**Problem 1**

a) In the previous EVAL assignment, you discovered that the client sends requests to the server with inter-arrival times that are exponentially distributed with mean 1/λ where λ is the arrival rate passed via the parameter -a. Also, the request lengths are exponentially distributed with mean 1/μ where μ is the service rate passed via the parameter -s. Great! But as it turns out, that’s just the default behavior of the client. Let’s discover what else the client can do.

Which distribution the client uses is controller with an extra parameter -d <dist. number>, where <dist. number> is a number from 0 to 2. When passing -d 0 you will be using the default exponential distribution. But what are the other two distributions? And does the -d <dist.number> parameter control both inter-arrival and service times?  
To begin answering these questions, run your server and client with the following parameters (let it run, it will take about 5 minutes):  
 ./server\_lim -q 1000 2222 & ./client -a 4.5 -s 5 -n 1500 -d 1 2222

Notice that the client is requested to generate traffic according to distribution 1. Just like you did in HW2, plot the experimental data of inter-arrival times and request lengths and recover the type and parameters of the distributions used by the client when -d 1 is passed.

Hint: there is some guesswork involved in recovering the distribution parameters. Start by looking at the shape of the distribution produced by the collected data and make a guess about which distribution might be. Setting the mean will be easy if you think about it. If there is a standard deviation to set, explore integer fractions or multiples of the mean.

Steps Taken:

Like before save the output to a server log file:

| ./build/server\_lim -q 1000 2222 > 1a\_Server\_Output.txt |
| --- |

| ./client -a 4.5 -s 5 -n 1500 -d 1 2222 |
| --- |

**Then:**

**Plot the Distributions**:

* + Use Python, matplotlib and seaborn, to plot histograms of inter-arrival times and request lengths. These histograms will help us visually analyze the shape of the distributions.

**Guess the Distribution Type**:

* + Look at the histograms and determine if they resemble a common distribution type (e.g., normal, uniform, exp).
  + Based on the shape, make a guess as to which distribution it might be.

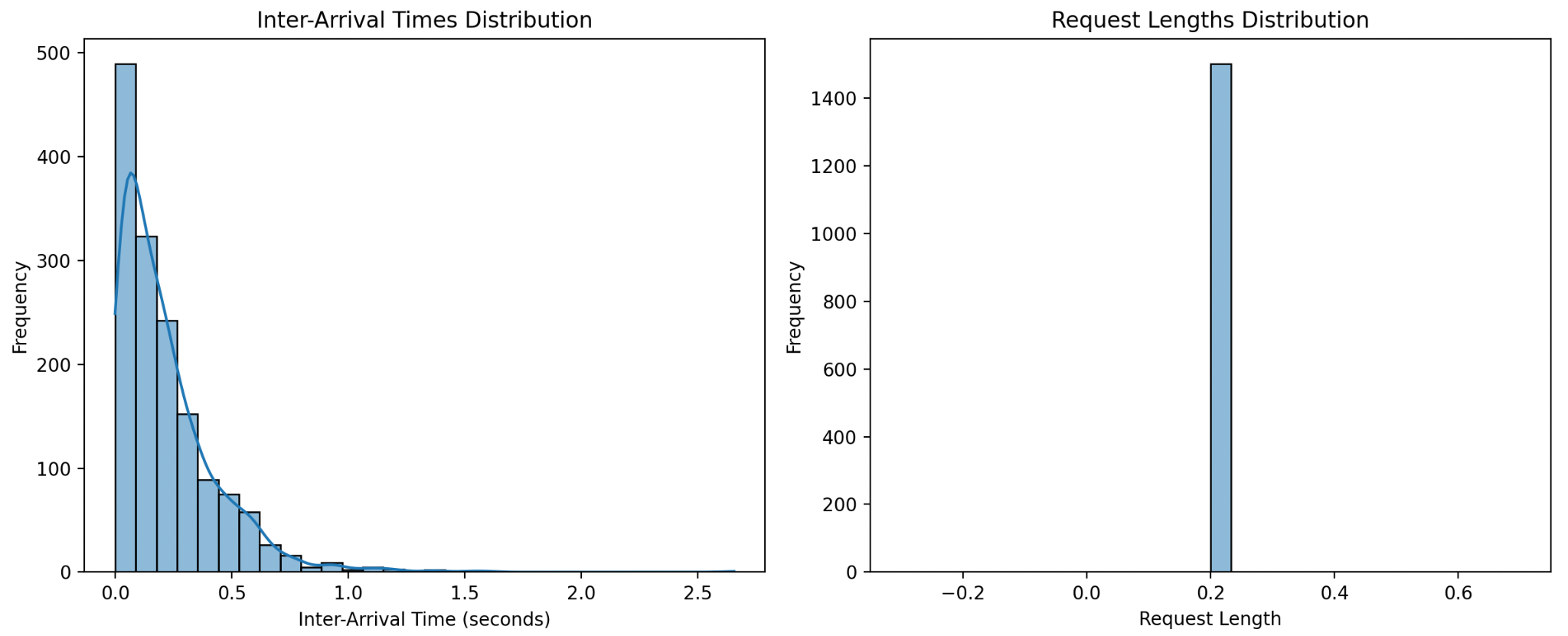
**Estimate the Parameters**:

* + Determine the distribution parameters, such as the mean and standard deviation.

**Script Used:**

| import re import pandas as pd import numpy as numpy import matplotlib.pyplot as plt import seaborn as sns   # Load the server log file\_path = './1a\_Server\_Output.txt'  # Lists to store extracted data request\_ids = [] sent\_timestamps = [] request\_lengths = []  # regex pattern to extract the data pattern = r'R(\d+):([\d\.]+),([\d\.]+),'  # Extract data from the log file with open(file\_path, 'r') as file:  for line in file:  match = re.match(pattern, line)  if match:  request\_ids.append(int(match.group(1)))  sent\_timestamps.append(float(match.group(2)))  request\_lengths.append(float(match.group(3)))  # create a dataFrame for analysis  data = pd.DataFrame({  'Request\_ID' : request\_ids,  'Sent\_Timestamp': sent\_timestamps,  'Request\_Length': request\_lengths })  # Calculate inter-arrival times data['Inter\_Arrival\_Time'] = data['Sent\_Timestamp'].diff()  # Plot histograms for Inter-Arrival Times and Request Length plt.figure(figsize=(12,5))  # Inter-Arrival Times Histogram plt.subplot(1,2,1) sns.histplot(data['Inter\_Arrival\_Time'].dropna(), kde=True, bins=30) plt.title('Inter-Arrival Times Distribution') plt.xlabel('Inter-Arrival Time (seconds)') plt.ylabel('Frequency')  # Request Length Histogram plt.subplot(1,2,2) sns.histplot(data['Request\_Length'], kde=True, bins=30) plt.title('Request Lengths Distribution') plt.xlabel('Request Length') plt.ylabel('Frequency')  plt.tight\_layout() plt.show() |
| --- |

**Output:**

****

It looks like the inter-arrival times follow an exponential distribution while the request lengths follow a normal distribution.

