

# Midterm Problem Set **Report**

---

Kevin Chen

March 21 2023

# Problem 1

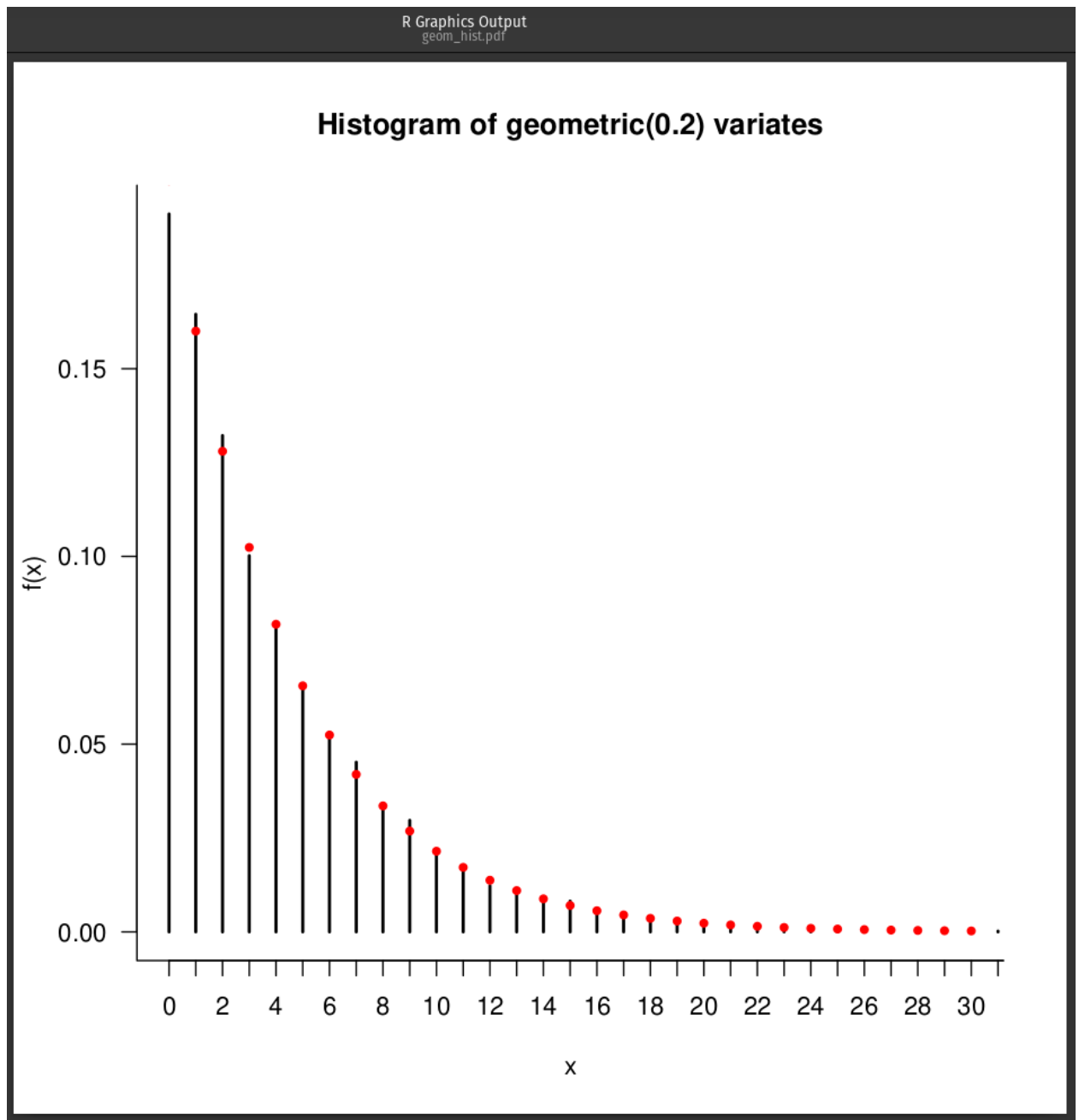
## Method Implementations

```
RNG.py x ssq.py RNG.R ssq copy.py ssq copy 2.py problem2c.R problem3.py
RNG.py > RNG > uniform
54
55 @classmethod
56 def random(cls, which_stream: Stream) -> np.float64:
57     """Generate a random float from a uniform distribution between 0 and 1.
58
59     Parameters:
60         which_stream (Stream): Enum value specifying which stream to draw the random number from.
61
62     Returns:
63         np.float64: A random float from a uniform distribution between 0 and 1.
64     """
65     if not cls._initialized:
66         cls._initializeStreams()
67     generator = cls._streams[which_stream.value]
68     return generator.random()
69
70 @classmethod
71 def randint(cls, a: int, b: int, which_stream: Stream) -> np.int64:
72     """Generate a random integer between a and b-1.
73
74     Parameters:
75         a (int): The lower bound (inclusive) of the interval.
76         b (int): The upper bound (exclusive) of the interval.
77         which_stream (Stream): Enum value specifying which stream to draw the random number from.
78
79     Returns:
80         np.int64: A random integer between a and b-1.
81     """
82     if not cls._initialized:
83         cls._initializeStreams()
84     generator = cls._streams[which_stream.value]
85     return generator.integers(a, b)
86
87 @classmethod
88 def uniform(cls, a: float, b: float, which_stream: Stream) -> np.float64:
89     """Generate a random float from a uniform distribution between a and b.
90
91     Parameters:
92         a (float): The lower bound (inclusive) of the interval.
93         b (float): The upper bound (exclusive) of the interval.
94         which_stream (Stream): Enum value specifying which stream to draw the random number from.
95
96     Returns:
97         np.float64: A random float from a uniform distribution between a and b.
98     """
99     if not cls._initialized:
100         cls._initializeStreams()
101     generator = cls._streams[which_stream.value]
102     return generator.uniform(a, b)
103
104 @classmethod
105 def exponential(cls, mu: float, which_stream: Stream) -> np.float64:
106     """Generate a random float from an exponential distribution with parameter mu.
107
108     Parameters:
109         mu (float): The rate parameter, which is the inverse of the mean.
110         which_stream (Stream): Enum value specifying which stream to draw the random number from.
111     """
```

