Aaron Fox
CECS 622
Dr. Elmaghraby

Homework 4

Part 1: Chi- Squared Test

The following 20 numbers are generated using the Excel uniformly distributed over the interval 0 to 1.

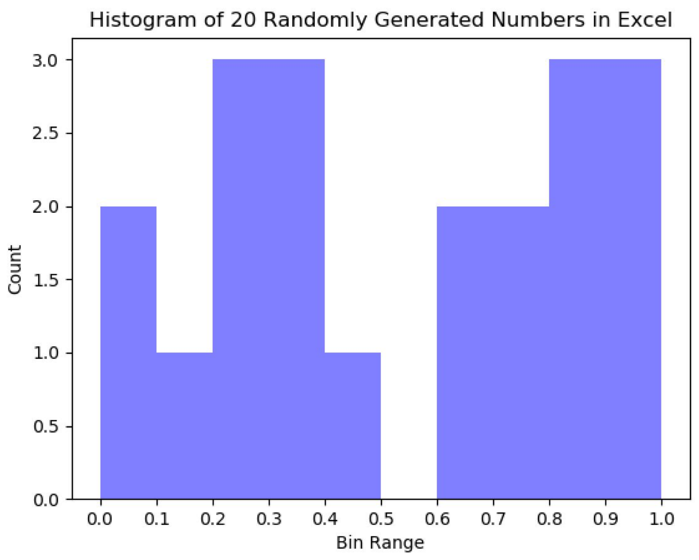
0.21	0.88	0.37	0.06	0.98	0.61	0.89	0.28	0.70	0.94
0.46	0.92	0.34	0.08	0.79	0.82	0.36	0.62	0.27	0.10

Random Numbers

Compute the histogram over the 10 subintervals: [0, 0.1), [0.1, 0.2) … [0.9, 1.0).
The following code in Python was used to compute the histogram over the specified intervals:

```python
# For plotting
import matplotlib.pyplot as plt

### Part 1: Chi-Squared Test ###

## Compute the histogram over the 10 subintervals: [0, 0.1), [0.1, 0.2) … [0.9, 1.0). ##
random_numbers = [0.21, 0.88, 0.37, 0.06, 0.98, 0.61, 0.89, 0.28, 0.70, 0.94, 0.46, 0.92,
0.34, 0.08, 0.79, 0.82, 0.36, 0.62, 0.27, 0.10]
bins_list = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
n, bins, patches = plt.hist(random_numbers, bins=bins_list, facecolor='blue', alpha=0.5)
plt.title("Histogram of 20 Randomly Generated Numbers in Excel")
plt.xticks(ticks=[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
plt.xlabel("Bin Range")
plt.ylabel("Count")
plt.show()
```

The variable `bins_list` is specified so that the code is inclusive of the first number and exclusive of the second number (i.e. like [0.0, 0.1), creating 10 bins). The histogram looks like:



Histogram of data as given in the homework prompt

The data was verified by hand after a hand-drawn chart was also drawn.

Perform the Chi-square Goodness-of-fit test with alpha = 0.05.

The following code and comments were written out to answer the problem:

```
## Perform the Chi-square Goodness-of-fit test with alpha = 0.05 ##
# Null hypothesis is that the data is consistent with a random uniform distribution (e.g. every bin
should have a 10% likelihood)
# Bin:            |   1  |   2  |   3  |   4  |   5  |   6  |   7  |   8  |   9  |  10  |
# Expected %:     |  10  |  10  |  10  |  10  |  10  |  10  |  10  |  10  |  10  |  10  |
# Observed Count:|   2  |   1  |   3  |   3  |   1  |   0  |   2  |   2  |   3  |   3  |
# Expected Count:|   2  |   2  |   2  |   2  |   2  |   2  |   2  |   2  |   2  |   2  |
# With alpha = 0.05
expected_counts = [2] * 10
observed_counts = [2, 1, 3, 3, 1, 0, 2, 2, 3, 3]

chi_squared_stat = 0.0
for i in range(len(observed_counts)):
    expected_count = expected_counts[i]
    observed_count = observed_counts[i]
    chi_squared_stat = chi_squared_stat + ((observed_count - expected_count) ** 2) / expected_count

print("For this random sampling in Excel , chi-squared statistic == " + str(chi_squared_stat))

# With N Categories - 1 = 9 degrees of freedom and alpha = 0.05,
# using chart found in Appendix,
# the probability of exceeding the critical value is 16.919
# Since chi-squared value = 5.0 < 16.919, we can ACCEPT the
# null hypothesis that the generated data follows
# the uniform distribution specified by Excel. We can therefore reject
# the alternate hypothesis that the generated data in Excel does not follow
# the random uniform distribution specified by Excel.
```

To summarize the conclusion above: first, a null hypothesis was created that states the 20 randomly generated numbers in Excel indeed followed the random uniform distribution. The alternate hypothesis stated that the 20 randomly generated numbers in Excel *did not* follow the random uniform distribution specified by Excel.

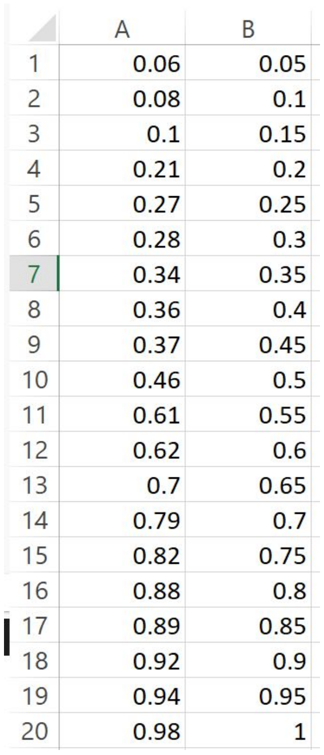
To test the null hypothesis, an array of the observed counts was counted by counting the amount in each bin of the histogram above. The expected counts were 10% for each bin since there were 10 bins, or 20 * 0.1 = 2 for every bin. The expected count of each bin was thus 2. The chi-squared statistic was then taken for every observed and expected count by taking the sum of every $\frac{(Observed\ Count_i - Expected\ Count_i)^2}{Expected\ Count_i}$ in the supplied arrays.

