EE-325
Programming Assignment-2

September 07, 2024

Name	Roll Number
Bhavishya Singh Premi	23B1230
Harsha Sai Srinivas	23B1202
Yaswanth Ram Kumar	23B1277

Question 1

Let the total number of fish in the lake be denoted by n. Suppose we catch m fish, mark them, and release them back into the lake. After this process, there are m marked fish and $n-m$ unmarked fish in the lake. We then allow the fish to mix well and catch m fish again. Out of these, p are those that were marked earlier. We assume that the actual fish population in the lake is n and remains unchanged between the two catches.

Let $P_{m,p}(n)$ represent the probability of the event A, where p marked fish are recaptured from a total of n fish in the lake. The event A occurs if we select p marked fish from m marked fish, and $m-p$ unmarked fish from $n-m$ unmarked fish. This can be done in $\binom{m}{p}$ ways for marked fish and $\binom{n-m}{m-p}$ ways for unmarked fish. The total number of ways to select m fish from the lake is $\binom{n}{m}$.

Thus, the formula for $P_{m,p}(n)$ is:

$$P_{m,p}(n) = \frac{\binom{m}{p} \cdot \binom{n-m}{m-p}}{\binom{n}{m}} \tag{1}$$

Plot for $P_{m,p}(n)$

We will plot $P_{m,p}(n)$ as a function of n for the following values of m and p:

- $m = 100$, $p = 10$
- $m = 100$, $p = 20$
- $m = 100$, $p = 50$
- $m = 100$, $p = 75$

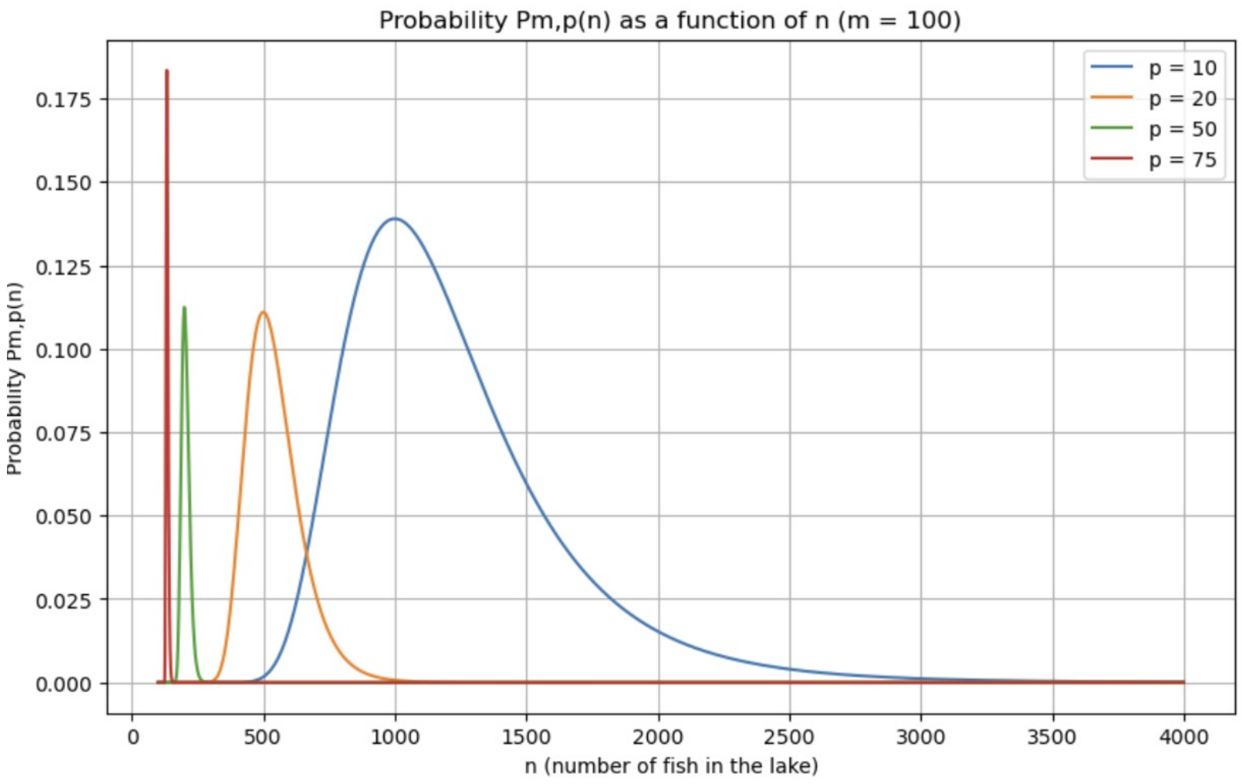


Figure 1: Plot for $P_{m,p}(n)$

Note: Programming files have been attached below.

Yaswanth's Notion of Best Guess

Yaswanth suggests that the best guess for the actual fish population n is the value n_0 for which $P_{m,p}(n)$ (for fixed p) is maximized. In other words, for this value n_0, the probability that we recapture p marked fish is the highest, making n_0 the most likely population size.

Based on this notion, Yaswanth's estimates for n are as follows:

- Best value of n for $p = 10$ is $n_0 = 999$
- Best value of n for $p = 20$ is $n_0 = 499$
- Best value of n for $p = 50$ is $n_0 = 199$
- Best value of n for $p = 75$ is $n_0 = 133$

Harsha's Notion of Best Guess

Harsha suggests that the best guess for the actual fish population n is the expected value n_1, which is calculated using the formula:

$$n_1 = \sum_n n \cdot P_{m,p}(n) \tag{2}$$

This gives the average number of fish in the lake based on the probability $P_{m,p}(n)$ for a fixed p. Since the expected value may not be an integer, Harsha Sai rounds it to the nearest integer to make the best guess.

Harsha Sai's estimates for n are as follows:

- Expected value of n for $p = 10$ is $n_1 = 1223$
- Expected value of n for $p = 20$ is $n_1 = 544$
- Expected value of n for $p = 50$ is $n_1 = 204$
- Expected value of n for $p = 75$ is $n_1 = 134$

Question 2

Consider a discrete-time system where packets arrive randomly at a router to be transmitted over a link. The time between successive packet arrivals is modeled as independent geometric random variables with parameter λ. The probability that the time between successive packet arrivals is k is given by:

$$P(\text{time between successive packets} = k) = (1-\lambda)^{k-1} \cdot \lambda \tag{3}$$

This probability is always less than or equal to λ for any k. Therefore, at any particular timestamp, if a random number generated is less than or equal to λ, a packet is added to the queue.

