Hrishikesh Hippalgaonkar
hippalga@usc.edu

EE 511 Simulation Methods for Stochastic Systems Project #1

❖ Generating Discrete Random Variables:

▪ The Inverse Transform Method:

Let us assume we want to generate a discrete random variable that has a probability mass function given by

$$P(X = x_j) = p_j, \ \ x_j \in S, \ \ j = 1, 2, 3, 4 \dots \dots$$

where $S = \{ x_1, x_2, x_3, x_4, \dots . \}$ denotes the sample space, and $\sum_j p_j = 1$

This can be achieved by generating a random number U that is uniformly distributed over (0, 1) and setting X as

$$X = \begin{cases} x_0 & If\ U < p_0 \\ x_1 & If\ p_0 < U < p_0 + p_1 \\ \vdots & \vdots \\ x_j & If \sum_{i=0}^{j-1} p_i < U < \sum_{i=0}^{j} p_i \\ \vdots & \vdots \end{cases}$$

Since, for 0 < r < s < 1, $p\{ r \ \le U \ \le s\} = s - r$, we have that

$$p\{X = x_j\} = p\left\{\sum_{i=0}^{j-1} p_i < U < \sum_{i=0}^{j} p_i\right\} = p_j$$

Thus, we obtain X, which has the desired distribution.

1. Problem 1

[The three distributions are based on Bernoulli trials]

(a) Write a routine to simulate a fair Bernoulli trial. Generate a histogram for 100 simulated Bernoulli trials.

Approach: Using the inverse transform method, generate a Bernoulli trial. This can be implemented in the following way:

1. Generate uniform random variable U from $\mathrm{U} = (0, 1)$. In python, this can be done using the *random.random* command.
2. If $U \le p$, then $\mathrm{X} = 0$; else $\mathrm{X} = 0$. In this case p will 0.5 as it is a fair Bernoulli trial

Code:

```
import numpy as np
import random as rand
import matplotlib.pyplot as plt

def main():

    bernoulli_arr = np.empty([100])

    for i in range(0,100):
    # Return random floats in the half-open interval [0.0, 1.0).
    # Results are from the “continuous uniform” distribution over the stated
     # interval
        x = rand.random()
        if(x >= 0.5):
            bernoulli_arr[i] = 1
        else:
            bernoulli_arr[i] = 0

    plt.hist(bernoulli_arr, bins = 3)
    plt.title("Histogram of 100 fair Bernoulli trials", alpha = 0.75)

    plt.xlabel('Output of the Bernoulli Trial')
    plt.ylabel('# of trials')
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    main()
```

Result:

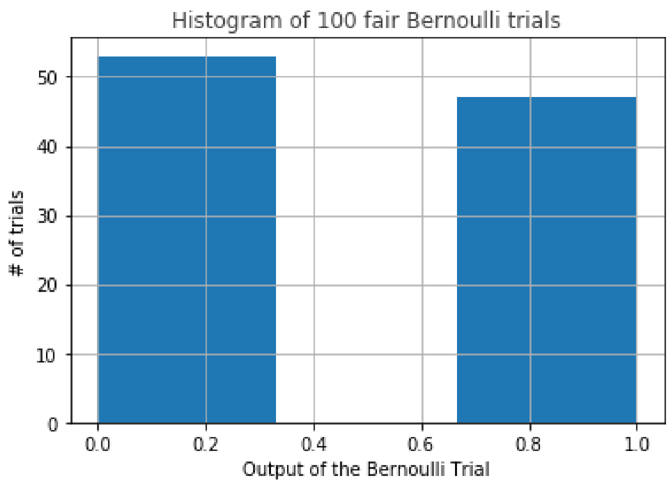

Figure 1.1 Histogram of 100 fair Bernoulli Trials

As this is a fair Bernoulli trial, the number of success and number of failures will be almost same.

(b) Write a routine to count the number of successes in 7 fair Bernoulli trials. Generate a histogram for 100 samples of this *success-counting* random variable.