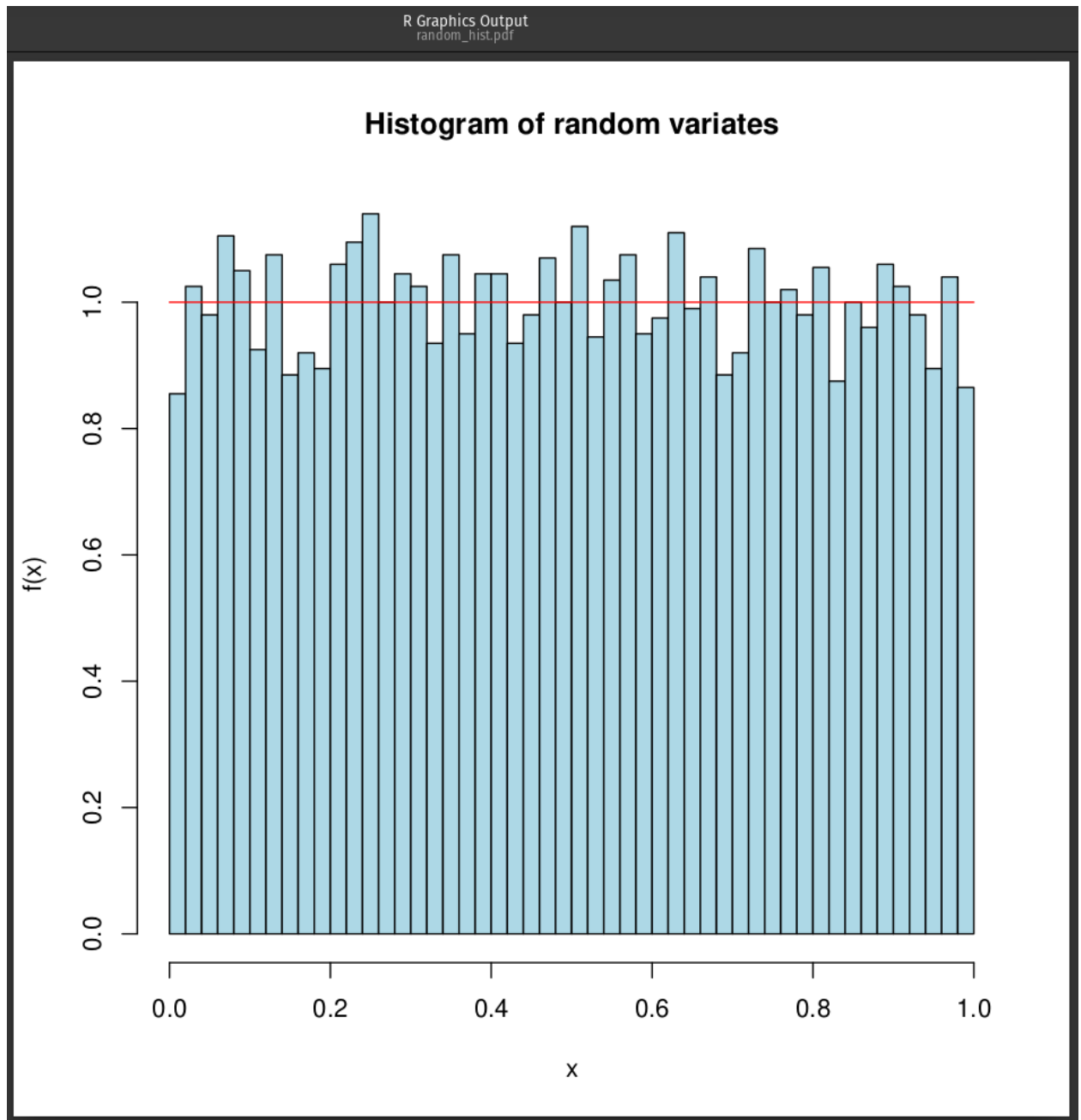
```
RNG.py x ssq.py RNG.R ssq copy.py ssq copy 2.py problem2c.R problem3.py
RNG.py > RNG > uniform
106 """Generate a random float from an exponential distribution with parameter mu.
107
108 Parameters:
109     mu (float): The rate parameter, which is the inverse of the mean.
110     which_stream (Stream): Enum value specifying which stream to draw the random number from.
111
112 Returns:
113     np.float64: A random float from an exponential distribution with parameter mu.
114 """
115 if not cls._initialized:
116     cls._initializeStreams()
117 generator = cls._streams[which_stream.value]
118 return generator.exponential(1 / mu)
119
120
121 #####
122 @classmethod
123 def geometric(cls, p: float, which_stream: Stream) -> numpy.int64:
124     """ class-level method to generate integer values drawn from a geometric(p)
125     distribution, where p corresponds the probability of success on an
126     individual trial (see numpy.Generator.geometric)
127
128     Parameters:
129         p: floating-point probability of success on any single trial
130         which_stream: named entry from Stream class
131
132     Returns:
133         a geometric(p) distributed integer value, corresponding to the
134         number of failures before the first success
135     """
136 if not cls._initialized:
137     cls._initializeStreams()
138 generator = cls._streams[which_stream.value]
139
140 # according to the API for numpy.Generator.geometric, it models the
141 # number of _trials_ until the first success, rather than the number of
142 # _failures_ before the first success as R does... so we subtract 1
143 # https://bit.ly/numpy_Generator_geometric
144 # https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Geometric.html
145 #return generator.geometric(p)
146 return generator.geometric(p) - 1 # see comment above
147
148 @classmethod
149 def gamma(cls, shape: float, scale: float, which_stream: Stream) -> np.float64:
150     """Generate a random float from a gamma distribution with parameters shape and scale.
151
152     Parameters:
153         shape (float): The shape parameter, also known as the alpha parameter.
154         scale (float): The scale parameter, also known as the beta parameter.
155         which_stream (Stream): Enum value specifying which stream to draw the random number from.
156
157     Returns:
158         np.float64: A random float from a gamma distribution with parameters shape and scale.
159     """
160 if not cls._initialized:
161     cls._initializeStreams()
162 generator = cls._streams[which_stream.value]
163 return generator.gamma(shape, scale)
164 #####
```