The packet transmission times are also modeled as independent geometric random variables with parameter μ. Only one packet can be transmitted at a time, and packets that arrive during the transmission of a previously arrived packet are stored in a buffer. The probability that the transmission time is k is given by:

$$P(\text{packet transmission time} = k) = (1-\mu)^{k-1} \cdot \mu \tag{4}$$

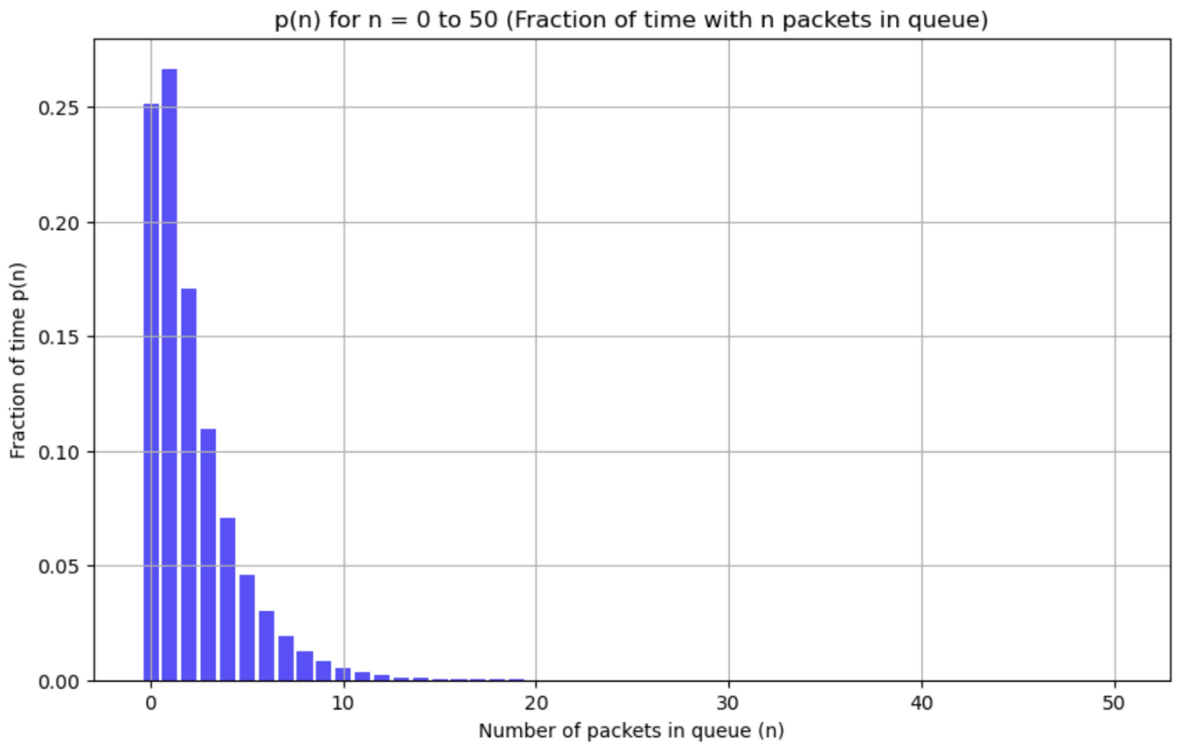
This probability is always less than or equal to μ for any k. Thus, at any particular timestamp, if the random number generated is less than or equal to μ, a packet is removed from the queue, provided the queue is non-empty. The system assumes infinite memory, meaning the queue can hold any number of packets.

Simulation

At the beginning of each second:

- If the queue is non-empty, a packet leaves the queue with probability μ.
- A new packet is added to the queue with probability λ.

We simulate this process for $1,000,000$ timestamps with $\lambda = 0.3$ and $\mu = 0.4$. For $n = 0, 1, 2, \ldots, 50$, let $p(n)$ represent the fraction of time there are exactly n packets in the queue.



```
Time average of number of packets in the memory:2.119088
```

Figure 2: Simulation results

Time Average of Number of Packet

The time average of the number of packets in memory is calculated by summing the number of packets in the queue at the end of each timestamp and dividing this sum by the total number of timestamps, $1,000,000$. The time average of the number of packets in the memory from the simulation is:

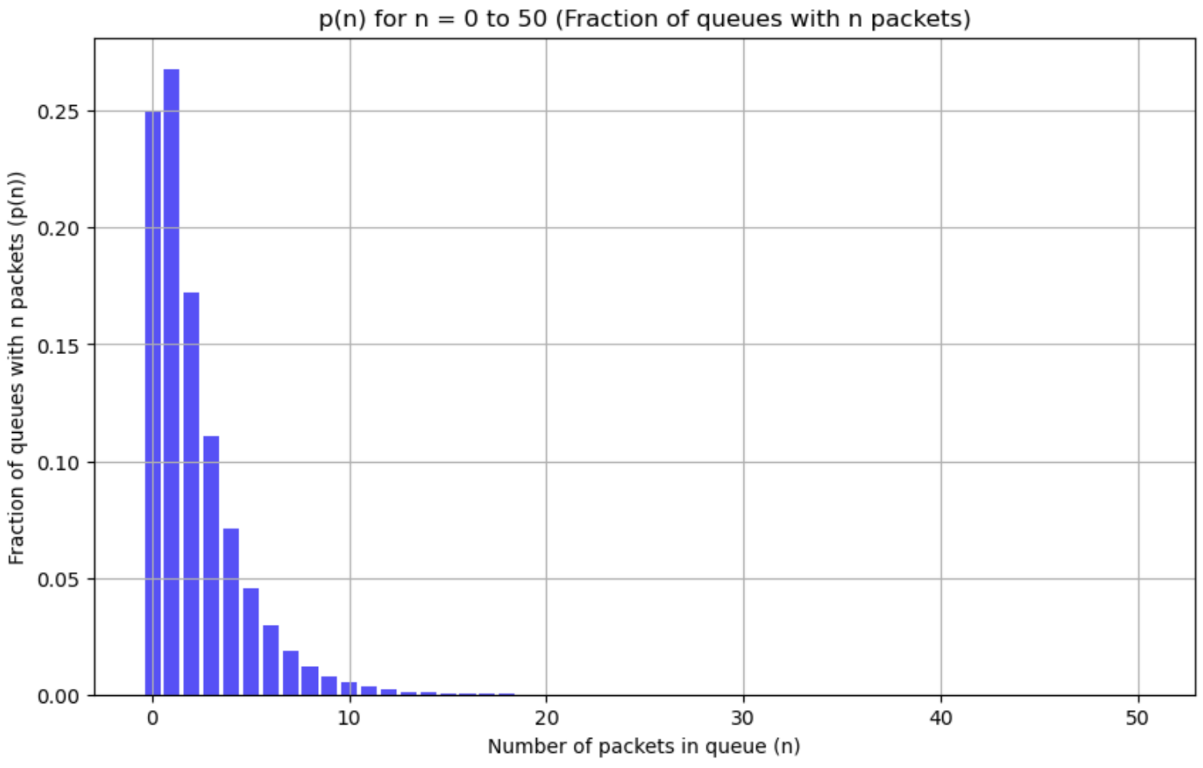
$$\text{Time average of number of packets} = 2.119088 \tag{5}$$

Question 3

We extend the simulation from the previous part to model $10,000$ queues running in parallel for $100,000$ time steps. For each queue, we track the number of packets in the system at the end of the simulation. This data is used to compute $p(n)$, the fraction of queues that have n packets in the system, and to calculate the sample average of the number of packets.

