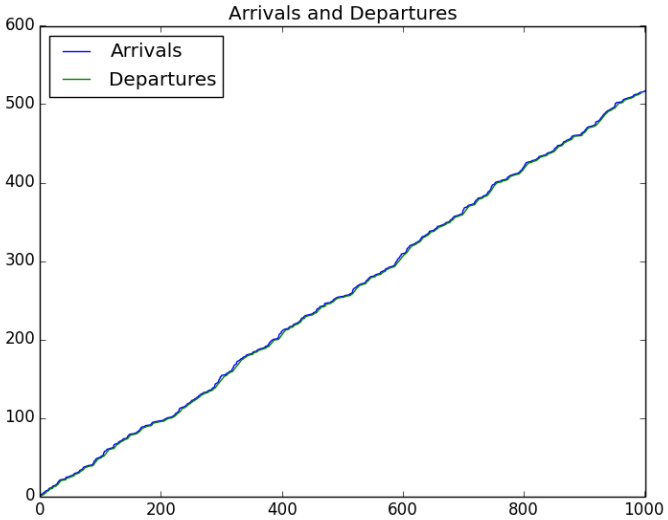


Figure 3, Arrivals and departures versus time

8. To compare the three different scenarios, the calculation results of the scenarios are listed in the table:

Table 1, Comparisons between three scenarios

| Scenario One | | Scenario Two | | Scenario Three | |
|---|---|---|---|---|---|
| Arrival rate | 0.495 | Arrival rate | 0.7 | Arrival rate | 0.495 |
| Service time mean | 2 | Service time mean | 2 | Service time mean | 1 |
| 1000-second period | | 1000-second period | | 1000-second period | |
| Average delay | 32.139 | Average delay | 174.954 | Average delay | 1.597 |
| Standard derivation | 13.835 | Standard derivation | 99.304 | Standard derivation | 0.832 |
| 100-second periods | | 100-second periods | | 100-second periods | |
| Average mini delay | 21.394 | Average mini delay | 156.997 | Average mini delay | 0.784 |
| Mini delay | 2.09 | Mini delay | 1.8 | Mini delay | 0.75 |

It's observed that the average delay, standard derivation, average mini delay and mini delay decrease significantly as the service time mean decreases from 2 to 1; the average delay, standard derivation and average mini delay increase significantly as the arrival rate increases from 0.495 to 0.7, and the mini delay doesn't change much.

9. Run 10000 times 1000-second period simulations in scenario one, and use k = 1 + 3.322*log10(n) to determine the number of bins. The results of the simulations are different, because the arrival and service times of each customer, which generated by the program, are random. The summary statistics:

Table 2, Summary statistics for average delay

| Summary statistics | Value |
|---|---|
| Minimum | 4.983 |
| Maximum | 99.233 |
| Mean | 22.763 |
| Median | 19.273 |
| Variance | 157.464 |
| Coefficient of variation | 0.551 |
| Skewness | 1.367 |

According to the calculations, choose Gamma and Weibull distributions for distribution fits in R:

Gamma fit (shape= 3.86053449 , rate = 0.16959953):