## Variates Histogram

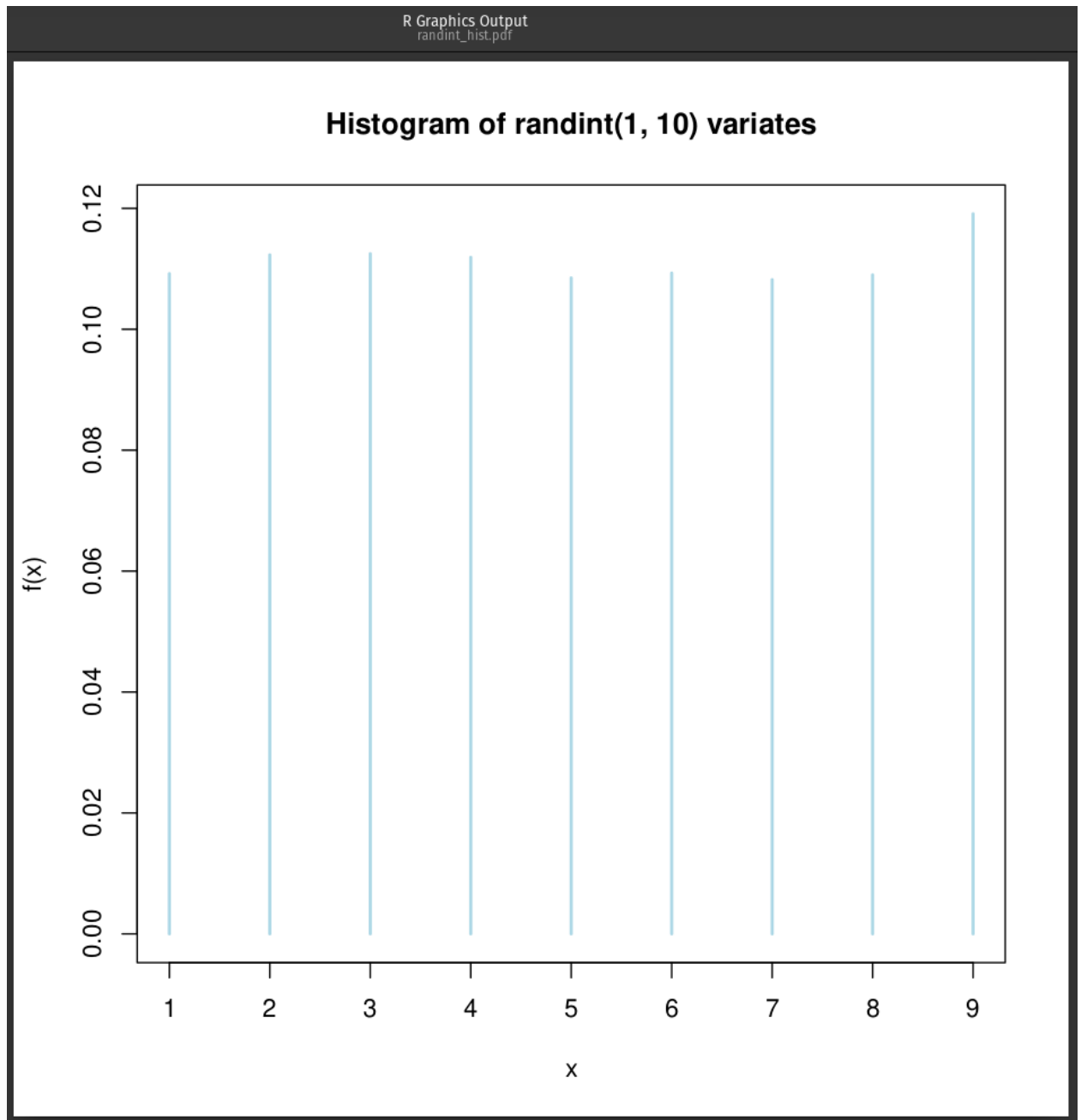
### 1. Geometric