Plot of $p(n)$

For each n, $p(n)$ represents the fraction of the $10,000$ queues that have exactly n packets at the final time step. We use this data to generate a plot of $p(n)$, showing the distribution of packet counts across the $10,000$ queues.



```
Sample Average Queue Size: 2.1105
```

Figure 3: Plot of p(n)

Sample Average of Number of Packets

The sample average of the number of packets in the system is calculated as:

$$\text{Sample Average(number of packets)} = \frac{1}{10,000} \sum_{i=1}^{10,000} (\text{number of packets in queue } i) \tag{6}$$

From the simulation, we found that the sample average of the number of packets in the system for the $10,000$ queues is:

$$\text{Sample Average} = 2.1105 \tag{7}$$

Question 4

From a population of countably infinite people, we select N people, the probability that each one of them makes the correct(fair) decision has a probability $(0.5 + c)$, with $0.05 \leq c \leq 0.25$. Identifying this to be a problem of Binomial distribution. We want combinations of c and N to be such that probability of correct decision is atleast $P_{atleast}$.

$$P_{fair}(c,N) = \sum_{i=\frac{N+1}{2}}^{N} \binom{N}{i} (0.5+c)^i (0.5-c)^{N-i} \tag{8}$$

$$P_{fair}(c,N) \geq P_{atleast} \tag{9}$$

We have analyzed the behavior of Equation 8 by varying c by steps of 0.01 and N through 1 to 59 for $P_{atleast} = 0.75$ and $P_{atleast} = 0.90$.

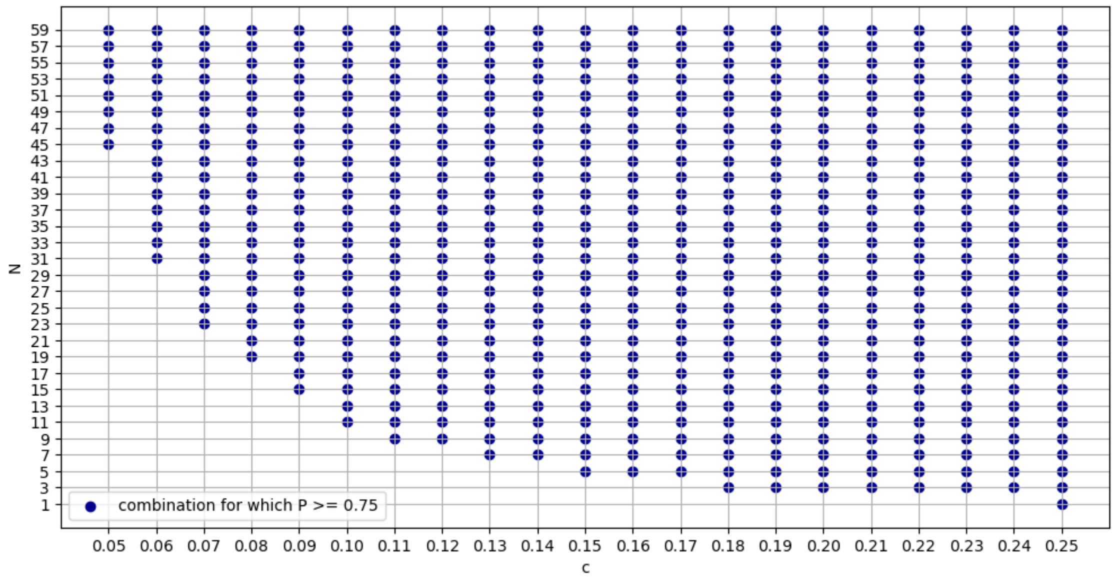


Figure 4: Plot of c vs N for $P_{atleast} = 0.75$

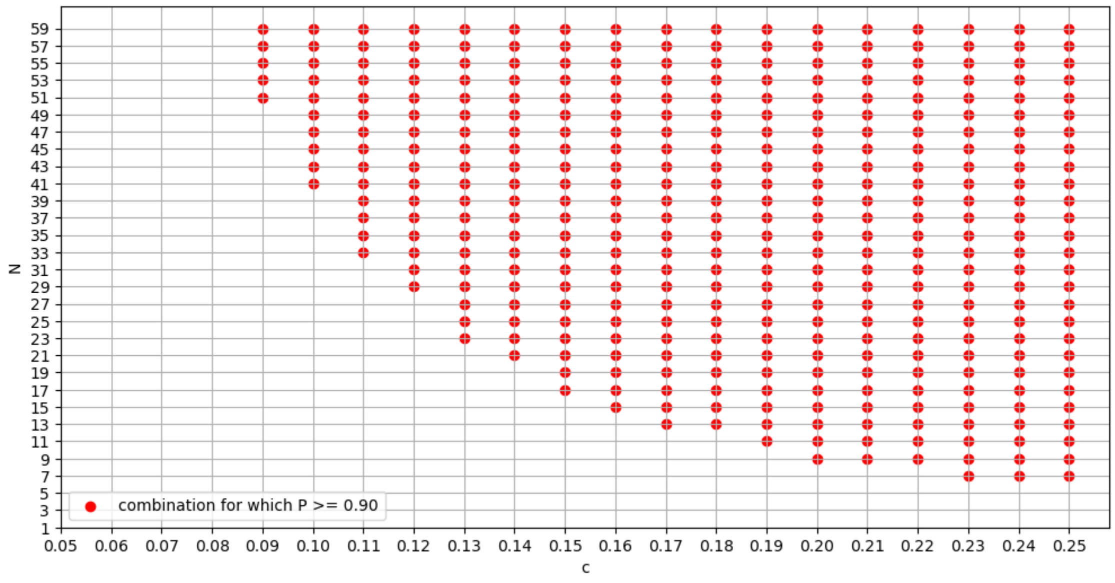


Figure 5: Plot of c vs N for $P_{atleast} = 0.90$

Findings

- We have observed that for a given N, there is a minimum required c for every one of that N people for probability of correct decision to be greater than or equal to $P_{atleast}$.
- Similarly, for a given c, we need a atleast some N for that c, such that $P_{atleast}$ is achieved.
- We tried to deduce a closed form for probability of being fair from *Equation* 8, but it wasn't possible.
- We observed that *Equation* 8 is increasing with respect to both c and N, it is obvious, an illustration of this is depicted below.