The calculated sum of the chi-squared statistic came out to be 5.0. After consulting the supplied Critical Values of the Chi-square Distribution with *d* Degrees Freedom supplied in the Appendix of the homework, it was found that the value having an alpha of 0.05 and a degrees of freedom of 9 (10 categorical bins - 1 = 9) was 16.919. Since the chi-square value of 5.0 is less than the critical chi-squared 16.919, we can therefore **accept** the null hypothesis and say that the distribution of data generated by Excel is indeed uniformly random.

For extra credit*: Perform the Kolmogrov-Smirnov Goodness-of-fit test with alpha = 0.05*

First, a null hypothesis was created, stating that the random data from Excel comes from a random uniform distribution. The alternative hypothesis states that the random data from Excel does *not* come from a random uniform distribution.

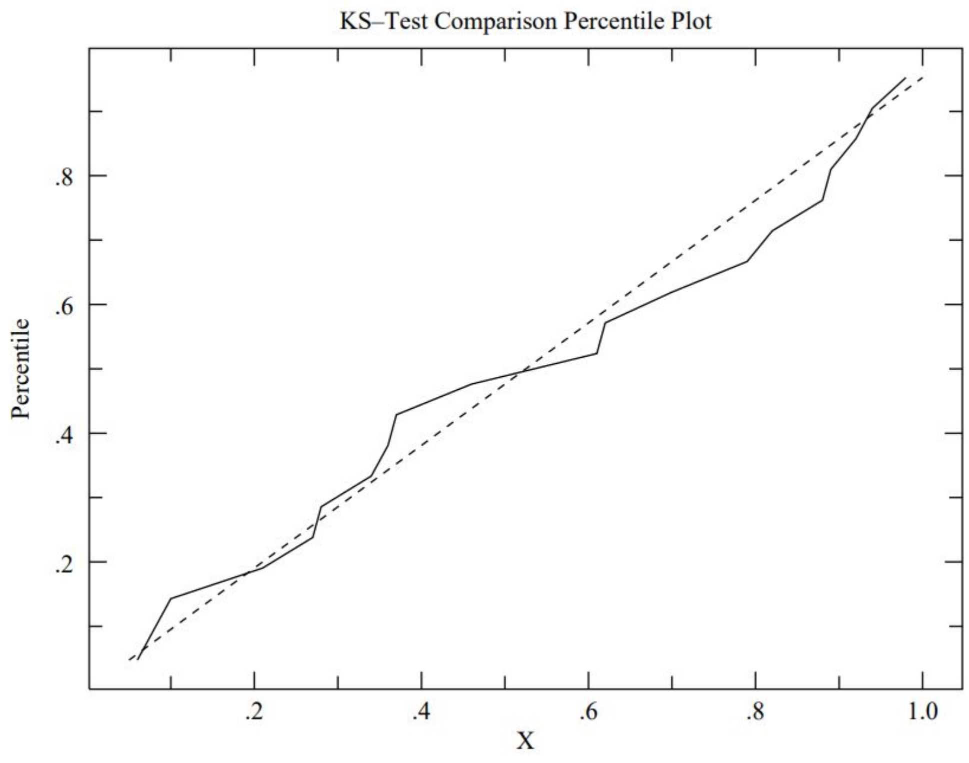
Next, an Empirical Distribution Function (EDF) was generated in Excel by entering all the data points in one column and then sorting them from smallest to largest.The Empirical data was then gather by using the Excel formula =ROW(B1) / 20 for the next columns while changing the respective row to show the empirical distribution. The Excel data looks like:



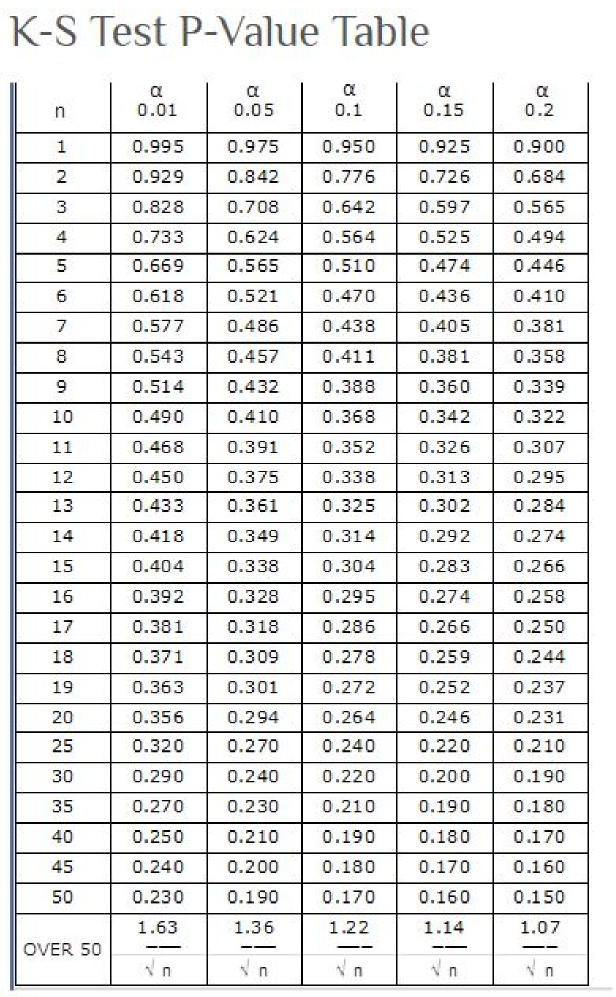
	A	B
1	0.06	0.05
2	0.08	0.1
3	0.1	0.15
4	0.21	0.2
5	0.27	0.25
6	0.28	0.3
7	0.34	0.35
8	0.36	0.4
9	0.37	0.45
10	0.46	0.5
11	0.61	0.55
12	0.62	0.6
13	0.7	0.65
14	0.79	0.7
15	0.82	0.75
16	0.88	0.8
17	0.89	0.85
18	0.92	0.9
19	0.94	0.95
20	0.98	1

An API developed at College of Saint Benedict at St. John's University was then used with both columns of data to evaluate the Kolmogrov-Smirnov Goodness-of-fit test with alpha = 0.05 [1].