**Lets now fit distributions to extract the parameters:**

**Script Used (code after previous code):**

############# Fitting Inter-Arrival Times to Exponential and Normal Distributions

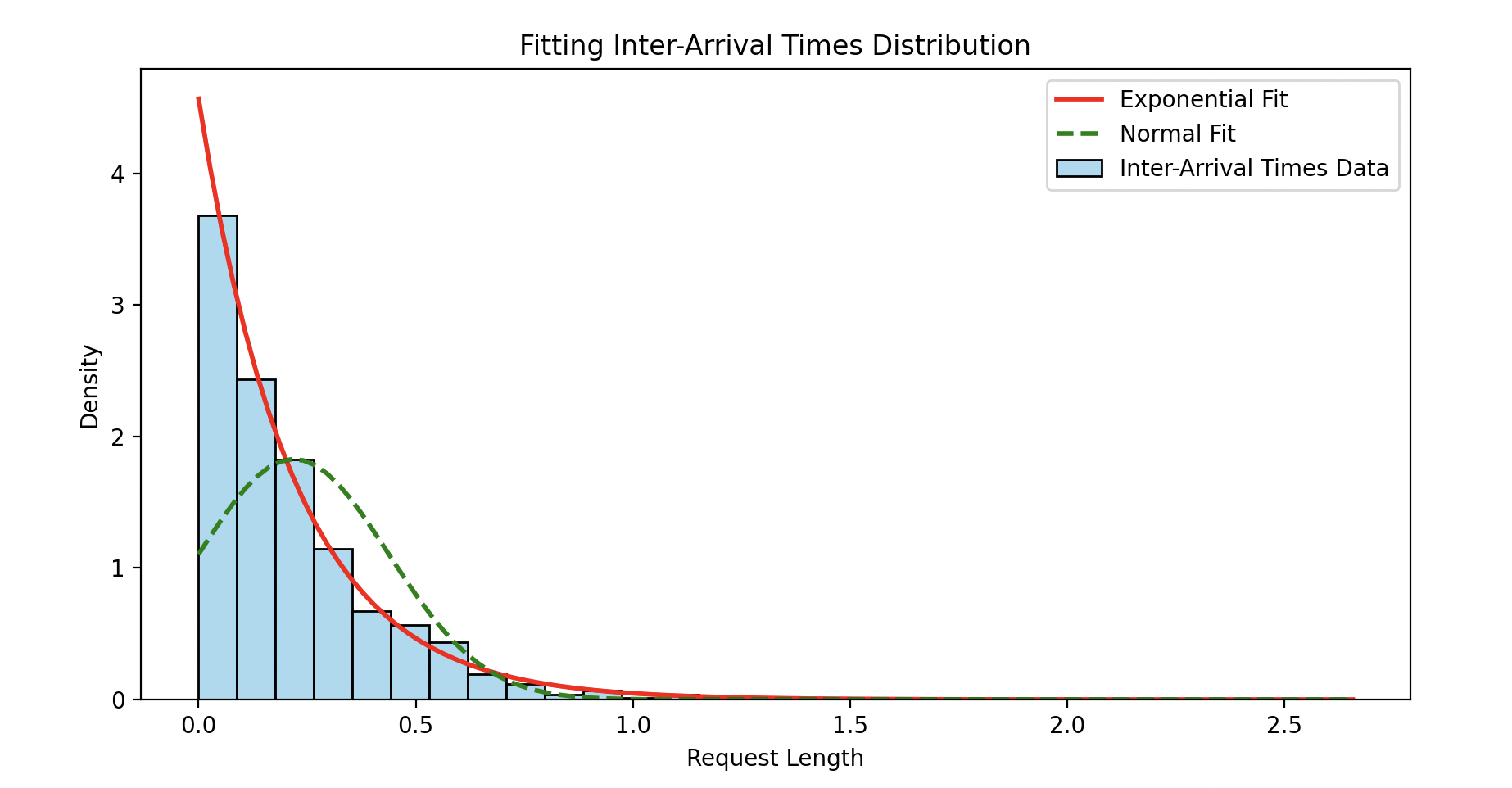
# Extract the inter-arrival times, removing any NaN values.

inter\_arrival\_times = data['Inter\_Arrival\_Time'].dropna()