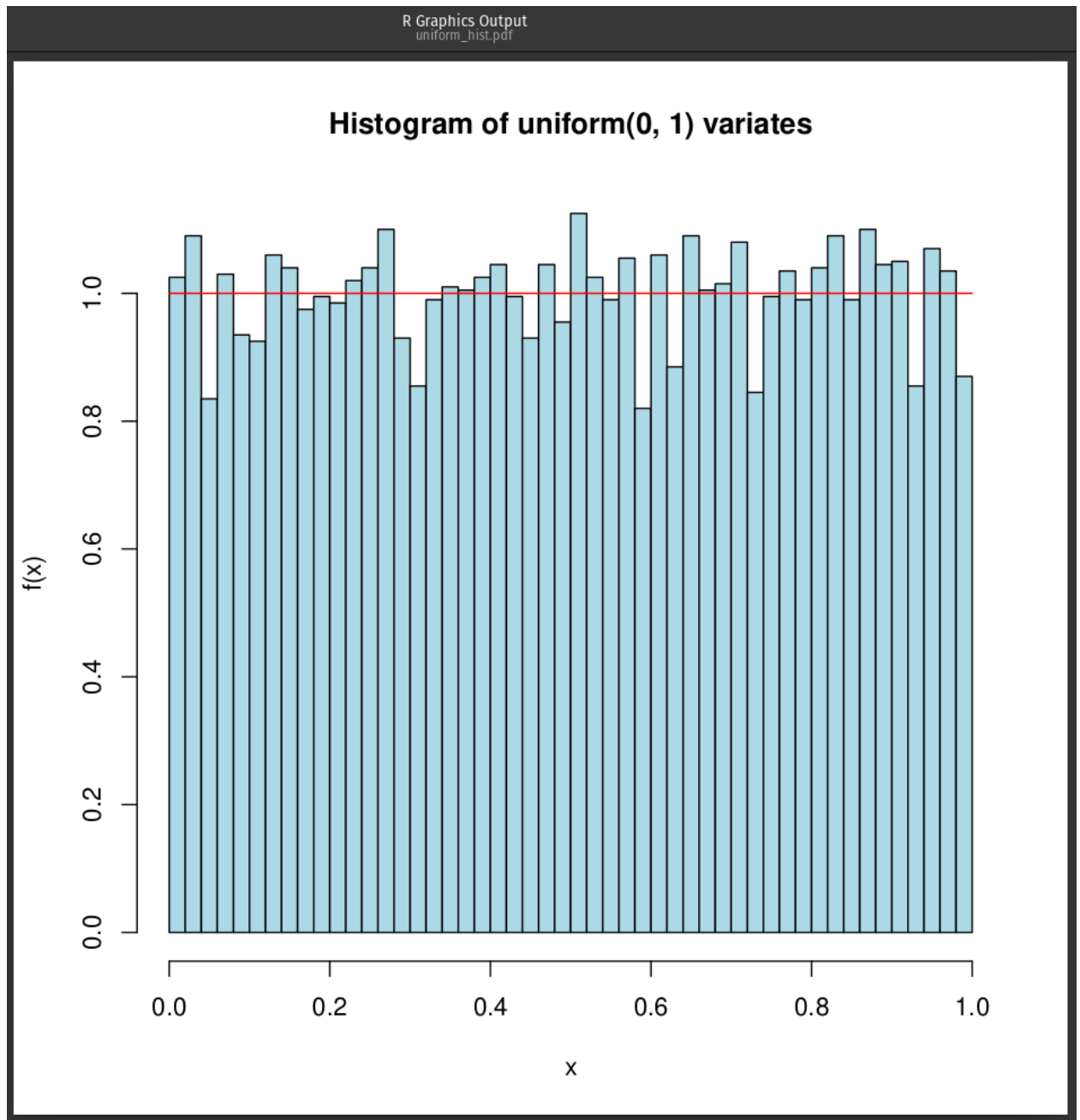
## 2. Random



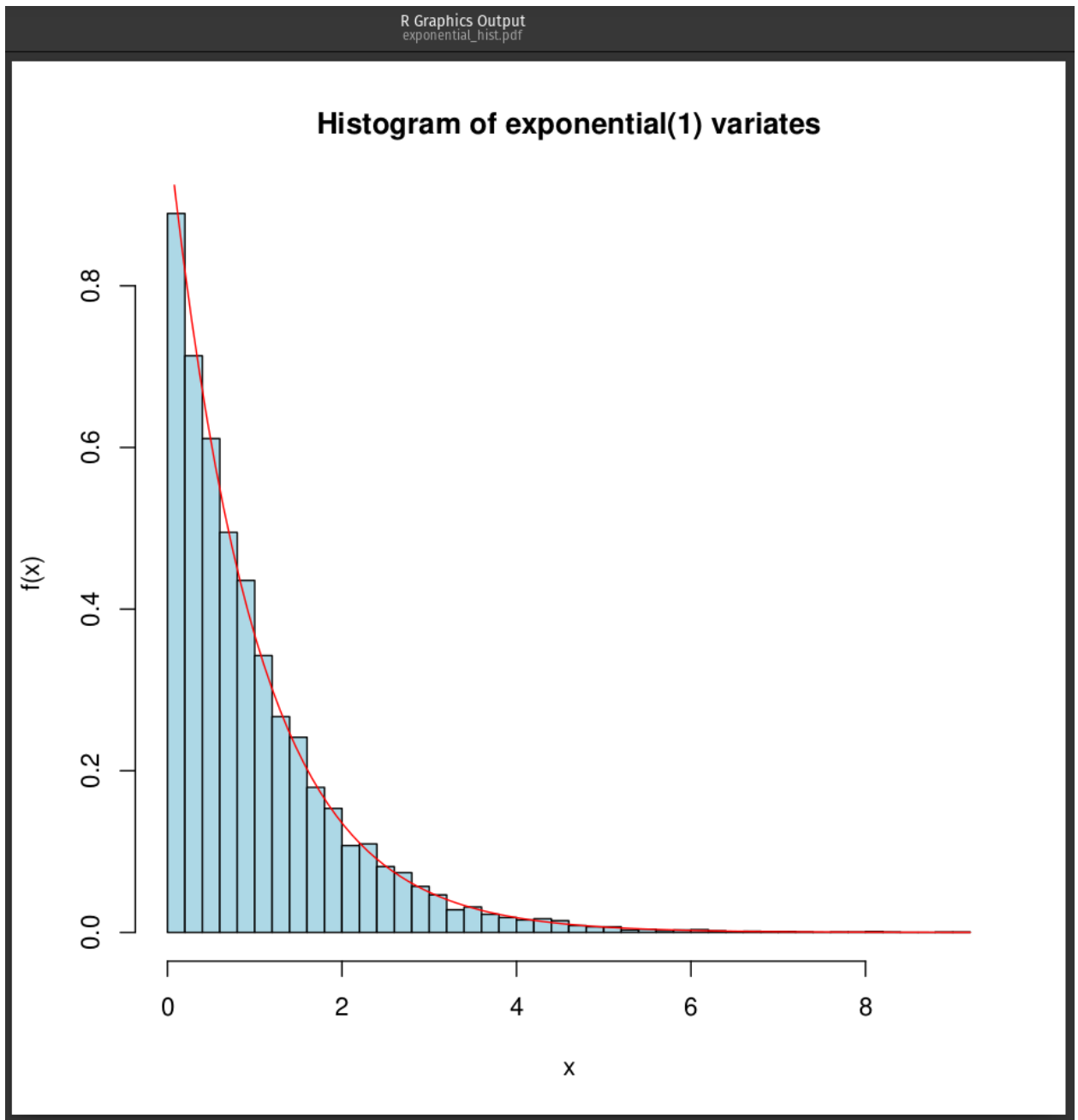
### 3. Randint



#### 