The greatest vertical difference between the two graphs was found to be 0.1000. It can be seen in the graph generated by the API below:



The following K-S Test P-Value Table was used :



K-S Test P-Value Table

n	α 0.01	α 0.05	α 0.1	α 0.15	α 0.2
1	0.995	0.975	0.950	0.925	0.900
2	0.929	0.842	0.776	0.726	0.684
3	0.828	0.708	0.642	0.597	0.565
4	0.733	0.624	0.564	0.525	0.494
5	0.669	0.565	0.510	0.474	0.446
6	0.618	0.521	0.470	0.436	0.410
7	0.577	0.486	0.438	0.405	0.381
8	0.543	0.457	0.411	0.381	0.358
9	0.514	0.432	0.388	0.360	0.339
10	0.490	0.410	0.368	0.342	0.322
11	0.468	0.391	0.352	0.326	0.307
12	0.450	0.375	0.338	0.313	0.295
13	0.433	0.361	0.325	0.302	0.284
14	0.418	0.349	0.314	0.292	0.274
15	0.404	0.338	0.304	0.283	0.266
16	0.392	0.328	0.295	0.274	0.258
17	0.381	0.318	0.286	0.266	0.250
18	0.371	0.309	0.278	0.259	0.244
19	0.363	0.301	0.272	0.252	0.237
20	0.356	0.294	0.264	0.246	0.231
25	0.320	0.270	0.240	0.220	0.210
30	0.290	0.240	0.220	0.200	0.190
35	0.270	0.230	0.210	0.190	0.180
40	0.250	0.210	0.190	0.180	0.170
45	0.240	0.200	0.180	0.170	0.160
50	0.230	0.190	0.170	0.160	0.150
OVER 50	$\frac{1.63}{\sqrt{n}}$	$\frac{1.36}{\sqrt{n}}$	$\frac{1.22}{\sqrt{n}}$	$\frac{1.14}{\sqrt{n}}$	$\frac{1.07}{\sqrt{n}}$

Using the table above with an n of 20 and an alpha of 0.05, the P-value was determined to be 0.294. Since the maximum vertical distance of 0.1000 is less than 0.294, we can **accept** the null hypothesis and state that the random data from Excel comes from a random uniform distribution.

b. The grade distribution from two semesters is given below

	This Year	Last year
a	20	10
b	22	19
c	13	25
d	2	4
f	2	1

Compare the grades using Chi-squared test to decide whether the overall distributions are statistically different or not.

In testing the homogeneity of the distribution from last year and this year, the chi-square test can also be used. Here, the null hypothesis can be said as

H_0: The overall distributions of this year and last year follow the same distribution and are *not* statistically different.

and the alternate hypothesis can be listed as

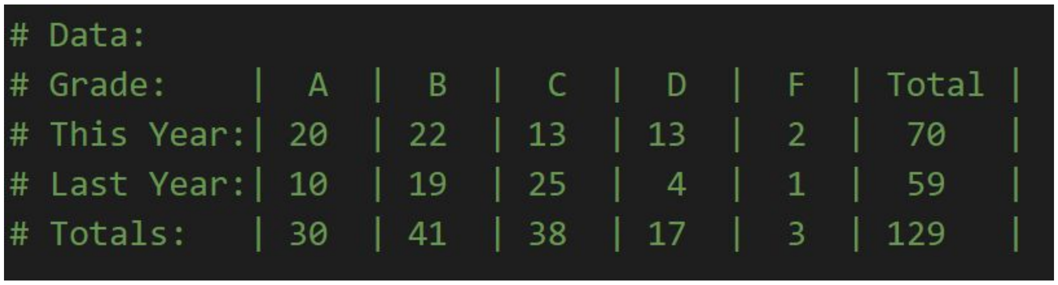
H_1: The overall distributions of this year and last year *do not* follow the same distribution and are statistically different.

The following code was used to test out the problem:

```
## Part b:  Compare the grades using Chi-squared test to decide whether the overall
## distributions are statistically different or not. ##

# Data:
# Grade:    |  A  |  B  |  C  |  D  |  F  | Total |
# This Year:| 20  | 22  | 13  | 13  |  2  |  70   |
# Last Year:| 10  | 19  | 25  |  4  |  1  |  59   |
# Totals:   | 30  | 41  | 38  | 17  |  3  | 129   |

# Example calculation for This Year of A:
# 20/70 * 100 = 28.571...
# Data as a Percentage:
# Grade:    |   A    |   B    |   C    |   D    |   F    |  Total  |
# This Year:| 28.571 | 31.429 | 18.571 | 18.571 |  2.857 |  100.0  |
# Last Year:| 16.949 | 32.203 | 42.373 |  6.780 |  1.695 |  100.0  |
# Totals:   | 23.256 | 31.783 | 29.457 | 13.178 |  2.326 |  100.0  |

# Example calculation for this year of A
# 0.23256 * 70 = 16.279 (percentage of total for both distributions of category * total for year)
# Example calculation for this year of A
# 0.23256 * 59 = 13.721
# Expected Counts:
# Grade:    |   A    | B      |   C    |   D    |    F   | Total |
# This Year:| 16.279 | 22.248 | 20.620 |  9.225 |  1.628 |  70   |
# Last Year:| 13.721 | 18.752 | 17.380 |  7.775 |  1.372 |  59   |
# Totals:   |   30   |   41   |   38   |   17   |    3   | 129   |
# With alpha = 0.05
# Place expected counts into array, starting from left to right of each line starting with This
Year
expected_counts = [16.279, 22.248, 20.620, 9.225, 1.628, 13.721, 18.752, 17.380, 7.775, 1.373]
observed_counts = [20, 22, 13, 13, 2, 10, 19, 25, 4, 1]
# Data:
# Grade:    |  A  |  B  |  C  |  D  |  F  | Total |
# This Year:| 20  | 22  | 13  | 13  |  2  |  70   |
# Last Year:| 10  | 19  | 25  |  4  |  1  |  59   |
# Totals:   | 30  | 41  | 38  | 17  |  3  | 129   |

chi_squared_stat = 0.0
```

```
for i in range(len(observed_counts)):
    expected_count = expected_counts[i]
    observed_count = observed_counts[i]
    chi_squared_stat = chi_squared_stat + ((observed_count - expected_count) ** 2) /
expected_count

print("Part 1.b: For this random sampling of grades, chi-squared statistic == " +
str(chi_squared_stat))

# With N Categories - 1 = 4 degrees of freedom and a
# chi-squared test statistic value = 11.586 > 9.488, we can determine that
# the p-value of the probability is 0.05 using the calculator for P-value found at
# https://stattrek.com/online-calculator/chi-square.aspx. Since a p-value less
# than or equal to 0.05 is statistically significant, it shows that there is strong
# evidence AGAINST the null hypothesis, indicating that there is less than a 5% chance that null
# hypothesis is correct. As such, we can REJECT the
# null hypothesis that the overall distributions of this year and last year follow
# the same distribution and are not statistically different. We can therefore ACCEPT
# the alternate hypothesis that the overall distributions of this year and last year
# do not follow the same distribution and are statistically different.
```