# Exponential Fit

| """ expon.fit(inter\_arrival\_times): Fits the exponential distribution to the data using scipy.stats.expon.fit(). It estimates the parameters of the distribution that best fit the given data. For an exponential distribution, the parameters returned are: Location parameter (loc): Usually represents the starting point of the distribution. Scale parameter (scale): Corresponds to 1/λ (where λ is the rate parameter). """ exp\_params = expon.fit(inter\_arrival\_times)  # Extract the Scale parameter - the mean exp\_mean = exp\_params[1]  # Compute the PDF for exponential """ np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100):  Creates an array of 100 evenly spaced values between the minimum and maximum inter-arrival times, which we use as the x-values for plotting. expon.pdf(..., \*exp\_params):  Computes the probability density function (PDF) of the exponential distribution at the given x-values, using the previously obtained parameters (\*exp\_params). """ exp\_pdf = expon.pdf(np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100), \*exp\_params)  # Normal Fit norm\_params = norm.fit(inter\_arrival\_times)  # Mean and STD norm\_mean, norm\_std = norm\_params norm\_pdf = norm.pdf(np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100), \*norm\_params)  plt.figure(figsize=(10, 5)) """ sns.histplot(..., kde=False, bins=30, ...): Plots a histogram of the inter-arrival times data using 30 bins. The kde=False parameter means that a kernel density estimate is not plotted. stat="density": Normalizes the histogram to show density rather than counts. """ sns.histplot(inter\_arrival\_times, kde=False, bins=30, color='skyblue', label='Inter-Arrival Times Data', stat="density")  """ plt.plot(...): Plots the fitted exponential (red solid line) and normal (green dashed line) distributions. 'r-' and 'g--' specify the line colors (r for red, g for green) and styles (- for solid, -- for dashed). lw=2: Sets the line width to 2. """ plt.plot(np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100), exp\_pdf, 'r-', lw=2, label='Exponential Fit') plt.plot(np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100), norm\_pdf, 'g--', lw=2, label='Normal Fit')  plt.title('Fitting Request Lengths Distribution') plt.xlabel('Request Length') plt.ylabel('Density') plt.legend() plt.show()  ############ Fitting Request Lengths to Uniform Distribution  # Extract the request lengths request\_lengths = data['Request\_Length']  # Uniform Fit """ uniform.fit(request\_lengths): Fits a uniform distribution to the request lengths, estimating: Location parameter (loc): The minimum value of the distribution. Scale parameter (scale): The range (i.e., difference between max and min). """ uniform\_params = uniform.fit(request\_lengths)  uniform\_min, uniform\_range = uniform\_params  uniform\_pdf = uniform.pdf(np.linspace(request\_lengths.min(), request\_lengths.max(), 100), \*uniform\_params)  # Plotting the fit for request Lengths plt.figure(figsize=(10, 5)) sns.histplot(request\_lengths, kde=False, bins=30, color='lightgreen', label='Request Lengths Data', stat="density")  plt.plot(np.linspace(request\_lengths.min(), request\_lengths.max(), 100), uniform\_pdf, 'b-', lw=2, label='Uniform Fit')  plt.title('Fitting Request Lengths Distribution') plt.xlabel('Request Length') plt.ylabel('Density') plt.legend() plt.show() |
| --- |

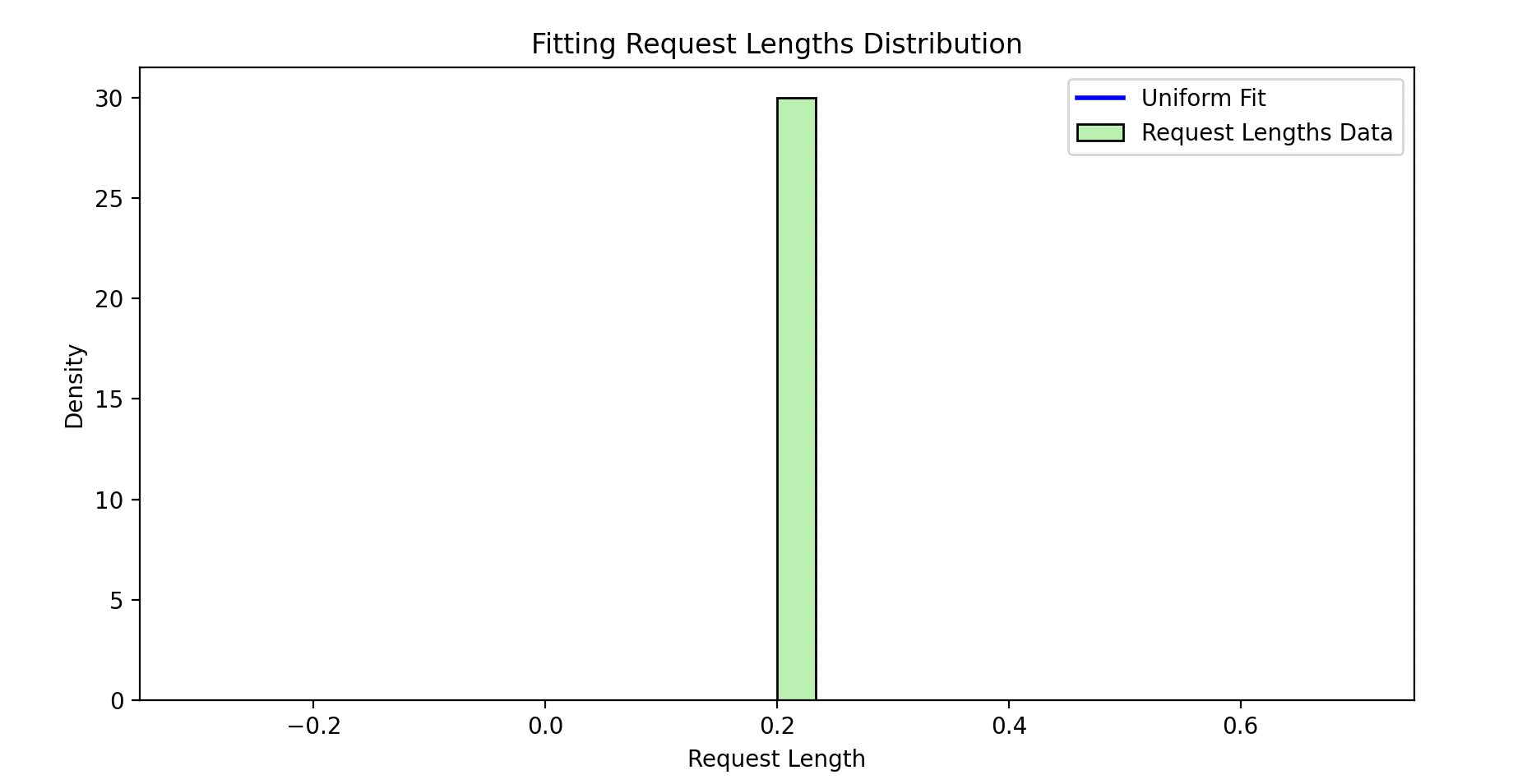
**Request Inter Distributions**



Exponential dist with mean:

exp\_mean = 0.2189292414812837

**Fitting Request Lengths Distribution**



uniform\_range ≈ 0.01 seconds

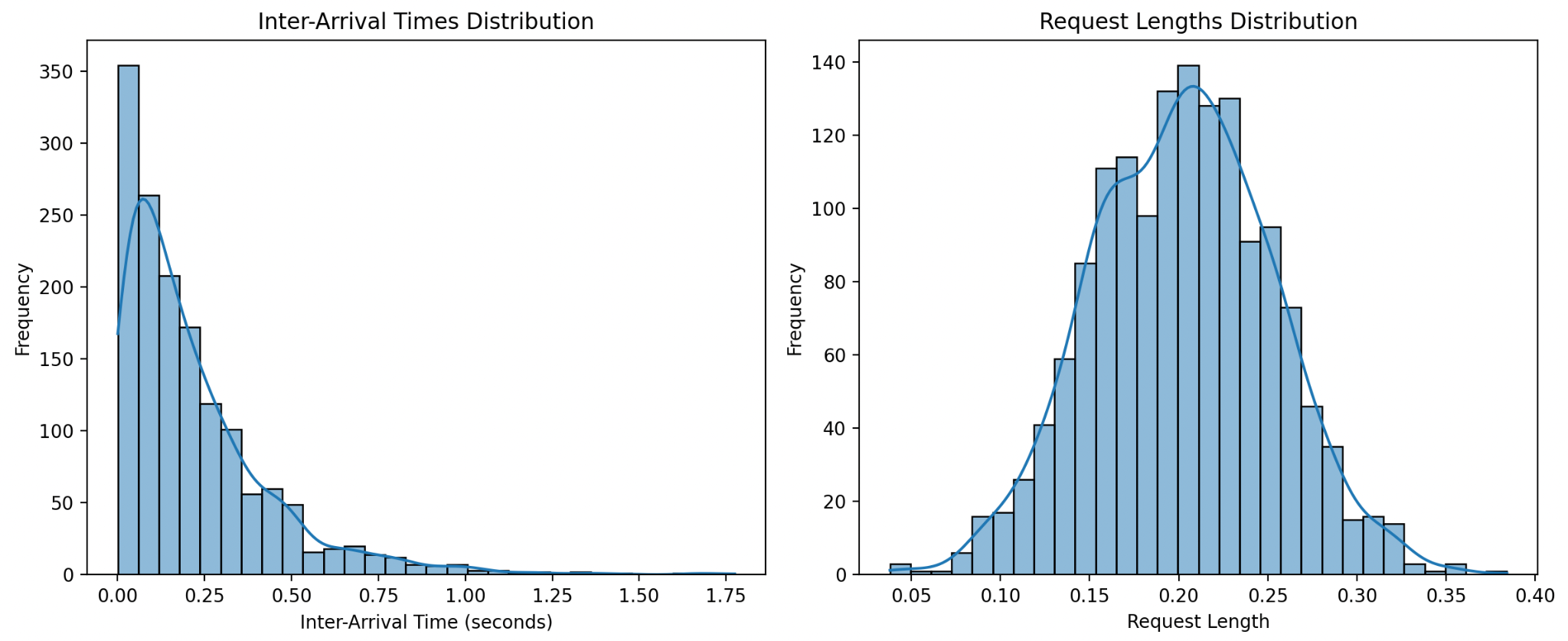
b) Do the same as above to recover the parameters of distribution number 2. In a similar way as above, collect and post-process the server output data generated by the following parameter:

./server\_lim -q 1000 2222 & ./client -a 4.5 -s 5 -n 1500 -d 2 2222

When decoding the distributions used by the client and their parameters, make conclusive statements about the distribution type for both inter-arrival times and service lengths, and explicitly state their mean and standard deviation parameters.

**Exact same script for first part:**

| from scipy.stats import expon, norm, uniform import re import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns   # Load the server log file\_path = './1b\_Server\_Output.txt'  # Lists to store extracted data request\_ids = [] sent\_timestamps = [] request\_lengths = []  # regex pattern to extract the data pattern = r'R(\d+):([\d\.]+),([\d\.]+),'  # Extract data from the log file with open(file\_path, 'r') as file:  for line in file:  match = re.match(pattern, line)  if match:  request\_ids.append(int(match.group(1)))  sent\_timestamps.append(float(match.group(2)))  request\_lengths.append(float(match.group(3)))  # create a dataFrame for analysis  data = pd.DataFrame({  'Request\_ID' : request\_ids,  'Sent\_Timestamp': sent\_timestamps,  'Request\_Length': request\_lengths })  # Calculate inter-arrival times data['Inter\_Arrival\_Time'] = data['Sent\_Timestamp'].diff()  # Plot histograms for Inter-Arrival Times and Request Length plt.figure(figsize=(12,5))  # Inter-Arrival Times Histogram plt.subplot(1,2,1) sns.histplot(data['Inter\_Arrival\_Time'].dropna(), kde=True, bins=30) plt.title('Inter-Arrival Times Distribution') plt.xlabel('Inter-Arrival Time (seconds)') plt.ylabel('Frequency')  # Request Length Histogram plt.subplot(1,2,2) sns.histplot(data['Request\_Length'], kde=True, bins=30) plt.title('Request Lengths Distribution') plt.xlabel('Request Length') plt.ylabel('Frequency')  plt.tight\_layout() plt.show() |
| --- |