4. Uniform

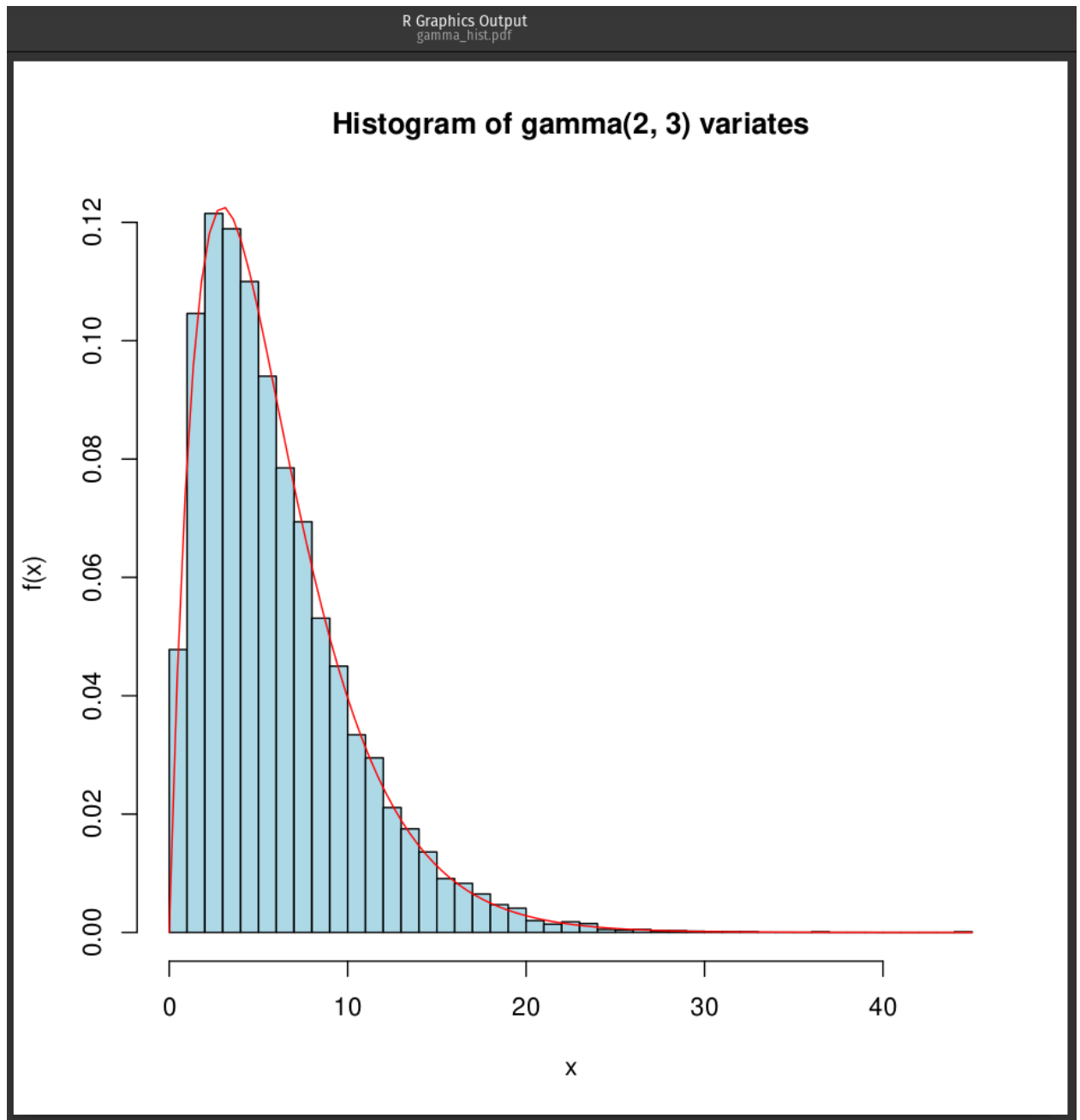


## 5. Exponential





## 6. Gamma



## Problem 