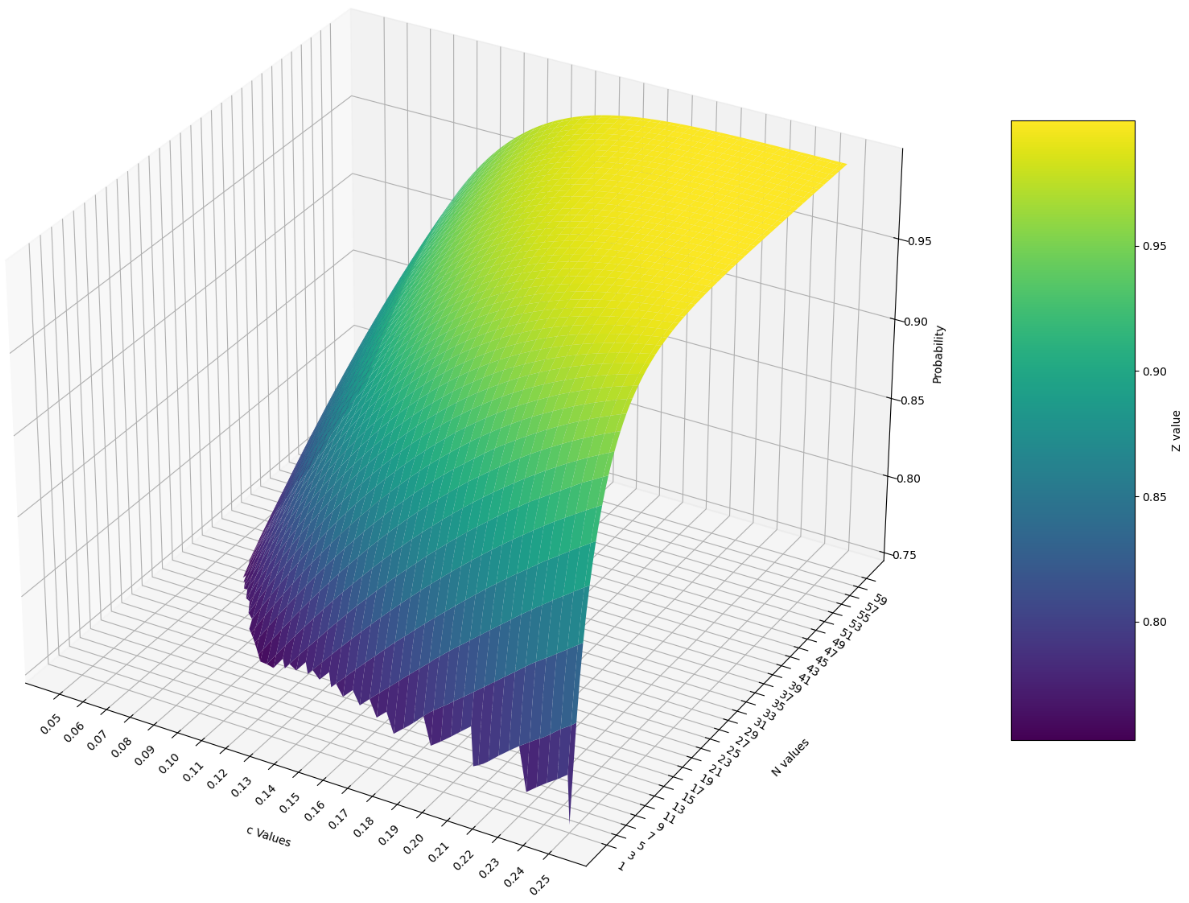


Figure 6: Probabilities of the N and c combinations with $P_{fair}(c, N) \geq 0.75$

Further Analysis and Creating an Objective Function

Selecting a person with *higher* c is costly and difficult, also if we select more and more number of people with *lesser* c, then management of logistics would become complex. So, we are facing a trade-off between c and N. This trade-off can be minimised by modelling an objective function $J_{P_{atleast}}(c, N)$. This objective function will give us an optimum minimum values of c and N and yet satisfy *Equation* 9.

But for this we have to model the cost of picking a juror with some c, also we have to quantify the management complexities as N increases. For this purpose we took the cost to increase exponentially with c, but it could be anything depending on our situation, it could be linear, total cost is the product of N with cost of a single person with c. We quantified the management complexity by taking the t_{poll} (*time taken*) for polling, by all N jurors, this objective function is just for an example.

$\therefore$, The two components of the Objective function are:

- $J_r = Nr_0e^{r_s(c-0.05)}$,where r_0 and r_s are positive real constants.
- $J_t = Nt_0$, where t_0 is the time taken for polling by a single person.

Depending on the values of the above parameters, our preferences, we can model our objective function $J_{P_{atleast}}(c, N)$. Let us take a simple case by just adding bot J_r and J_t, but this is not always the best function, in fact there is no best function in this scenario, it will depend on the values of the parameters, what are we going to give up over what, is something that takes us to the path of building the best objective function.

$$J_{P_{atleast}}(c, N) = J_r + J_t \tag{10}$$

Now we minimise the above objective function by taking derivative with respect to both c and N.

$$\hat{c}, \hat{N} = \arg\min_{c,N} J_{P_{atleast}}(c, N) \tag{11}$$

The optimum $\hat{c}$, $\hat{N}$ should also satisfy the *Equation* 9, hence by solving *Equation* 9 and *Equation* 11, we could get the optimal c and N.