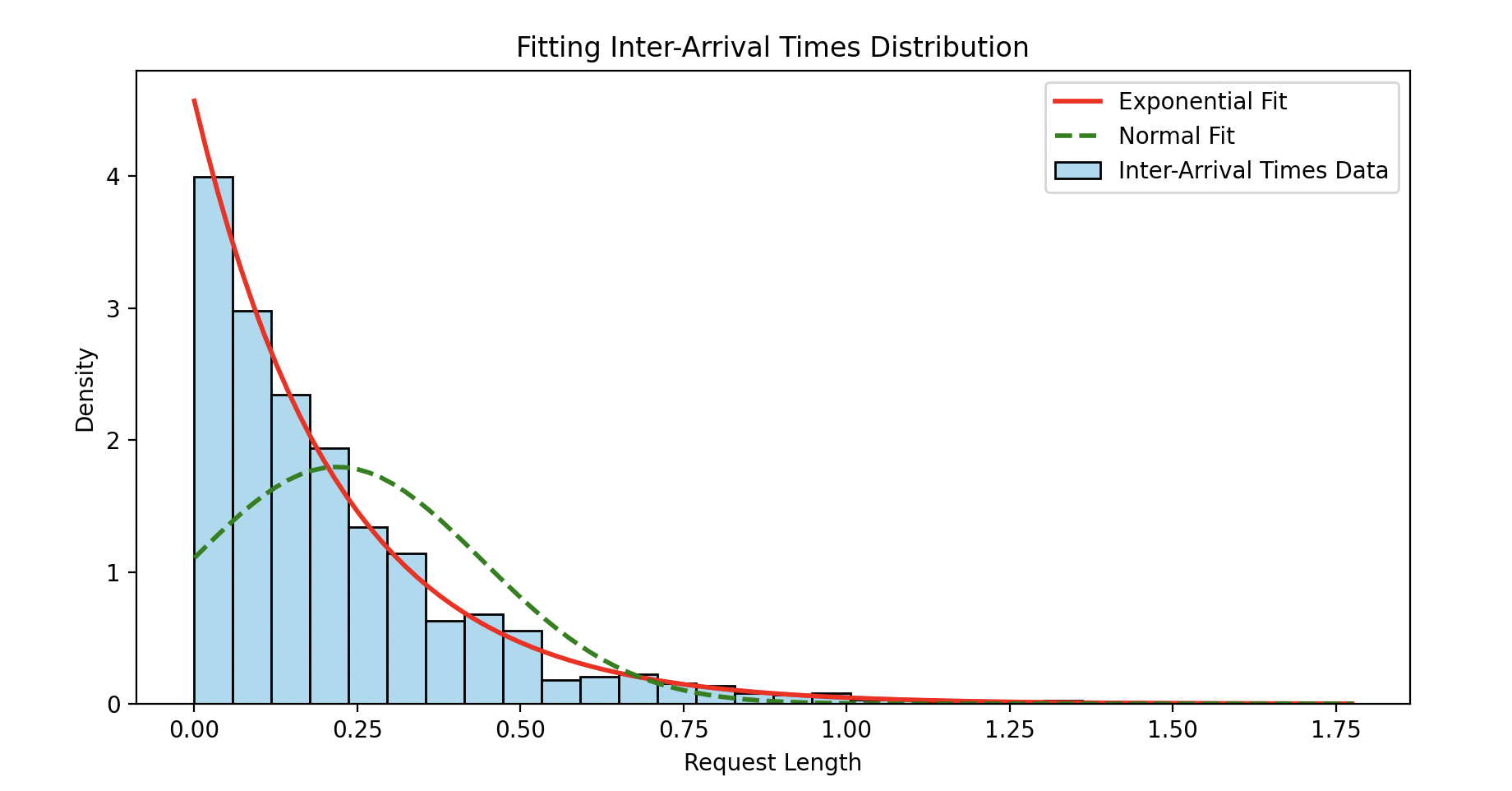
The Inter-Arrival times histogram appears to follow a pattern that could fit an exponential distribution.

While the request lengths distribution seems to follow a normal distribution.

**Lets find the parameters:**

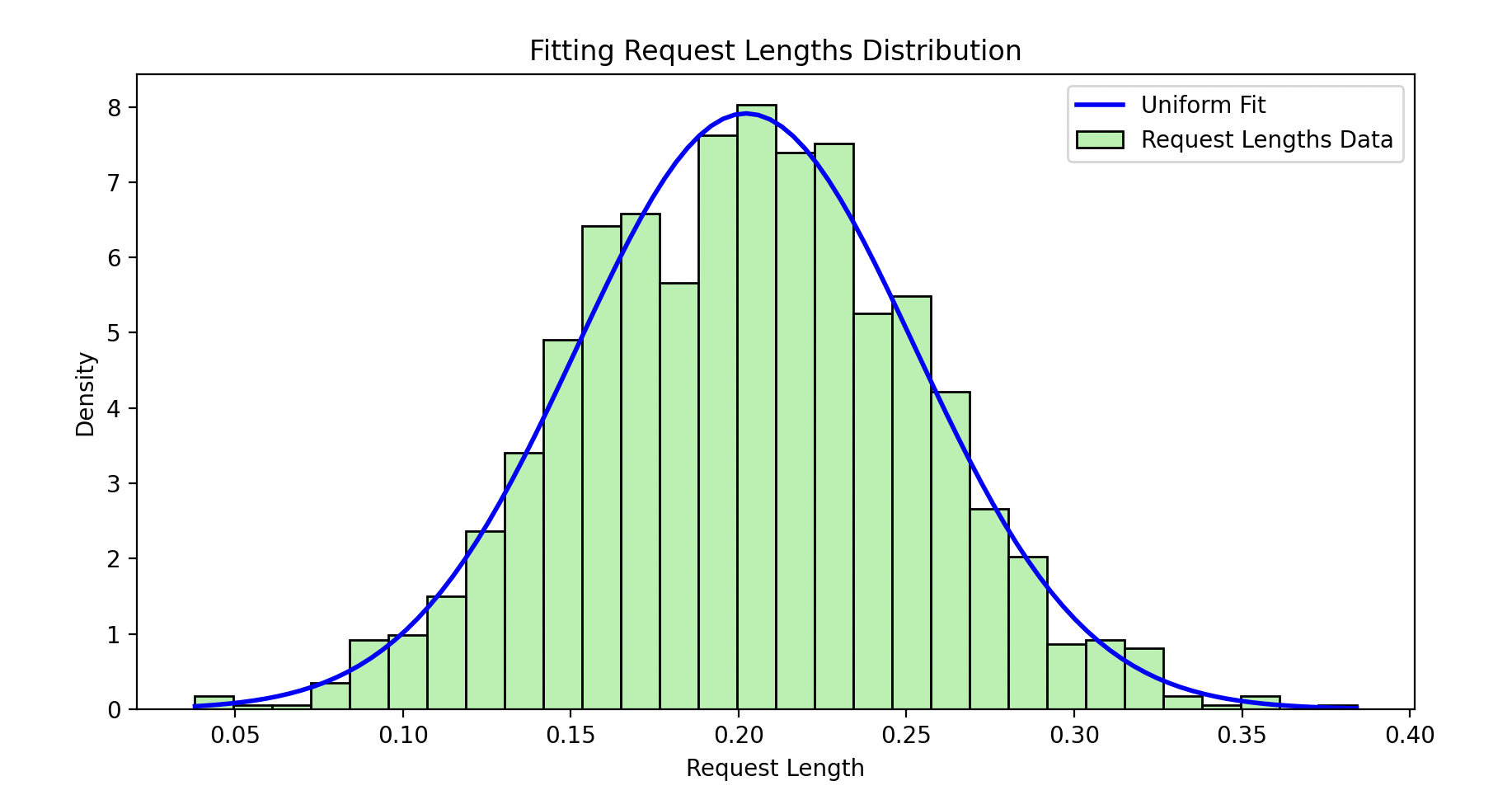
**Same script as before, just using a normal dist for the request lengths**