2

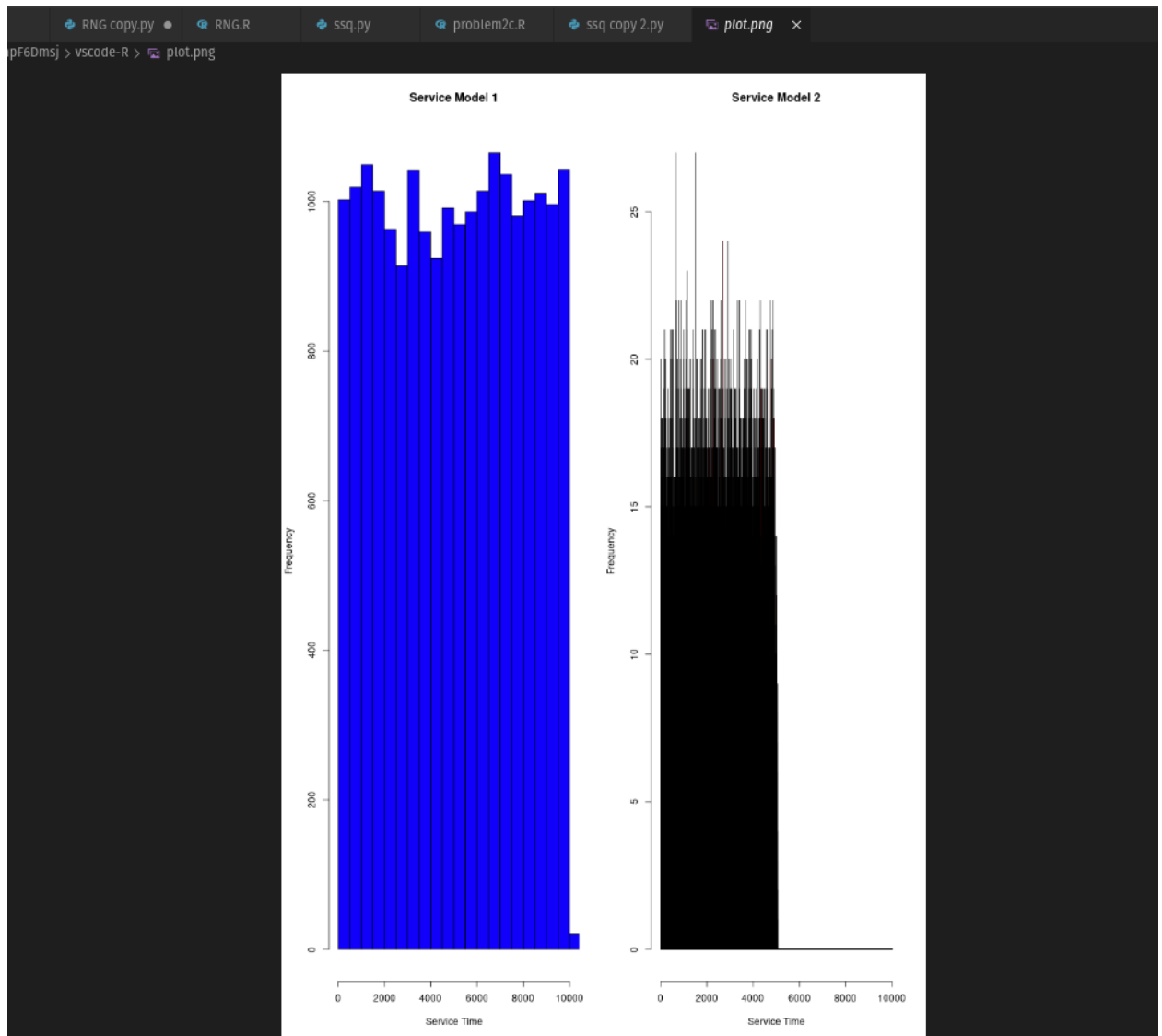
A. Based on approximately 1 000 000 processed customers:

Service Node Capacity	Probability of Rejection	Average Sojourn Time
1	0.5001	0.9442
2	0.3413	0.0714
3	0.2551	0.0863
4	0.2019	0.3941
5	0.1680	0.3887
6	0.1440	0.0027

B. Service task is changed to be uniform(0.1, 0.3):

Service Node Capacity	Probability of Rejection	Average Sojourn Time
1	0.5001	0.9442
2	0.3476	0.0714
3	0.2708	0.0863
4	0.2247	0.3941
5	0.1795	0.3887
6	0.1594	0.0027

C.



Service Model 1 represents a scenario where the service times are consistently around a specific value, leading to a more predictable queuing system. This model would be appropriate if the service times for tasks are relatively uniform and follow a known pattern.

Service Model 2 represents a more varied range of service times, possibly indicating that tasks have different complexities, and the service provider has varying levels of efficiency. This model might be more realistic in situations where tasks are diverse and the server's performance is influenced by external factors.

The appropriateness of the service models depends on the specific context of the single-server queuing system. If the service times are consistent and predictable, then Service Model 1 is more appropriate. However, if there is variability in the service times, then Service Model 2 would be more suitable.

#### D. *Probability of Rejection:*

The probability of rejection is the likelihood that an arriving customer is rejected because the system has reached its capacity. This probability is influenced by the service process in the following ways:

If the service process is slow, meaning it takes a long time to serve each customer, the system will be more likely to reach its capacity. As a result, the probability of rejection will increase.

Conversely, if the service process is faster, the system will have more capacity to serve new arrivals, reducing the probability of rejection.

In our simulation, the service process is determined by the time it takes to complete a series of tasks. If the tasks have a larger time range (i.e., tasks take longer to complete), the probability of rejection is likely to increase as it takes longer to serve each customer.

#### *Average Sojourn Time:*

The average sojourn time is the average time a customer spends in the system, including both waiting in the queue and being served. It is affected by the service process in the following ways:

If the service process is slow, customers will spend more time being served, increasing the average sojourn time.

If the service process is fast, the time spent being served will be shorter, reducing the average sojourn time.

In our simulation, the average sojourn time is influenced by the service process, which consists of a series of tasks. If the tasks have a larger time range or a higher probability of adding more tasks (i.e., lower probability in the geometric distribution), the average sojourn time will likely increase.

The probability of rejection and average sojourn time are influenced by the underlying service process. Faster service processes generally result in lower probabilities of rejection and shorter average sojourn times, while slower service processes tend to result in higher probabilities of rejection and longer average sojourn times.

#### E.

- Basic checks: Ensure that the total number of arrivals, departures, and rejections are consistent with the simulation parameters, and the sum of departures and rejections equals the total number of arrivals.