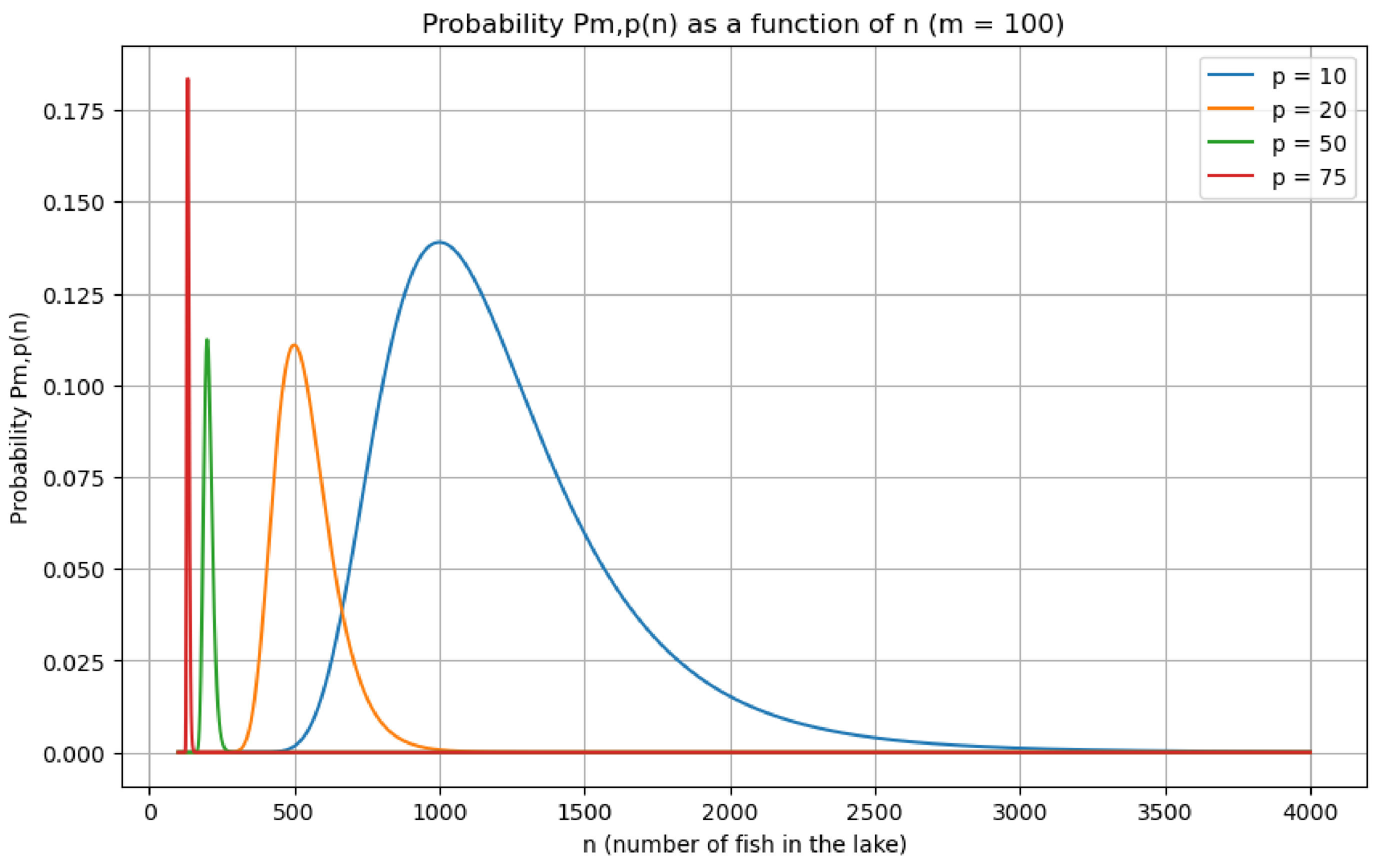
...Thank You

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

In [2]:
```
#Function to find Pm,p(n)
def Pm_p(n,m,p):
    return (math.comb(m,p)*math.comb(n-m,m-p))/math.comb(n,m)
```

In [3]:
```
m=100
p_val=[10, 20, 50, 75]
n_val=np.arange(100, 4000)  #let number of fish in the pond be in the range 100 to 5000
#using the above defined function to calculate probabilities
probabilities = {p: [Pm_p(n, m, p) for n in n_val] for p in p_val}
```

In [4]:
```
plt.figure(figsize=(10, 6))
for p in p_val:
    plt.plot(n_val, probabilities[p], label=f'p = {p}')

plt.title('Probability Pm,p(n) as a function of n (m = 100)')
plt.xlabel('n (number of fish in the lake)')
plt.ylabel('Probability Pm,p(n)')
plt.legend()
plt.grid(True)
plt.show()
```



In [5]:
```
# Function to find the best value of n for each p (the one with max probability)
def best_n_for_p(p):
    index_of_max_prob = np.argmax(probabilities[p])  # Find the index of the max probabi
    return n_val[index_of_max_prob]  # Corresponding n value

# Loop through the values of p and find the best value of n for each
for p in p_val:
    print(f"Best value of n for p={p} is {best_n_for_p(p)}")
```

```
Best value of n for p=10 is 999
Best value of n for p=20 is 499
Best value of n for p=50 is 199
Best value of n for p=75 is 133
```

In [6]:

```python
# Function to calculate the expected value of n for each p (weighted average)
def n_avg(p):
    probabilities_sum = np.sum(probabilities[p])
    return np.sum(n_val* probabilities[p]) / probabilities_sum  # Weighted average

# Loop through the values of p and calculate the expected value of n for each
for p in p_val:
    expected_n = n_avg(p)
    print(f"Expected value of n for p={p} is {expected_n:.0f}")
```

```
Expected value of n for p=10 is 1223
Expected value of n for p=20 is 544
Expected value of n for p=50 is 204
Expected value of n for p=75 is 134
```

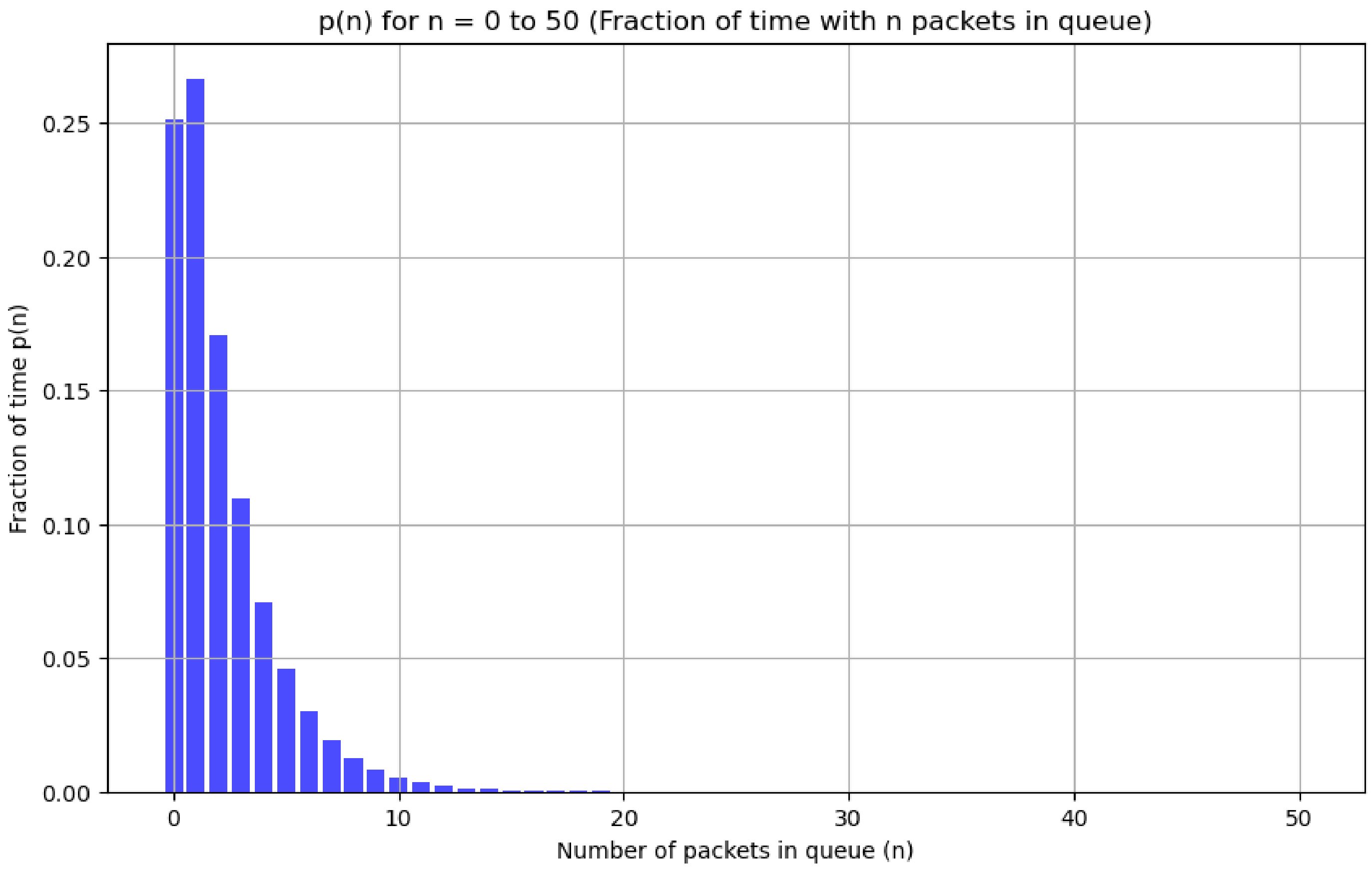
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: #Assigning values to variables
        lamda=0.3  # probability of packet arrival in next time step
        mu=0.4     # probability of packet departure in next time step
        max_queue_size=50
        time_steps=1000000
```