| ############# Fitting Inter-Arrival Times to Exponential and Normal Distributions  # Extract the inter-arrival times, removing any NaN values. inter\_arrival\_times = data['Inter\_Arrival\_Time'].dropna()  # Exponential Fit """ expon.fit(inter\_arrival\_times): Fits the exponential distribution to the data using scipy.stats.expon.fit(). It estimates the parameters of the distribution that best fit the given data. For an exponential distribution, the parameters returned are: Location parameter (loc): Usually represents the starting point of the distribution. Scale parameter (scale): Corresponds to 1/λ (where λ is the rate parameter). """ exp\_params = expon.fit(inter\_arrival\_times)  # Extract the Scale parameter - the mean exp\_mean = exp\_params[1] print(f'exp\_mean = {exp\_mean}') print(f'exp\_std = {exp\_mean\*\*2}') # Compute the PDF for exponential """ np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100):  Creates an array of 100 evenly spaced values between the minimum and maximum inter-arrival times, which we use as the x-values for plotting. expon.pdf(..., \*exp\_params):  Computes the probability density function (PDF) of the exponential distribution at the given x-values, using the previously obtained parameters (\*exp\_params). """ exp\_pdf = expon.pdf(np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100), \*exp\_params)  # Normal Fit norm\_params = norm.fit(inter\_arrival\_times)  # Mean and STD norm\_mean, norm\_std = norm\_params norm\_pdf = norm.pdf(np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100), \*norm\_params)  plt.figure(figsize=(10, 5)) """ sns.histplot(..., kde=False, bins=30, ...): Plots a histogram of the inter-arrival times data using 30 bins. The kde=False parameter means that a kernel density estimate is not plotted. stat="density": Normalizes the histogram to show density rather than counts. """ sns.histplot(inter\_arrival\_times, kde=False, bins=30, color='skyblue', label='Inter-Arrival Times Data', stat="density")  """ plt.plot(...): Plots the fitted exponential (red solid line) and normal (green dashed line) distributions. 'r-' and 'g--' specify the line colors (r for red, g for green) and styles (- for solid, -- for dashed). lw=2: Sets the line width to 2. """ plt.plot(np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100), exp\_pdf, 'r-', lw=2, label='Exponential Fit') plt.plot(np.linspace(inter\_arrival\_times.min(), inter\_arrival\_times.max(), 100), norm\_pdf, 'g--', lw=2, label='Normal Fit')  plt.title('Fitting Inter-Arrival Times Distribution') plt.xlabel('Request Length') plt.ylabel('Density') plt.legend() plt.show()  ############ Fitting Request Lengths to Uniform Distribution  # Extract the request lengths request\_lengths = data['Request\_Length']  # Normal Fit norm\_params2 = norm.fit(request\_lengths)  # Mean and STD norm\_mean2, norm\_std2 = norm\_params2 norm\_pdf2 = norm.pdf(np.linspace(request\_lengths.min(), request\_lengths.max(), 100), \*norm\_params2) print(f"Norm mean: {norm\_mean2}, STD: {norm\_std2}")   # Plotting the fit for request Lengths plt.figure(figsize=(10, 5)) sns.histplot(request\_lengths, kde=False, bins=30, color='lightgreen', label='Request Lengths Data', stat="density")  plt.plot(np.linspace(request\_lengths.min(), request\_lengths.max(), 100), norm\_pdf2, 'b-', lw=2, label='Uniform Fit')  plt.title('Fitting Request Lengths Distribution') plt.xlabel('Request Length') plt.ylabel('Density') plt.legend() plt.show() |
| --- |



exp\_mean = 0.21881210734035442

exp\_std = 0.047878738318726785



Norm mean: 0.20219557866666665,

STD: 0.050416546080612044

c) We are now ready to see how different distributions affect the quality of service in our simple FIFO server. Let us focus on the comparison between an exponential distribution and whatever distribution 1 is. Run and collect experimental data for the following template command:  
   
 ./server\_lim -q 1000 2222 & ./client -a <arr. rate> -s 20 -n 1500 -d 0 2222

where <arr. rate> is varied from (and including) 10 up until 19. Notice that these experiments will ask the client to use an exponential distribution (-d 0). Use this set of experiments to produce a plot of the average response time as a function of the server utilization—in a way similar to what you did in HW1.  
   
Overlap on the same plot produced above the curve obtained by post-processing in the same exact way the results coming from running the following template command:  
   
 ./server\_lim -q 1000 2222 & ./client -a <arr. rate> -s 20 -n 1500 -d 1 2222

where once again <arr. rate> is varied from (and including) 10 up until 19. What can you conclude about the quality service perceived by the user when the load (a.k.a. its utilization) is comparable and only the distribution of the traffic characteristics changes?

**Steps Takens:**

First wrote a simple bash script to automate running the distributions:

| #!/bin/bash  # Script to automate server-client experiment runs and save the results to output files  # Define the port number and queue size PORT=2222 QUEUE\_SIZE=1000  # Define the output directory OUTPUT\_DIR="1c\_output\_files"  # Create the output directory if it doesn't exist mkdir -p $OUTPUT\_DIR  # Define arrival rate range START\_ARR\_RATE=10 END\_ARR\_RATE=19  # Loop for both distribution types (-d 0 and -d 1) for DIST in 0 1 do  # Loop through arrival rates from 10 to 19  for ARR\_RATE in $(seq $START\_ARR\_RATE $END\_ARR\_RATE)  do   # Run the server and save the output to a file  SERVER\_OUTPUT\_FILE="$OUTPUT\_DIR/1c\_server\_output\_d${DIST}\_a${ARR\_RATE}.txt"  ./build/server\_lim -q $QUEUE\_SIZE $PORT > $SERVER\_OUTPUT\_FILE & SERVER\_PID=$!    # Give the server a moment to start  sleep 1   # Run the client  ./client -a $ARR\_RATE -s 20 -n 1500 -d $DIST $PORT    # Kill the server process   kill $SERVER\_PID   # Give the server a moment to shut down  sleep 1  done done  echo "All experiments completed. Results saved in $OUTPUT\_DIR." |
| --- |