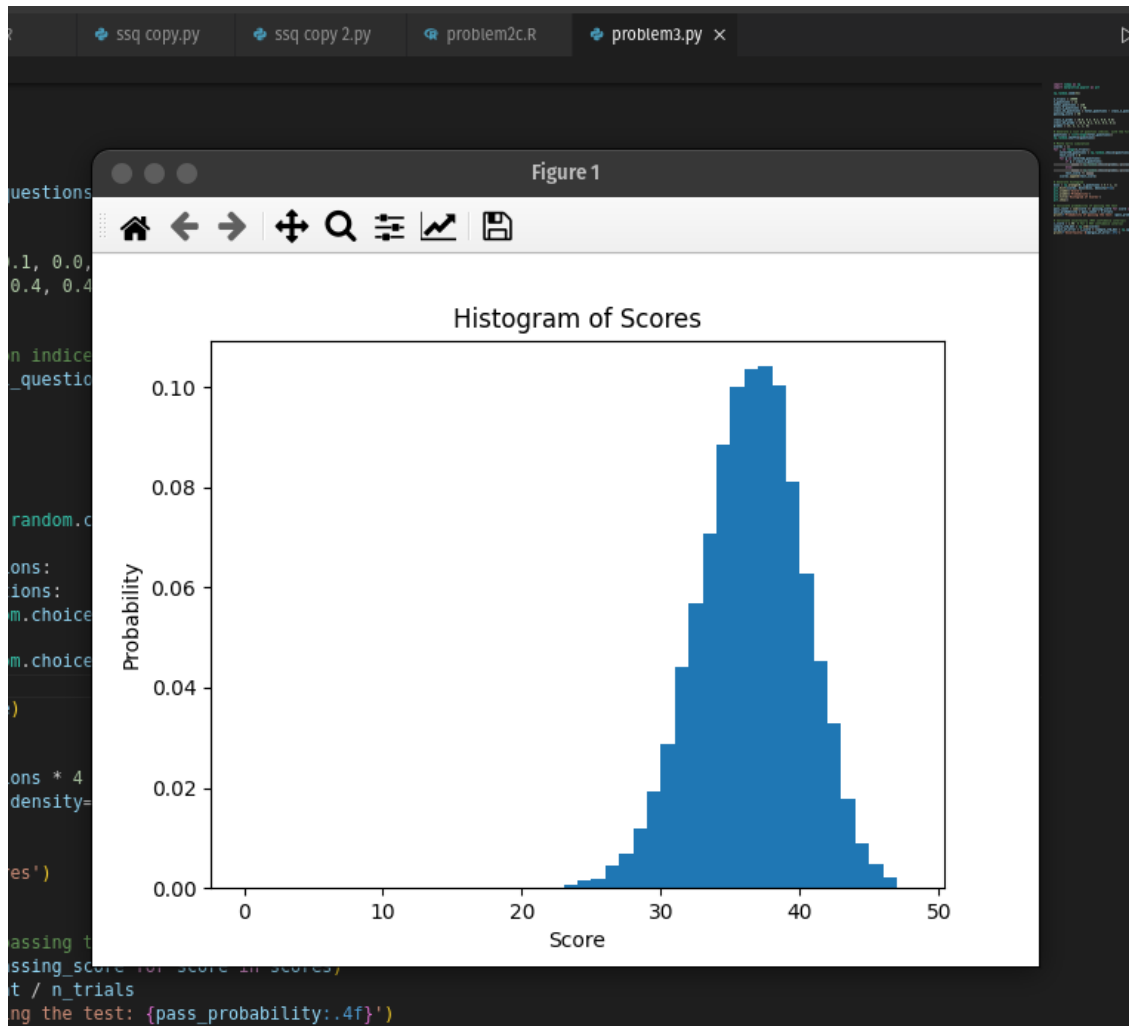
- Conservation equation: Verify if  $\bar{n} = \bar{q} + \bar{x}$ , where  $\bar{n}$  is the average number of customers in the system,  $\bar{q}$  is the average number of customers in the queue, and  $\bar{x}$  is the average number of customers in service. These statistics should be collected and computed independently during the simulation.
- Utilization: Ensure that the server utilization remains within the expected range (0 to 1) and is consistent with the arrival rate and service rate. Utilization should increase with a slower service process or higher arrival rate.
- Little's Law: According to Little's Law,  $\bar{n} = \lambda * \bar{W}$ , where  $\bar{n}$  is the average number of customers in the system,  $\lambda$  is the arrival rate, and  $\bar{W}$  is the average time a customer spends in the system (average sojourn time). We can verify if the results from our simulation adhere to Little's Law.
- Probability of rejection: When the capacity is 1 (server with no queue), the probability of rejection should be very high, close to 1, as any new customer arriving when the server is busy will be rejected. When the capacity is 2 (server with a queue length of 1), the probability of rejection should be lower than with a capacity of 1, as now the system can accommodate one customer in the queue. As the capacity increases, the probability of rejection should decrease.
- Monte Carlo analysis: Perform multiple runs of the simulation with different random seeds and ensure that the results are consistent across different runs. This will help confirm that the results are not an artifact of a particular random sequence.

## Problem 3

Steps:

1. Randomly select 12 questions from the pool of 120 questions.
2. For each question, determine its class (I or II) and generate a grade based on the given probabilities.
3. Sum the scores for the 12 questions to get the total score for that test.
4. Repeat this process a large number of times (e.g., 10,000 or more) to generate a histogram of scores.
5. Calculate the probability of passing the test based on the histogram.
6. Estimate the uncertainty in the result.

A.




B. The Probability of passing the test: 0.5643

Based on the Monte Carlo simulation I ran, the probability of passing the test is 0.5643, or 56.43%. This means that given the distribution of Class I and Class II, there was 56.43% chance of scoring 36 or higher on the test, which is the requirement to pass.

C. Uncertainty:  $\pm 0.0736$

The uncertainty of the result,  $\pm 0.0736$ , is a measure of the precision of the estimated probability. This uncertainty is calculated using a 95% confidence interval, which means that we can be 95% confident that the true probability of passing the test lies within the range of (0.5643 - 0.0736) to



( $0.5643 + 0.0736$ ), or approximately between 0.4907 and 0.6379. This range provides an indication of how much the probability might vary if we were to repeat the Monte Carlo simulation with a different random seed or under slightly different conditions.

Based on the study and classification of the questions, there was a 56.43% chance of passing the test. However, there is some uncertainty in this estimate, and the true probability could lie within the range of 49.07% to 63.79%.