```
In [3]: # Initializing queue and tracking variables
        queue_size = 0
        total_queue_size = 0 #for computing time average
        queue_counts = np.zeros(max_queue_size + 1)
```

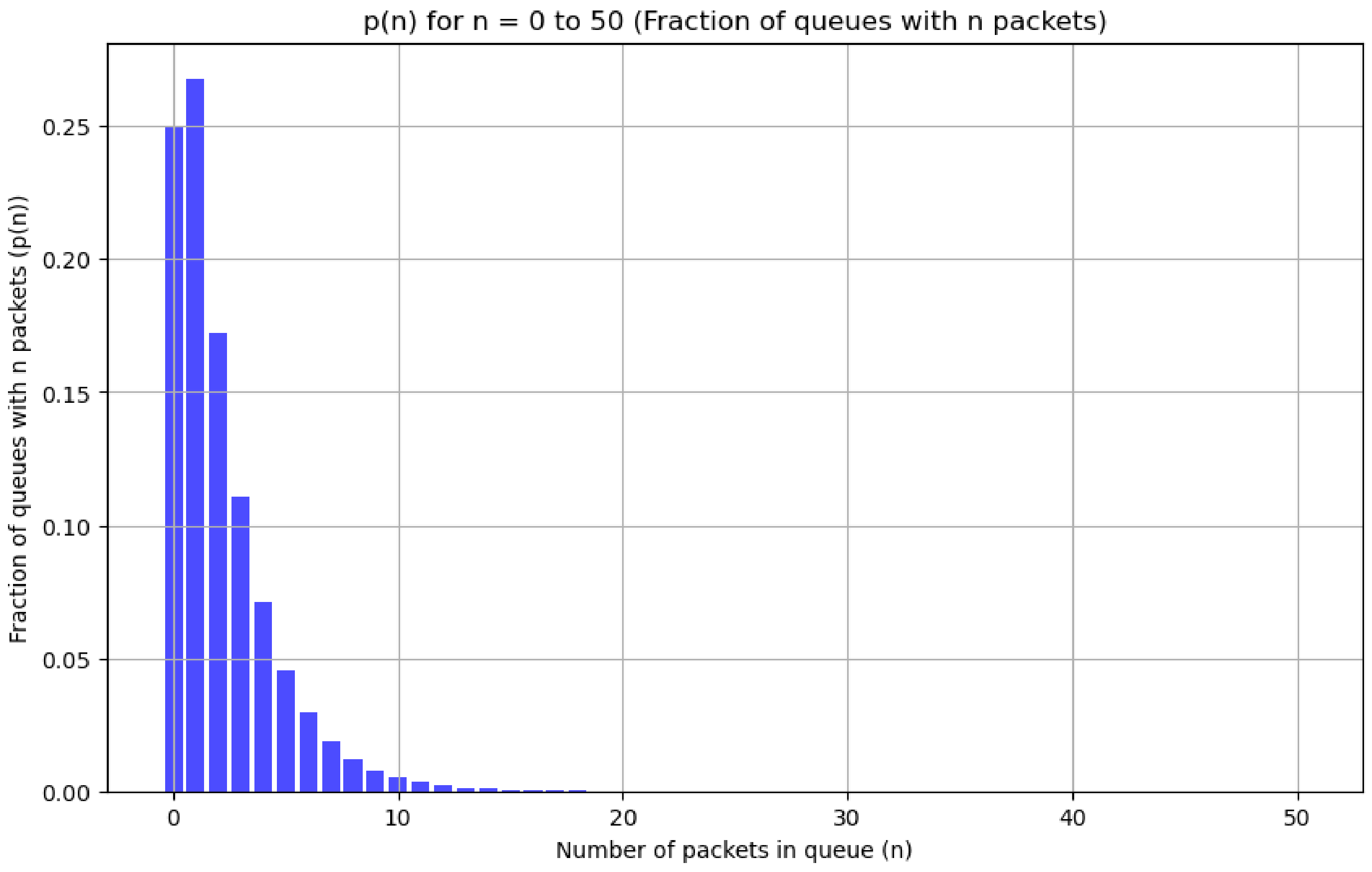
```
In [4]: # Simulating this in loop
        for _ in range(time_steps):
            # Departure: If the queue is not empty(atleast 1 packet exists), a packet departs wi
            if queue_size > 0 and np.random.rand() < mu:
                queue_size -= 1
            # Arrival: A new packet arrives with probability lambda
            if np.random.rand() < lamda:
                queue_size += 1
            # Updating the queue count for the current queue size
            queue_counts[queue_size] += 1
            # Tracking total queue size for time-average computation
            total_queue_size += queue_size
            # Computing p(n) = fraction of time that there are n packets in the queue
            p_n = queue_counts / time_steps
```

```
In [6]: # Computing time-averaged number of packets in the memory
        time_average_queue_size = total_queue_size / time_steps

        # Plotting p(n) for n = 0, ..., 50
        plt.figure(figsize=(10, 6))
        plt.bar(range(max_queue_size + 1), p_n, color='blue', alpha=0.7)
        plt.xlabel('Number of packets in queue (n)')
        plt.ylabel('Fraction of time p(n)')
        plt.title('p(n) for n = 0 to 50 (Fraction of time with n packets in queue)')
        plt.grid(True)
        plt.show()