**Then, I wrote a Python script to process the output files:**

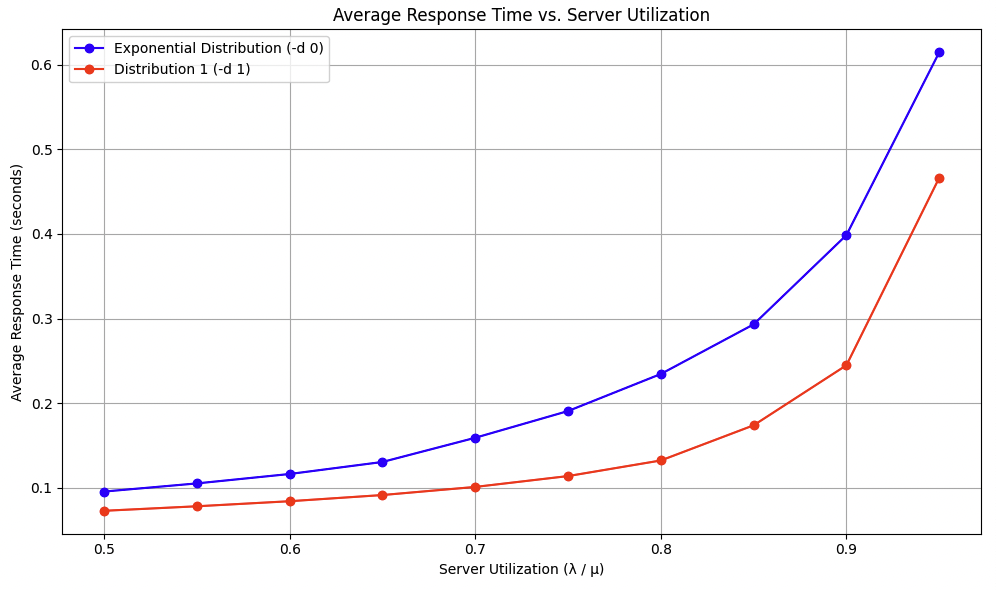
### **Plan for Processing the Output Files:**

1. **Extract Data**:
   * From each server output file, extract relevant timestamps to calculate the **response time** for each request.
   * The response time is calculated as: Response Time=Completion Timestamp−Receipt Timestamp.
   * Calculate the **average response time** for each run.
2. **Calculate Server Utilization**:
   * The **utilization** (U) of the server is given by:U=λ/μ
   * Where:
     + λ: Arrival rate (varied from 10 to 19).
     + μ=20: Fixed service rate.
3. **Produce Plots**:
   * Plot **average response time** as a function of **server utilization** for both distributions (-d 0 and -d 1).
   * Overlap both plots for comparison.



| import os import re import pandas as pd  import matplotlib.pyplot as plt   # Define the directory containing the server output files output\_dir = '1c\_output\_files'  # Intialize list to store results arrival\_rates = list(range(10, 20)) average\_response\_times\_d0 = [] average\_response\_times\_d1 = []  # Define regex pattern to extract relevant data from the server output pattern = r'R(\d+):([\d\.]+),([\d\.]+),([\d\.]+),([\d\.]+),([\d\.]+)'   # Loop through arrival rates from 10 to 19 for arrival\_rate in range(10,20):  for dist in [0,1]:  # Define the filename  filename = f"1c\_server\_output\_d{dist}\_a{arrival\_rate}.txt"  filepath = os.path.join(output\_dir, filename)   # Initialize a list to store response times  response\_times = []   # Read the file and extract response times  with open(filepath, 'r') as file:  for line in file:  match = re.match(pattern, line)  if match:  receipt\_timestamp = float(match.group(4))  completion\_timestamp = float(match.group(6))  response\_time = completion\_timestamp - receipt\_timestamp  response\_times.append(response\_time)    # Calculate the average response time for the current arrival rate and dist  avg\_response\_time = sum(response\_times) / len(response\_times) if response\_times else 0   # Store the result  if dist == 0:  average\_response\_times\_d0.append(avg\_response\_time)  else:  average\_response\_times\_d1.append(avg\_response\_time)  # Calculate server utilization (U = lambda / mu, where mu = 20) service\_rate = 20 utilization = [arr\_rate / service\_rate for arr\_rate in arrival\_rates]   # Plot the results plt.figure(figsize=(10, 6)) plt.plot(utilization, average\_response\_times\_d0, marker='o', linestyle='-', color='b', label='Exponential Distribution (-d 0)') plt.plot(utilization, average\_response\_times\_d1, marker='o', linestyle='-', color='r', label='Distribution 1 (-d 1)')  plt.xlabel('Server Utilization (λ / μ)') plt.ylabel('Average Response Time (seconds)') plt.title('Average Response Time vs. Server Utilization') plt.legend() plt.grid(True) plt.tight\_layout() plt.show() |
| --- |

**Output:**

****

When the inter-arrival times and request lengths follow an **exponential distribution**, the average response time increases sharply as the server utilization approaches 1 (i.e., as λ/μ approaches 1).

With **Distribution 1**, where inter-arrival times are **exponentially distributed,** and request lengths are **uniformly distributed**, the average response time increases more gradually with server utilization.

The normal and uniform distributions have lower variability compared to the exponential distribution.

Even with the same average load, the **distribution characteristics of the traffic significantly impact the server's performance and the user's experience**. Systems should account for traffic variability in their design and capacity planning to ensure high-quality service.

d) Let us now explore what happens when the queue has a constrained size. To do that, run the following command:  
   
 ./server\_lim -q 10 2222 & ./client -a 19.6 -s 20 -n 1500 -d 0 2222

Post-process the server output to understand what happened to our requests. First, calculate the ratio of requests that get rejected over the total. Next, plot the distribution of the inter-rejection time, i.e. the time that elapses between a rejection and the next. What does that distribution look like? Do not recover the parameters of the distribution, but simply share your insights on the shape of the distribution.

### **Objective:**

1. Calculate the ratio of rejected requests over the total number of requests.
2. Plot the distribution of the inter-rejection times (time between consecutive rejections).