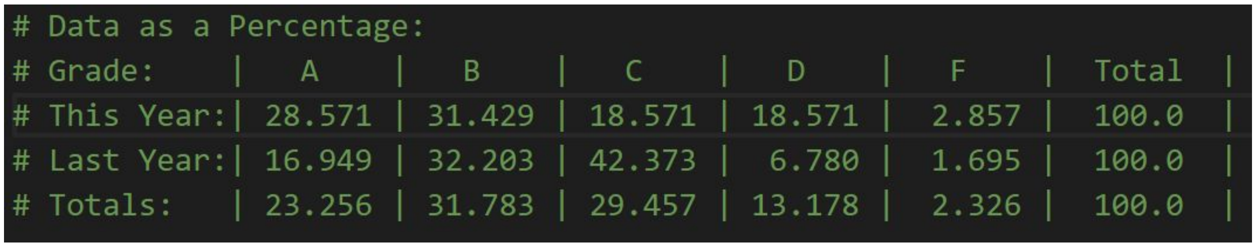
The data was transformed into the following tables:



```
# Data:
# Grade:      |  A  |  B  |  C  |  D  |  F  | Total |
# This Year:| 20  | 22  | 13  | 13  |  2  |  70   |
# Last Year:| 10  | 19  | 25  |  4  |  1  |  59   |
# Totals:     | 30  | 41  | 38  | 17  |  3  | 129   |
```

The original data with totals added

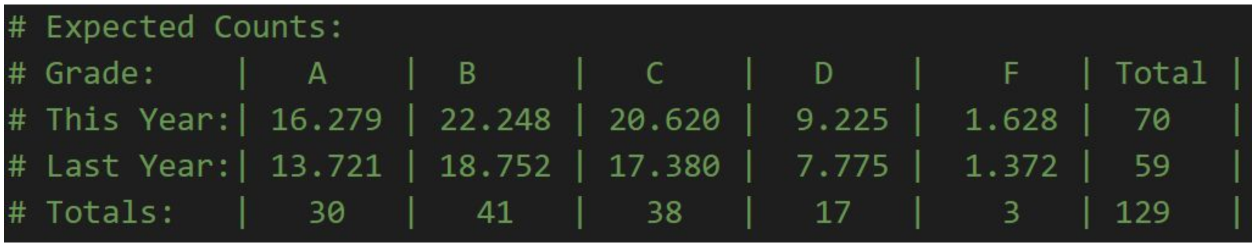
The original data was then transformed into a percentage of the total amount of students that took the course that year, with an example for the calculation for the students of “This Year” receiving an A being 20 / 70 * 100 = 28.571%.



```
# Data as a Percentage:
# Grade:      |   A    |   B    |   C    |   D    |   F   |  Total  |
# This Year:| 28.571 | 31.429 | 18.571 | 18.571 |  2.857 |  100.0  |
# Last Year:| 16.949 | 32.203 | 42.373 |  6.780 |  1.695 |  100.0  |
# Totals:     | 23.256 | 31.783 | 29.457 | 13.178 |  2.326 |  100.0  |
```

The data written as a percentage of the total

The expected counts of each year and the number of students who received each grade was then calculated. For example, the expected count of students who received an A in “This Year” can be calculated by multiplying the percentage of the total for both distributions of the grade category by the total number of students that took that class during that year, or 0.23256 * 70 = 16.279%.



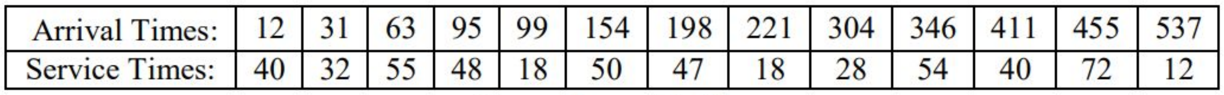
```
# Expected Counts:
# Grade:     |   A    |   B    |   C    |   D    |   F    | Total |
# This Year:| 16.279 | 22.248 | 20.620 |  9.225 |  1.628 |  70   |
# Last Year:| 13.721 | 18.752 | 17.380 |  7.775 |  1.372 |  59   |
# Totals:    |   30   |   41   |   38   |   17   |   3    | 129   |
```

The expected counts of each student receiving that grade each year

A chi squared statistic was then taken into account for the data by inserting the data into their respective arrays, `expected_counts` and `observed_counts`, the structure of which can be seen above. The calculated chi-squared statistic using the code above turned out to be 11.586. The p-value was then calculated using the p-value calculator found Stat Trek.com [2]. The p-value was determined to be 0.05. Since a p-value less than or equal to 0.05 is statistically significant, it shows that there is strong evidence AGAINST the null hypothesis, indicating that there is less than a 5% chance that null hypothesis is correct. As such, we can REJECT the null hypothesis that the overall distributions of this year and last year follow the same distribution and are not statistically different. We can therefore ACCEPT the alternate hypothesis that the overall distributions of this year and last year do not follow the same distribution and are statistically different.

Part 2: Queue Analysis

The following data provides the arrival times and service times that each customer will require, for the first 13 customers at a single server system. Upon arrival, a customer either enters service if the server is free or joins the waiting line. When the server completes work on a customer, the next one in line (i.e. the one who has been waiting the longest) enters service. Assume that the system is clear at time 0 and customer #13 is the last customer to be serviced.