        # printing the time-averaged number of packets in the memory
        print(f"Time average of number of packets in the memory:{time_average_queue_size}")
```



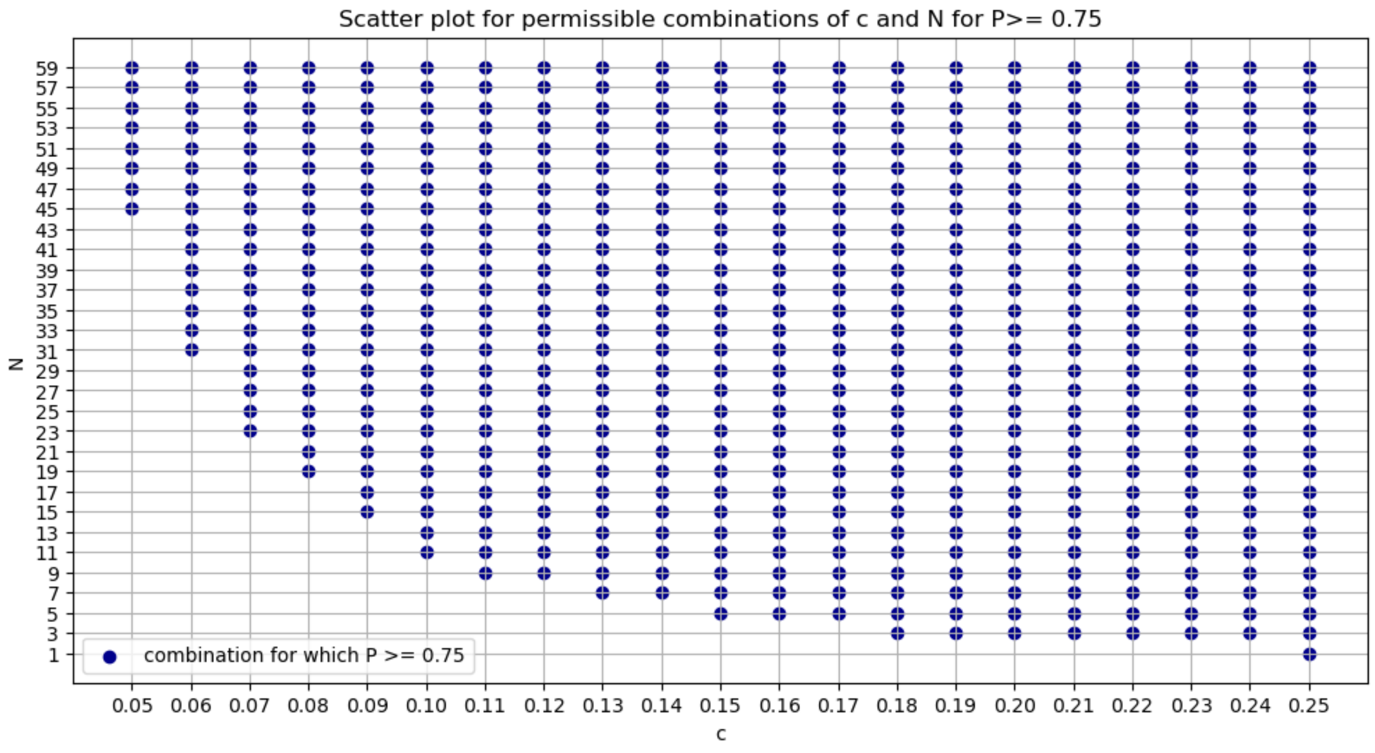
```
Time average of number of packets in the memory:2.119088
```

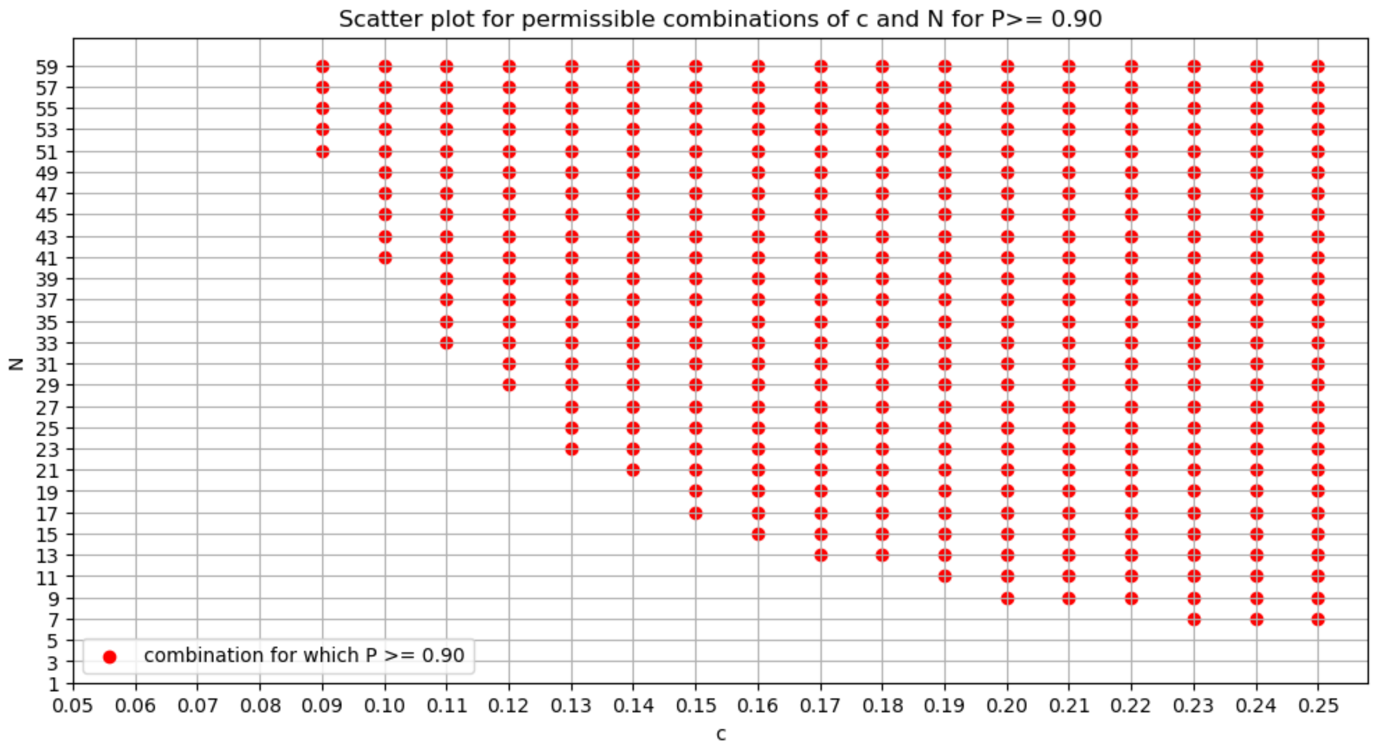
In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

lam = 0.3
mu = 0.4
t_steps = 100000 #Number of time steps
n_queues = 10000 #Number of parallel queues
max_size=50
#Here we intialised queue sizes to be 0
qs = np.zeros(n_queues, dtype=int)
qc = np.zeros(max_size + 1)

#Simulating loop for 10,000 time steps
for _ in range(t_steps):
    # Departures: packets leave with probability mu (if queue size > 0)
    dep=(np.random.rand(n_queues)<mu)&(qs>0)
    qs-=dep
    # Arrivals: new packets arrive with probability lambda
    arr=np.random.rand(n_queues)<lam
    qs+=arr
    #Updating the queue count based on current queue size
    for size in qs:
        qc[size] +=1

#Computing p(n)= fraction of queues that have n packets in the queue
p_n=qc/(n_queues*t_steps)
#Computing the sample average
avg_size= np.mean(qs)

#Plotting p(n) for n=0,1,....,50
plt.figure(figsize=(10, 6))
plt.bar(range(max_size+1),p_n,color='blue',alpha=0.7)
plt.xlabel('Number of packets in queue (n)')
plt.ylabel('Fraction of queues with n packets (p(n))')
plt.title('p(n) for n = 0 to 50 (Fraction of queues with n packets)')
plt.grid(True)
plt.show()
print("Sample Average Queue Size:", avg_size)
```



```
Sample Average Queue Size: 2.1105
```