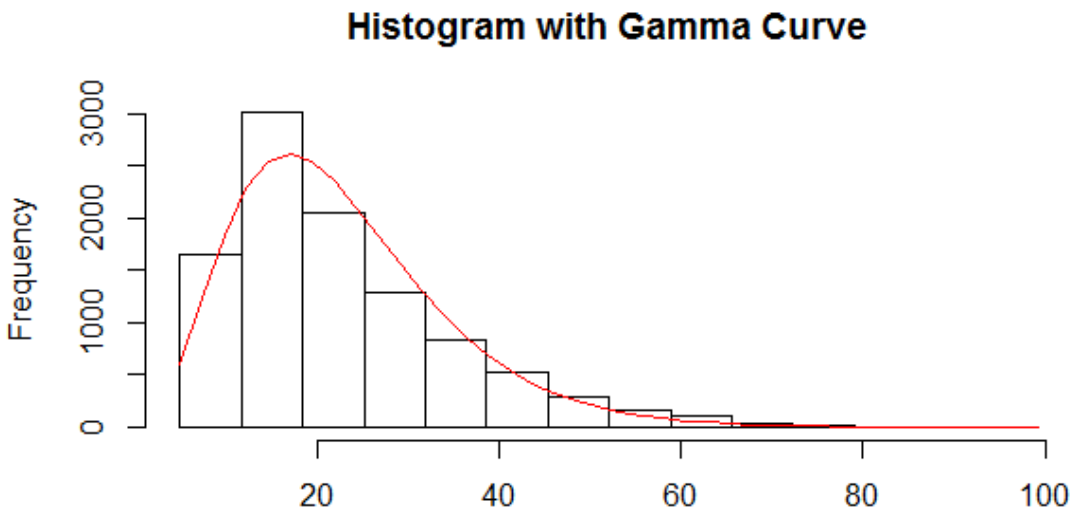


Figure 5, Histogram and Gamma Curve

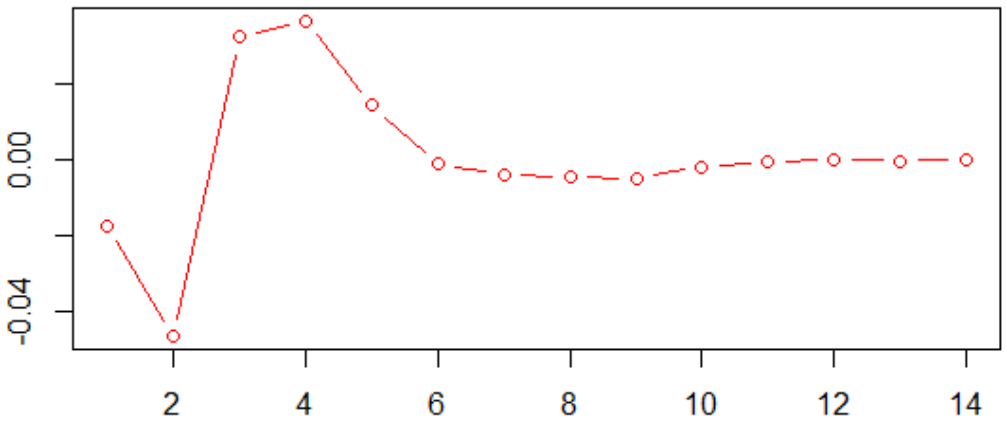


Figure 6, Gamma distribution function difference

The differences are less than 0.1 at each point.

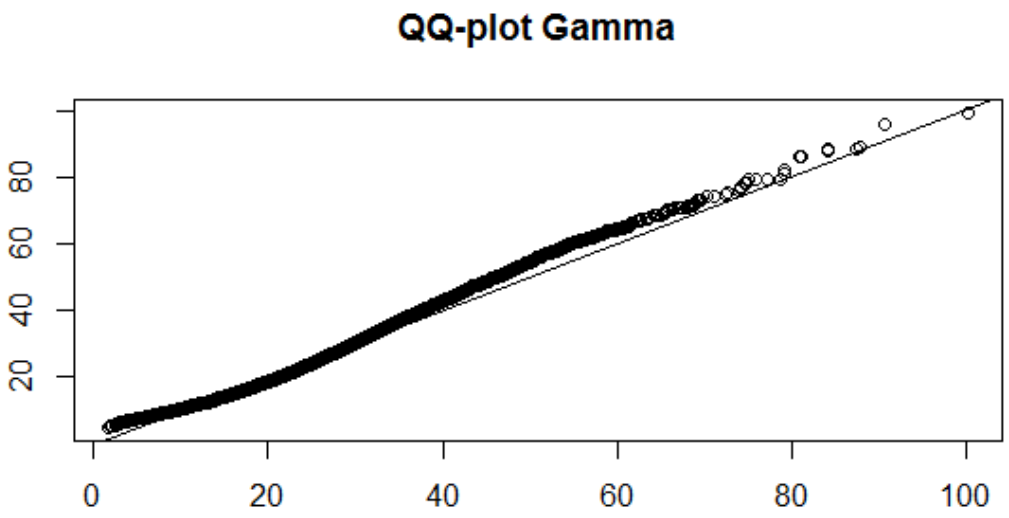


Figure 7, QQ plot of Gamma fit

Gamma fit K-S test result, $D_n = 0.1895$, p-value = 0.366.

Weibull fit (shape= 1.95160124, scale = 25.83152661):