Arrival Times:	12	31	63	95	99	154	198	221	304	346	411	455	537
Service Times:	40	32	55	48	18	50	47	18	28	54	40	72	12

Queuing Data

a) Determine the system departure and response times of these 13 customers

The following program was written to answer the system departure and response times of the 13 customers:

```
    arrival_times = [12, 31, 63, 95, 99, 154, 198, 221, 304, 346, 411, 455, 537]
    service_times = [40, 32, 55, 48, 18, 50, 47, 18, 28, 54, 40, 72, 12]
    # arrival_times = [2, 4, 7]
    # service_times = [3, 1, 4]
    # The departure times are the time in which the customer is completely serviced
    system_departure_times = []
    # The response time is the total amount of time a customer spends in both the queue and in
service.
    system_response_times = []

    time = 0
    customers_serviced = 0
    customers_arrived = 0
    queue = []
    customer_just_arrived = False  # customer_just_arrived ensure that a customer isn't serviced
in the same minute they arrive
    while customers_serviced != len(arrival_times):
        # Check if new arrival of customer should be added to queue
        # print("Time == " + str(time))
        if customers_arrived != len(arrival_times) and time == arrival_times[customers_arrived]:
            print("Appending to queue at t = " + str(time))
            queue.append(service_times[customers_arrived])
            customer_just_arrived = True
            customers_arrived = customers_arrived + 1
        elif customer_just_arrived:
            customer_just_arrived = False

        if len(queue) >= 1 and not (len(queue) == 1 and customer_just_arrived):
            queue[0] = queue[0] - 1

            # Check if first customer in queue is finished servicing
            if queue[0] == 0:
                print("Popping customer from queue at t = " + str(time))
                # Delete fully serviced customer from queue since it's done being serviced
                del queue[0]
                # Add data of customer to respective system response and departure times
                system_departure_times.append(time)
                system_response_times.append(time - arrival_times[customers_serviced])
                customers_serviced = customers_serviced + 1
        time = time + 1

    print("System Response Times: " + str(system_response_times))
    print("System Departure Times: " + str(system_departure_times))
```

The code yielded the following output:

```
Appending to queue at t = 12
Appending to queue at t = 31
Popping customer from queue at t = 52
Appending to queue at t = 63
Popping customer from queue at t = 84
Appending to queue at t = 95
Appending to queue at t = 99
Popping customer from queue at t = 139
Appending to queue at t = 154
Popping customer from queue at t = 187
```

```
Appending to queue at t = 198
Popping customer from queue at t = 205
Appending to queue at t = 221
Popping customer from queue at t = 255
Popping customer from queue at t = 302
Appending to queue at t = 304
Popping customer from queue at t = 320
Appending to queue at t = 346
Popping customer from queue at t = 348
Popping customer from queue at t = 402
Appending to queue at t = 411
Popping customer from queue at t = 451
Appending to queue at t = 455
Popping customer from queue at t = 527
Appending to queue at t = 537
Popping customer from queue at t = 549
System Response Times: [40, 53, 76, 92, 106, 101, 104, 99, 44, 56, 40, 72, 12]
System Departure Times: [52, 84, 139, 187, 205, 255, 302, 320, 348, 402, 451, 527,
549]
```

When put in a table, the table looks like:

Customer	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Response Time	40	53	76	92	106	101	104	99	44	56	40	72	12
Departure Time	52	84	139	187	205	255	302	320	348	402	451	527	549

Note: These numbers came about due to the assumption that the minute they arrived was not counted in the service time as it was assumed that the server had to prep that minute to add that customer to the queue. Otherwise, the departure and arrival times would **all be subtracted by one** if that assumption were not made.

The core of the program rested in the while loop, which continually ran until every customer had been serviced. If a customer was to be added to the queue at the time specified in `arrival_times`, then the customer's service time was added to the queue. An if statement then checked to see if there was a customer currently in the queue, checking to make sure that the minute to allow the customer to arrive was not counted twice in the departure and arrival times. Otherwise, all the final Arrival and Departure Times would be subtracted by one or more.

An if check was then used to see if the front of the queue was fully serviced; if so, the current time was added to departure time and the response time was added using the formula

```
Response Time = Departure Time - Arrival Time
```

to their respective `arrival_times` and `departure_times` arrays. The time was then incremented after that and the while loop cycled again to run everything again until all customers were serviced.

b) What is the percent server utilization?

The code was adjusted to discover the "wasted minutes" that a server was not actively serving a customer. An if check was added to the code to ensure that a minute was not counted as wasted if a customer was serviced. The edited code used to output the percent server utilization was:

```
    arrival_times = [12, 31, 63, 95, 99, 154, 198, 221, 304, 346, 411, 455, 537]
    service_times = [40, 32, 55, 48, 18, 50, 47, 18, 28, 54, 40, 72, 12]
    # arrival_times = [2, 4, 7]
    # service_times = [3, 1, 4]
    # arrival_times = [0, 2, 3]
    # service_times = [2, 1, 3]
    # The departure times are the time in which the customer is completely serviced
    system_departure_times = []
    # The response time is the total amount of time a customer spends in both the queue and in service.
    system_response_times = []

    time = 0
    customers_serviced = 0
    customers_arrived = 0
    queue = []
    wasted_minutes = 0
    customer_just_arrived = False  # customer_just_arrived ensure that a customer isn't serviced in the same
minute they arrive
    while customers_serviced != len(arrival_times):
        # Check if new arrival of customer should be added to queue
        # print("Time == " + str(time))
        if customers_arrived != len(arrival_times) and time == arrival_times[customers_arrived]:
            # print("Appending to queue at t = " + str(time))
            queue.append(service_times[customers_arrived])
            customer_just_arrived = True
            customers_arrived = customers_arrived + 1
        elif customer_just_arrived:
            customer_just_arrived = False

        if len(queue) >= 1 and not (len(queue) == 1 and customer_just_arrived):
            queue[0] = queue[0] - 1

            # Check if first customer in queue is finished servicing
            if queue[0] == 0:
                # print("Popping customer from queue at t = " + str(time))
                # Delete fully serviced customer from queue since it's done being serviced
                del queue[0]
                # Add data of customer to respective system response and departure times
                system_departure_times.append(time)
                system_response_times.append(time - arrival_times[customers_serviced])
                customers_serviced = customers_serviced + 1
        elif not (len(queue) == 1 and customer_just_arrived):
            print("wasted minute at t = " + str(time))
            wasted_minutes = wasted_minutes + 1
        time = time + 1


    print("System Response Times: " + str(system_response_times))
    print("System Departure Times: " + str(system_departure_times))
    print("Wasted minutes: " + str(wasted_minutes))

    # Last system departure time is the overall minutes used
    total_time = system_departure_times[-1]
    print("Percent Server Utilization: " + str((total_time - wasted_minutes) / total_time * 100) + "%")
```

The equation used to find the percent server utilization is

```
    Percent Server Utilization = (Total Time - Wasted Minutes) / Total Time * 100
```

After running the following code, it was found that the percent server utilization for this problem was **94.1712%**, given the stated assumptions at the beginning of this problem.

c) Repeat (a) and (b) using two servers and a customer can be served by server #1 or by the trainee server #2 if the main server #1 is busy.