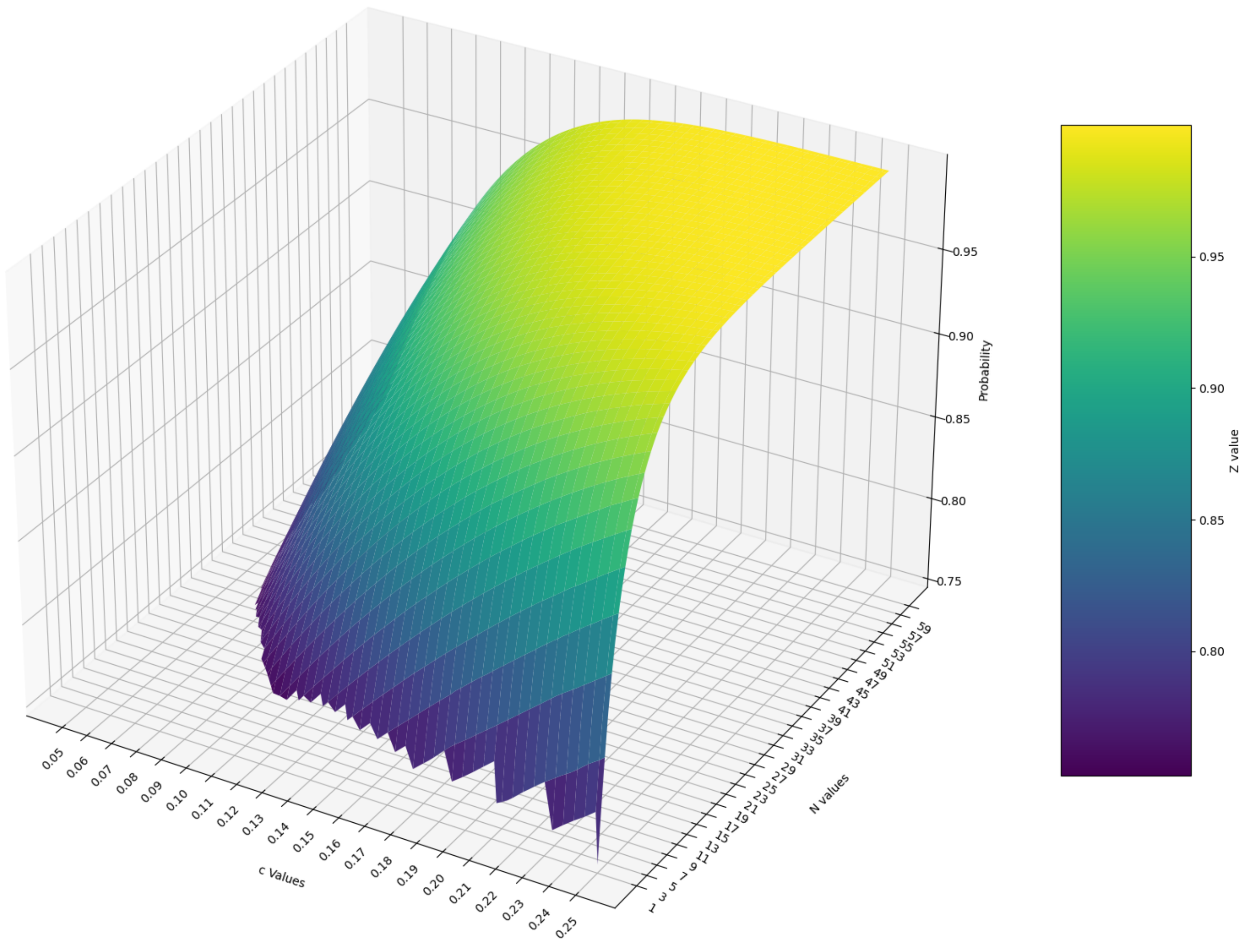
```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import math

        pd.options.display.float_format = '{:.10f}'.format

        # Define the Probalilty for correct function
        def PMF(c, N):
            pmf_value = 0
            p = 0.5+c
            # Calculate the sum from r = (N+1)//2 to N
            for r in range((N+1)//2, N+1):
                binomial_coeff = math.comb(N, r)  #N choose r
                term = binomial_coeff * (p**r) * ((1 - p)**(N - r))
                pmf_value += term

            return pmf_value

        T = 60 #took this value after observing data for multiple larger N.
        c_values = np.arange(0.05, 0.26, 0.01)
        N_values = [i for i in range(1, T+1) if i % 2 != 0]

        df = pd.DataFrame(index=N_values, columns=c_values)
        for N in N_values:
            for c in c_values:
                df.at[N, c] = PMF(c, N)

        threshold_75 = 0.75
        c_scatter_75 = []
        N_scatter_75 = []
        prob_75=[]

        threshold_90 = 0.90
        c_scatter_90 = []
        N_scatter_90 = []
        prob_90=[]

        for N in N_values:
            for c in c_values:
                if df.at[N, c] >= threshold_75:
                    c_scatter_75.append(c)
                    N_scatter_75.append(N)
                    prob_75.append(df.at[N, c])

        for N in N_values:
            for c in c_values:
                if df.at[N, c] >= threshold_90:
                    c_scatter_90.append(c)
                    N_scatter_90.append(N)
                    prob_90.append(df.at[N, c])

        plt.figure(figsize=(12, 6))
        plt.scatter(c_scatter_75,N_scatter_75,label='combination for which P >= 0.75',color='dar
        plt.title('Scatter plot for permissible combinations of c and N for P>= 0.75')
        plt.xlabel('c')
        plt.ylabel('N')
        plt.yticks(N_values)
        plt.xticks(np.arange(0.05, 0.26, 0.01))
        plt.grid(True)
        plt.legend()
        plt.show()

        plt.figure(figsize=(12, 6))
plt.scatter(c_scatter_90,N_scatter_90,label='combination for which P >= 0.90',color='red
plt.title('Scatter plot for permissible combinations of c and N for P>= 0.90')
plt.xlabel('c')
plt.ylabel('N')
plt.yticks(N_values)
plt.xticks(np.arange(0.05, 0.26, 0.01))
plt.grid(True)
plt.legend()
plt.show()

#print(df)
```





In [2]:
```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.colors import Normalize
from scipy.interpolate import griddata
```

```
fig = plt.figure(figsize=(20, 20))
ax = fig.add_subplot(111, projection='3d')
x = c_scatter_75
y = N_scatter_75
z = prob_75

xi = np.linspace(min(x), max(x), 100)
yi = np.linspace(min(y), max(y), 100)
xi, yi = np.meshgrid(xi, yi)

zi = griddata((x, y), z, (xi, yi), method='cubic')
surf = ax.plot_surface(xi, yi, zi, cmap='viridis', edgecolor='none')
cbar = plt.colorbar(surf, ax=ax, shrink=0.5, aspect=5)
cbar.set_label('Z value')
ax.set_xlabel('c Values', labelpad=20)
ax.set_ylabel('N values', labelpad=20)
ax.set_zlabel('Probability')
ax.set_xticks(np.arange(0.05, 0.26, 0.01))
ax.set_xticklabels([f'{tick:.2f}' for tick in np.arange(0.05, 0.26, 0.01)], rotation=45)
ax.set_yticks(N_values)
ax.set_yticklabels([str(int(label)) for label in N_values], rotation=-30)
plt.show()
```


c Values
N values
Probability
Z value