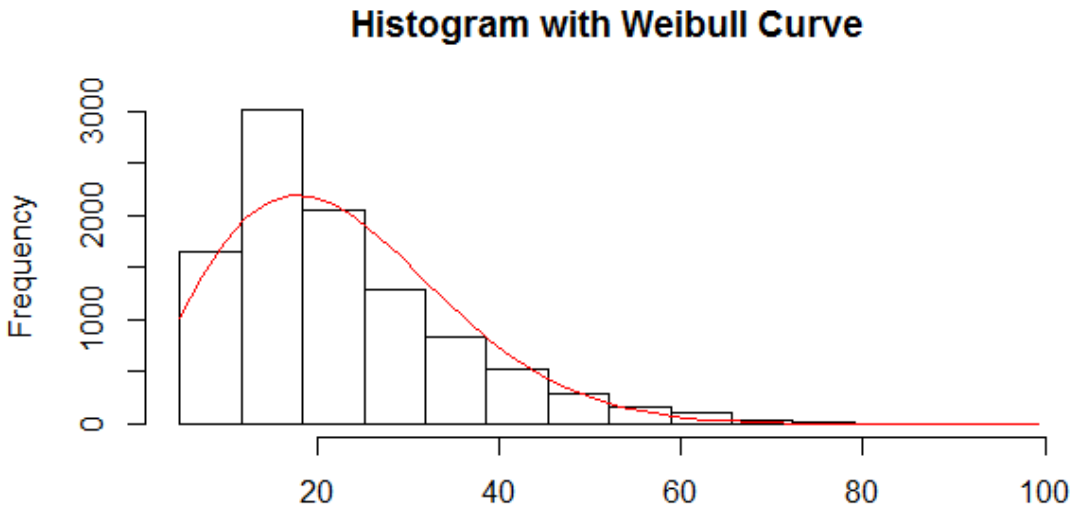


Figure 8, Histogram and Weibull Curve

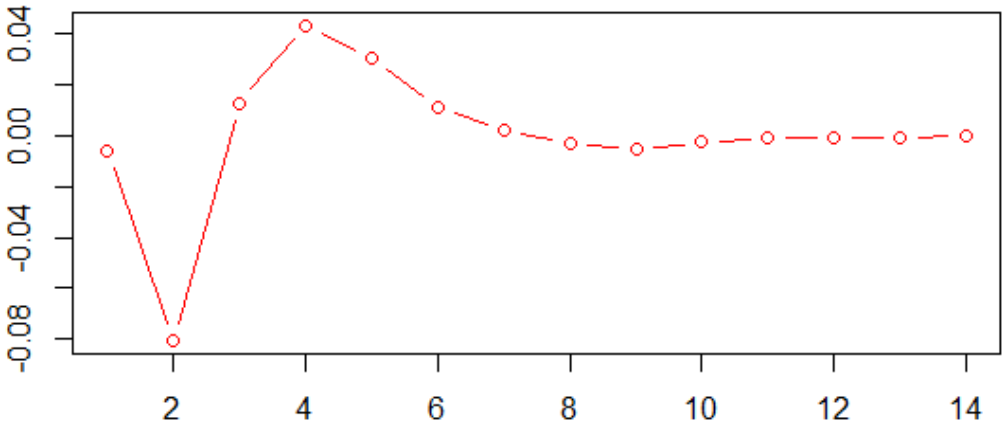


Figure 9, Weibull distribution function difference

The differences are also less than 0.1 at each point, but trend to be much larger than the Gamma fit at the second point.

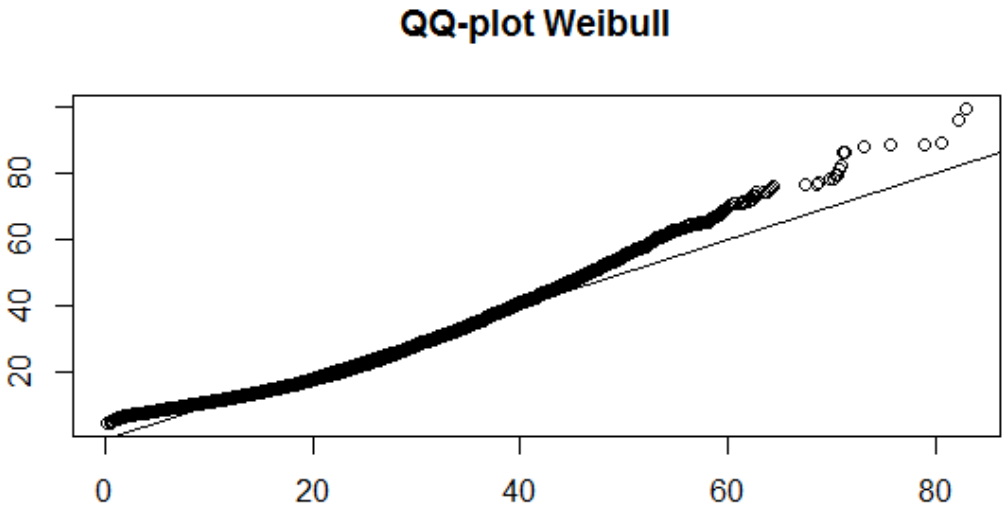


Figure 10, QQ plot of Weibull fit

Weibull fit K-S test result, $D_n = 0.158$, p-value = 0.497. Since Weibull distribution fit has a larger p-value, we choose Weibull distribution with shape= 1.95160124 and scale = 25.83152661 as the probability distribution of average delay.

The average of all delays from the simulation is 22.763.

10. To get a reduction in average delay, we can reduce arrival rate or service mean time or do them both. Take 1000-second period simulation in scenario one for example, the aim is to reduce the average delay of 10000 times runs from 22.763 to around 17.07:

Reduce arrival rate, but keep the service mean time = 2: when rate = 0.481, the average delay is around 17.07;

Reduce service mean time, but keep the arrival rate = 0.495: when service time = 1.947, the average delay is around 17.07.

The combinations of these two ways could also work. It seems that the average delay is less sensitive with arrival rate than service mean time, since 2.83% reduction in arrival rate could make 25% reduction in average delay, while 2.65% reduction in service mean time has the same effect on average delay.

If there's more time, we can run service mean time or arrival rate through a range like [1, 3] or [0.4, 0.6] with certain spacing such as 0.001, collect corresponding average delays, use n-fold cross-validations to find polynomial orders, then build OLS models, thus a more systemic prediction of average delay can be achieved.