The following code was used to find the departure and response times if there were two servers and their server utilization time using the assumption that the total time is double for each server and that each minute one server is not serving a customer then a wasted minute is incurred.

```
arrival_times = [12, 31, 63, 95, 99, 154,
                     198, 221, 304, 346, 411, 455, 537]
    service_times = [40, 32, 55, 48, 18, 50, 47, 18, 28, 54, 40, 72, 12]
    service_times = [[0, 40], [1, 32], [2, 55], [3, 48], [4, 18], [5, 50], [6, 47], [7, 18], [8,
28], [9, 54], [10, 40], [11, 72], [12, 12]]
    # arrival_times = [2, 4, 7]
    # service_times = [3, 1, 4]
    # arrival_times = [0, 2, 3]
    # service_times = [2, 1, 3]
    # arrival_times = [2, 3, 5, 6]
    # service_times = [[0, 3], [1, 5], [2, 5], [3, 2]]
    # The departure times are the time in which the customer is completely serviced
    system_departure_times = [0] * len(arrival_times)
    # The response time is the total amount of time a customer spends in both the queue and in
service.
    system_response_times = [0] * len(arrival_times)

    time = 0
    customers_serviced = 0
    customers_arrived = 0
    queue = []
    wasted_minutes = 0
    # customer_just_arrived ensure that a customer isn't serviced in the same minute they arrive
    customer_just_arrived = False
    while customers_serviced != len(arrival_times):
        # Check if new arrival of customer should be added to queue
        if customers_arrived != len(arrival_times) and time == arrival_times[customers_arrived]:
            queue.append(service_times[customers_arrived])
            customer_just_arrived = True
            customers_arrived = customers_arrived + 1
        elif customer_just_arrived:
            customer_just_arrived = False
        # Case 1: Nobody is in queue
        if len(queue) == 0:
            wasted_minutes = wasted_minutes + 2 # + 2 since both servers are wasting minutes

        # Case 2: length of queue is 1
        if len(queue) == 1 and not customer_just_arrived:
            queue[0][1] = queue[0][1] - 1
            # If only one customer, then one server is not being used
            wasted_minutes = wasted_minutes + 1
        elif len(queue) == 1 and customer_just_arrived:  # Otherwise, customer just arrived
            wasted_minutes = wasted_minutes + 1

        # Case 3: Queue length = 2 (No wasted minutes when 2+ in queue)
        if len(queue) == 2:
            # if customer just arrived, only count first person in queue
            if customer_just_arrived:
                queue[0][1] = queue[0][1] - 1
            else:
                queue[0][1] = queue[0][1] - 1
                queue[1][1] = queue[1][1] - 1

        # Case 4: Queue length > 2 (no wasted minutes and don't need to check for customer just
arriving)
        if len(queue) > 2:
            queue[0][1] = queue[0][1] - 1
            queue[1][1] = queue[1][1] - 1

        if len(queue) == 1:
            # Check if queue spot 0 is serviced
            if queue[0][1] == 0:
                print("Popping customer from queue at t = " + str(time))
                # Delete fully serviced customer from queue since it's done being serviced
                # Add data of customer to respective system response and departure times
                system_departure_times[queue[0][0]] = time
                system_response_times[queue[0][0]] = time - arrival_times[queue[0][0]]
                customers_serviced = customers_serviced + 1
                del queue[0]
        elif len(queue) >= 2:
            # Check if either queue spot 0 or 1 are serviced
            if queue[0][1] == 0 and queue[1][1] == 0:
                print("Popping customer from queue at t = " + str(time))
                # Delete fully serviced customer from queue since it's done being serviced
                # Add data of customer to respective system response and departure times
                system_departure_times[queue[0][0]] = time
                system_response_times[queue[0][0]] = time - arrival_times[queue[0][0]]
                customers_serviced = customers_serviced + 1
                del queue[0]

                ## Repeat for next finished item in queue
                print("Popping another customer from queue at t = " + str(time))
                # Add data of customer to respective system response and departure times
                system_departure_times[queue[0][0]] = time
                system_response_times[queue[0][0]] = time - arrival_times[queue[0][0]]
                customers_serviced = customers_serviced + 1
                del queue[0]
            elif queue[0][1] == 0 and queue[1][1] != 0:
                print("Popping customer from queue at t = " + str(time))
                # Delete fully serviced customer from queue since it's done being serviced
                # Add data of customer to respective system response and departure times
                system_departure_times[queue[0][0]] = time
                system_response_times[queue[0][0]] = time - arrival_times[queue[0][0]]
                customers_serviced = customers_serviced + 1
                del queue[0]

            elif queue[0][1] != 0 and queue[1][1] == 0:
                print("Popping customer from queue at t = " + str(time))
                # Delete fully serviced customer from queue since it's done being serviced
                # Add data of customer to respective system response and departure times
                system_departure_times[queue[1][0]] = time
                system_response_times[queue[1][0]] = time - arrival_times[queue[1][0]]
                customers_serviced = customers_serviced + 1
                del queue[1]

        time = time + 1


    print("System Response Times: " + str(system_response_times))
    print("System Departure Times: " + str(system_departure_times))
    print("Wasted minutes: " + str(wasted_minutes))

    # Last system departure time is the overall minutes used
    total_time = max(system_departure_times) * 2
    print("Percent Server Utilization: " + str((total_time - wasted_minutes) / total_time * 100)
+ "%")
```

The output of the code is as follows:

```
Popping customer from queue at t = 52
Popping customer from queue at t = 63
Popping customer from queue at t = 118
Popping customer from queue at t = 136
Popping customer from queue at t = 143
Popping customer from queue at t = 204
Popping customer from queue at t = 239
Popping customer from queue at t = 245
Popping customer from queue at t = 332
Popping customer from queue at t = 400
Popping customer from queue at t = 451
Popping customer from queue at t = 527
Popping customer from queue at t = 549
System Response Times: [40, 32, 55, 48, 37, 50, 47, 18, 28, 54, 40, 72, 12]
System Departure Times: [52, 63, 118, 143, 136, 204, 245, 239, 332, 400, 451, 527,
549]
Wasted minutes: 574
Percent Server Utilization: 47.72313296903461%
```

When put into tabular form, the data appears like:

Customer	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Response Time	40	32	55	48	37	50	47	18	28	54	40	72	12
Departure Time	52	63	118	143	136	204	245	239	332	400	451	527	549

And the cumulative server utilization between the two servers was found to be **47.7231%**.