**Script Used:**

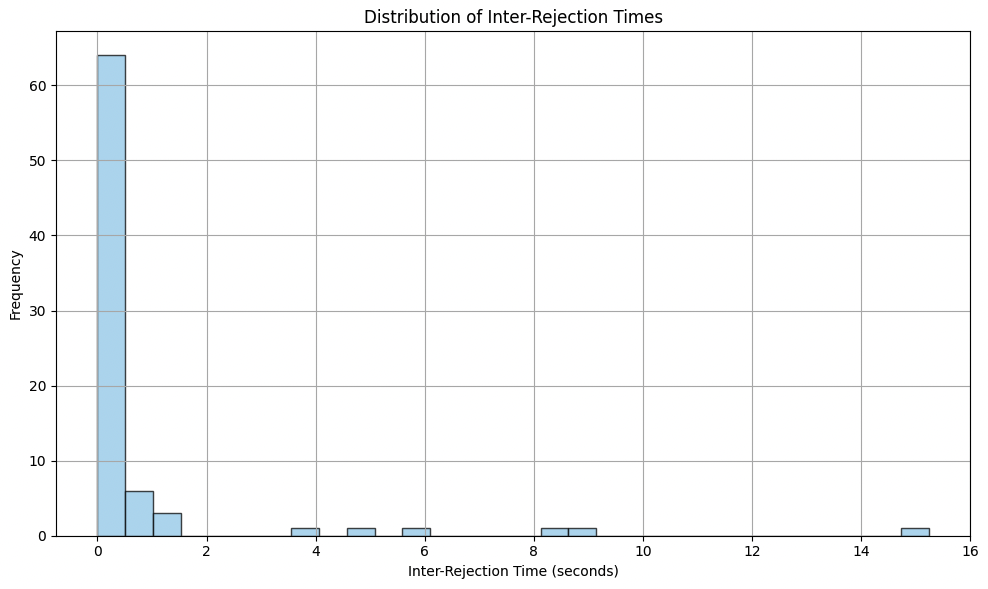
| import re import matplotlib.pyplot as plt  server\_output\_file = '1d\_server\_output.txt'   # Initialize counters and lists total\_requests = 0 rejected\_requests = 0 rejection\_timestamps = []  # Pattern for accepted requests accepted\_pattern = r'^R(\d+):([\d\.]+),([\d\.]+),([\d\.]+),([\d\.]+),([\d\.]+)' # Pattern for rejected requests rejected\_pattern = r'^X(\d+):([\d\.]+),([\d\.]+),([\d\.]+)'  # Read the server output file with open(server\_output\_file, 'r') as file:  for line in file:  # Check for accepted requests  accepted\_match = re.match(accepted\_pattern, line)  if accepted\_match:  total\_requests += 1  continue # Move to next line   # Check for rejected requests  rejected\_match = re.match(rejected\_pattern, line)  if rejected\_match:  total\_requests += 1  rejected\_requests += 1   # Extract the reject timestamp  reject\_timestamp = float(rejected\_match.group(4))  rejection\_timestamps.append(reject\_timestamp)  # Calculate the rejection ratio rejection\_ratio = rejected\_requests / total\_requests if total\_requests > 0 else 0  print(f"Total Requests: {total\_requests}") print(f"Rejected Requests: {rejected\_requests}") print(f"Rejection Ratio: {rejection\_ratio:.4f}")  # Calculate inter-rejection times # Sort the rejection timestamps in case they're out of order rejection\_timestamps.sort()  """ zip(rejection\_timestamps[:-1], rejection\_timestamps[1:]): The zip() function combines two or more iterables (like lists) into an iterator of tuples. In this case, it pairs each element from rejection\_timestamps[:-1] with the corresponding element from rejection\_timestamps[1:]. Effectively, it creates pairs of consecutive rejection timestamps.  # Given: rejection\_timestamps[:-1] = [t0, t1, t2, t3] rejection\_timestamps[1:] = [t1, t2, t3, t4]  # zip() pairs: zip(rejection\_timestamps[:-1], rejection\_timestamps[1:]) # Results in [(t0, t1), (t1, t2), (t2, t3), (t3, t4)]  """ inter\_rejection\_times = [t2 - t1 for t1, t2 in zip(rejection\_timestamps[:-1], rejection\_timestamps[1:])]  # Plot the distribution of inter-rejection times plt.figure(figsize=(10, 6)) plt.hist(inter\_rejection\_times, bins=30, color='skyblue', edgecolor='black', alpha=0.7) plt.title('Distribution of Inter-Rejection Times') plt.xlabel('Inter-Rejection Time (seconds)') plt.ylabel('Frequency') plt.grid(True) plt.tight\_layout() plt.show() |
| --- |

**Output:**

Total Requests: 1500

Rejected Requests: 80

Rejection Ratio: 0.0533



It seems to follow an exponential distribution.

e) Repeat the same analysis as above, when distribution number 1 is used instead, thus by running the following command:

./server\_lim -q 10 2222 & ./client -a 19.6 -s 20 -n 1500 -d 1 2222

If you compare the new rejection rate and shape of the distribution, would you conclude that the new system (the one that uses -d 1) offers a better or worse service to its users?

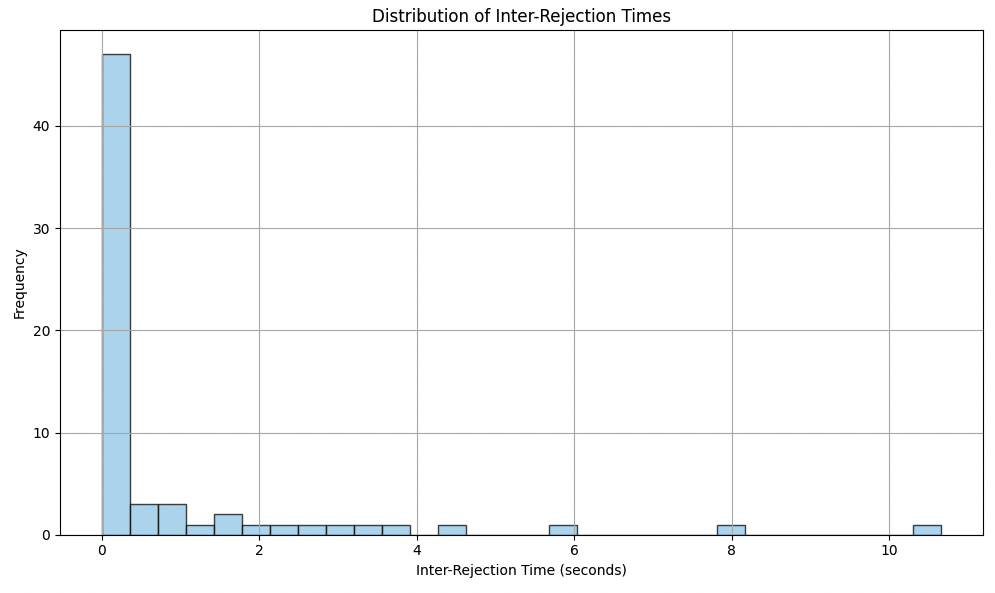
Used the exact same script as above just different file.

**Output:**

Total Requests: 1500

Rejected Requests: 67

Rejection Ratio: 0.0447



It's hard to say. Both distributions have similar rejection rates with dist 1 being marginally smaller however, dist 1 also has higher variability as rejections seems to occur more unpredictably.