The above code is very similar to the previously discussed code from sections a and b, except that this code accounted for two servers in all of its logic. This meant that more than one minute could be wasted at a time by the servers and that the departure time would not necessarily always be increasing if a newly added customer had a shorter service time than a recently arrived previous customer.

Part 3: Probability Functions

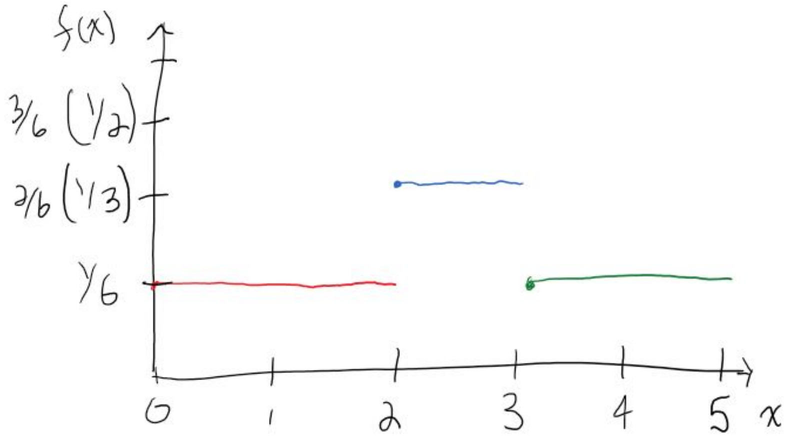
Suppose that X has a piecewise probability density function which is defined as following:

$$f(x) = \begin{cases} c_1; & 0 \leq x < 2, \\ c_2; & 2 \leq x < 3, \\ c_1; & 3 \leq x < 5; \end{cases}$$

$c1$ and c_2 are constants

(a) Sketch f(x) and find c1 and c2 assuming that the probabilities over the intervals [0, 2), [2, 3), and [3, 5) are the same and equal to 1/3 for each.

A sketch of f(x) as well as the solution to how $C_1 = \frac{1}{6}$ and $C_2 = \frac{1}{3}$ were found are shown below, with C_1; 0 <= x < 2 being shown in red, C_2; 2 <= x < 3 being shown in blue, and C_1; 3 <= x < 5 being shown in green:



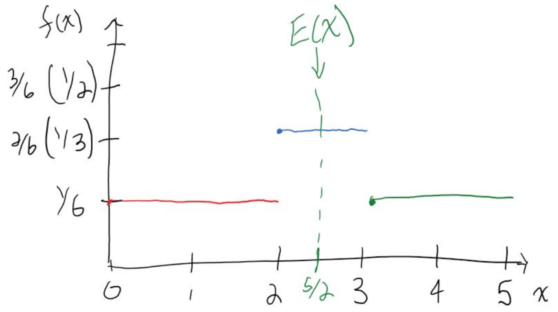
$$\int_0^2 C_1\,dx = \frac{1}{3} \Rightarrow C_1 = \frac{1}{6} \text{ since } \int_0^2 \frac{1}{6} = \frac{x}{6}\Big|_0^2 = \frac{2}{6} - \frac{0}{6} = \frac{1}{3} \checkmark$$

$$\int_2^3 C_2\,dx = \frac{1}{3} \Rightarrow C_2 = \frac{1}{3} \text{ since } \int_2^3 \frac{1}{3}dx = \frac{x}{3}\Big|_2^3 = \frac{3}{3} - \frac{2}{3} = \frac{1}{3} \checkmark$$

$$\text{Verify: } \int_3^5 C_1\,dx = \int_3^5 \frac{1}{6}dx = \frac{1}{6} \cdot x\Big|_3^5 = \frac{1}{6}(5-3) = \frac{2}{6} = \frac{1}{3} \checkmark$$

(b)Determine E(X) and Var(X)

The mean E(X) determined to be 5/2, which makes sense since the mean is halfway between the piecewise functions, at 2.5, as seen in the graph below:



The work for the mean is shown below:

$$E(X) = \int_{-\infty}^{\infty} x f(x)\,dx = \int_0^5 x f(x)\,dx = \frac{1}{6}\int_0^2 x\,dx + \frac{1}{3}\int_2^3 x\,dx + \frac{1}{6}\int_3^5 x\,dx$$

$$= \frac{x^2}{12}\Big|_0^2 + \frac{x^2}{6}\Big|_2^3 + \frac{x^2}{12}\Big|_3^5$$

$$\left(\frac{4}{12} - 0\right) + \left(\frac{9}{6} - \frac{4}{6}\right) + \left(\frac{25}{12} - \frac{9}{12}\right)$$

$$= \frac{2}{6} + \frac{5}{6} + \frac{8}{6} = \frac{15}{6} = \frac{5}{2}$$

$$\therefore\ E(x) = \frac{5}{2}$$

The variance V(X) was determined to be 7/4. The work is shown below:

Variance $V(X) = E(x^2) - E(x)^2$

$$E(x^2) = \int_{-\infty}^{\infty} x^2 f(x)\,dx = \int_0^5 x^2 f(x)\,dx$$

$$= \frac{1}{6}\int_0^2 x^2 dx + \frac{1}{3}\int_2^3 x^2 dx + \frac{1}{6}\int_3^5 x^2 dx$$

$$= \frac{x^3}{18}\Big|_0^2 + \frac{x^3}{9}\Big|_2^3 + \frac{x^3}{18}\Big|_3^5$$

$$= \left(\frac{8}{18} - 0\right) + \left(\frac{27}{9} - \frac{8}{9}\right) + \left(\frac{125}{18} - \frac{27}{18}\right)$$

$$= \frac{8 + 38 + 98}{18} = \frac{144}{18} = 8$$

$$\therefore E(x^2) = 8 \quad \& \quad E(x)^2 = \left(\frac{5}{2}\right)^2 = \frac{25}{4}$$

$$= \frac{7}{4}$$

$$\therefore V(x) = \frac{7}{4}$$

(c) P {1< X < 3}

The probability that the value lies between 1 < X < 3 is ½ or 0.5. The work in finding this answer is shown below:

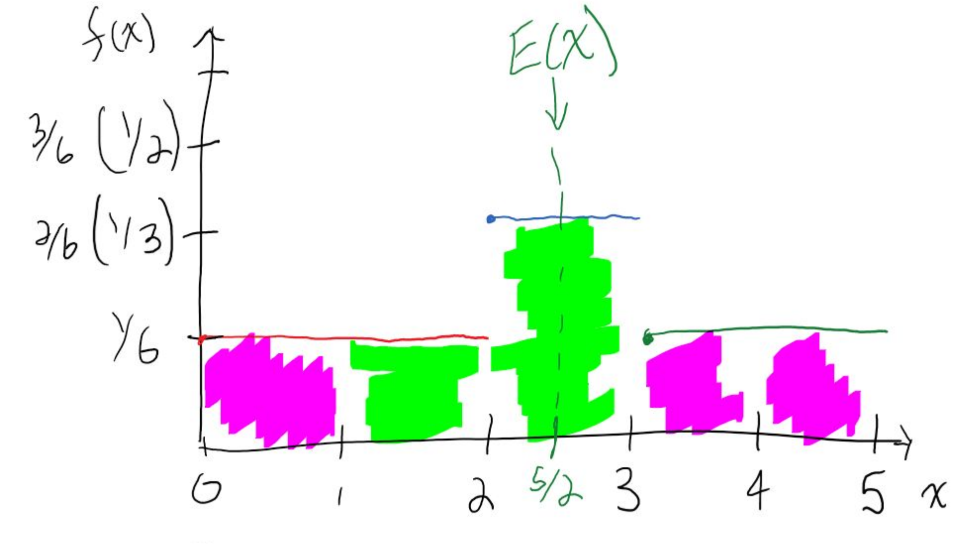
$$P\{1<X<3\} = \int_1^2 \frac{1}{6}\,dx + \int_2^3 \frac{1}{3}\,dx$$

$$= \frac{x}{6}\Big|_1^2 + \frac{x}{3}\Big|_2^3$$

$$= \frac{1}{6}(2-1) + \frac{1}{3}(3-2)$$

$$= \frac{1}{6} + \frac{1}{3} = \frac{1}{2}$$

This value makes sense, for 50% of the blocks are between 1 and 3, and those blocks are shaded in green below (notice the three green blocks and the three purple blocks, or 20%.)



(d) P{X > 3 | X > 1}

The probability that X is greater than 3 given that X > 1 is ⅖ or 0.4. The calculation is shown below:

$$P(X > 3 \mid X > 1) = \frac{P(X > 3 \wedge X > 1)}{P(X > 3)}$$

$$= \frac{P(X > 3)}{P(X > 1)}$$

$$P(X > 3) = \int_3^5 \frac{1}{6}\,dx = \frac{1}{6} x \Big|_3^5 = \frac{(5-3)}{6} = \frac{1}{3}$$

$$P(X > 1) = \int_1^2 \frac{1}{6}\,dx + \int_2^3 \frac{1}{3}\,dx + \int_3^5 \frac{1}{6}\,dx$$

$$= \frac{x}{6}\Big|_1^2 + \frac{x}{3}\Big|_2^3 + \frac{x}{6}\Big|_3^5$$

$$= \frac{(2-1)}{6} + \frac{(3-2)}{3} + \frac{(5-3)}{6}$$

$$= \frac{1}{6} + \frac{2}{6} + \frac{2}{6} = \frac{5}{6}$$

$$\therefore \frac{P(X > 3)}{P(X > 1)} = \frac{\frac{1}{3}}{\frac{5}{6}} = \frac{1}{3} \cdot \frac{6}{5} = \frac{2}{5}$$

$$\therefore P(X > 3 \mid X > 1) = \frac{2}{5} = 0.4$$

(e) Is X memoryless? Please elaborate.

X is memoryless, and this can be seen by stemming from the example in question d above. The condition of X > 1 does not in any way alter P(X > 3), X can be considered memoryless wrt t if, for all s with t not equal to 0:

$$P(x > s + t \mid x > t) = P(x > s).$$

per Wolfram Mathworld [3]. This can be disproved by part d, where, with t = 1 and s = 2, P(X > 2 + 1 | X > 1) = ⅖ is not equal to ⅕ = P (X > 2) (see below):

$$P(x > 2) = \int_{2}^{3} \frac{1}{3} dx + \int_{3}^{5} \frac{1}{6} dx$$

$$= \frac{x}{3}\Big|_{2}^{3} + \frac{x}{6}\Big|_{3}^{5} = \frac{(3-2)}{3} + \frac{(5-3)}{6}$$

$$= \frac{1}{3} + \frac{2}{6} = \frac{1}{5}$$

Since the two are not equal for all s and t as the case of s = 2 and t = 1 shows above, then X is considered to not be memoryless like it may be if it were an exponential function.

Appendix

[1] College of Saint Benedict at St. John's University Physics Department, *KS-test Data Entry*. [Online]. Available: http://www.physics.csbsju.edu/stats/KS-test.n.plot_form.html. [Accessed: 15-Apr-2020].

[2] Stat Trek, "Stat Trek," *Chi-Square Calculator*. [Online]. Available: https://stattrek.com/online-calculator/chi-square.aspx. [Accessed: 14-Apr-2020].

[3] "Memoryless," *from Wolfram MathWorld*. [Online]. Available: https://mathworld.wolfram.com/Memoryless.html. [Accessed: 15-Apr-2020].

The entire code file used for this homework is attached to this PDF on Blackboard.