Approach: The procedure is same as the above problem. Generate seven Bernoulli trials using the inverse transform method and count the number of successes in each trial. Let us call this a sample. Now we must repeat this for 100 samples and count the number of successes for each sample

Code:

```
import numpy as np
import random as rand
import matplotlib.pyplot as plt

def sevenBernoulliTrials():
    bernoulli_arr = np.empty([7])

    count = 0
    for i in range(0,7):
    # Return random floats in the half-open interval [0.0, 1.0).
    # Results are from the “continuous uniform” distribution over the stated
   # interval
        x = rand.random()
        if(x >= 0.5):
            bernoulli_arr[i] = 1
            count = count + 1
        else:
            bernoulli_arr[i] = 0

    return count
```

```
def main():

    success = np.empty([100])
    for i in range(0,100):
        success[i] = sevenBernoulliTrials()

    plt.hist(success, bins = 'auto')
    plt.title("Histogram of 100 samples of success - counting random
variable", alpha = 0.75)

    plt.xlabel('# of successes in 7 fair Bernoulli trials')
    plt.ylabel('# of samples')
    plt.grid(True)
    plt.savefig("Success_in_7_Bernoulli.jpg")
    plt.show()

if __name__ == "__main__":
    main()
```

Result:

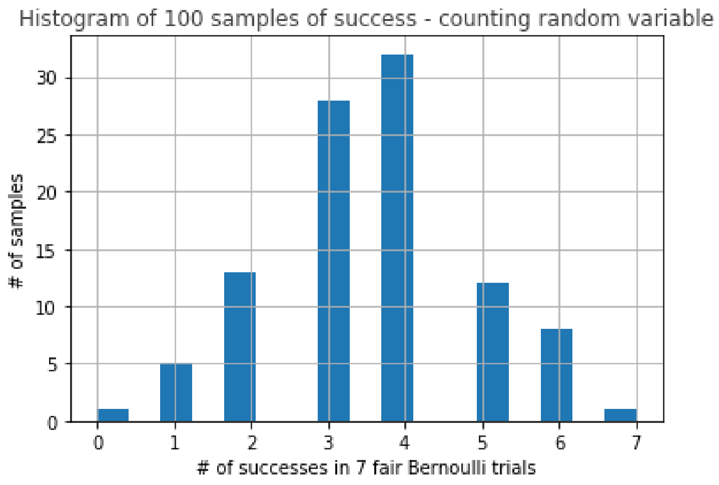

Figure 1.2: Histogram of 100 samples of success – counting random variable

The above random variable of *counting the number of success in seven Bernoulli trials* can also be represented as

$$S_n = X_1 + X_2 + X_3 + \cdots + X_7$$

where $X_i = 1 \text{ } or \text{ } 0$ according as the ith outcome is a success or failure. Thus, S_n is i.i.d sum of Bernoulli random variables and indicates the number of successes in seven trials. S_n has a binomial distribution $b(7,0.5,k)$, where k is the number of successes. However, according to Central Limit Theorem, the distribution curve of S_n (S_n being an i.i.d sum) approaches to a Gaussian random variable with mean given by *np* = 7x0.5 = 3.5 and variance given by *npq* = 7x0.5x0.5 = 1.75

Hence the above histogram looks almost like a Gaussian curve.

(c) Write a routine to count the longest run of heads in 100 Bernoulli samples. Generate a histogram for this random variable.

Approach: A run of heads can be defined as getting heads in consecutive trials, without getting any tails in between two heads. Using inverse transform method, we must generate 100 Bernoulli trials and count the longest run of heads (successes) in these 100 trials. This experiment must be repeated 400 times for generating the histogram.

Code:

```
import numpy as np
import random as rand
import matplotlib.pyplot as plt

def BernoulliTrials(k):
    bernoulli_arr = np.empty([k])

    count = 0
    for i in range(0,k):
    # Return random floats in the half-open interval [0.0, 1.0).
     # Results are from the "continuous uniform" distribution over the
    #stated interval
        x = rand.random()
        if(x >= 0.5):
            bernoulli_arr[i] = 1
            count = count + 1
        else:
            bernoulli_arr[i] = 0

    return count

def main():

    success = np.empty([300])
    for i in range(0,300):
        success[i] = BernoulliTrials(50)

    plt.hist(success, bins = 'auto')
    plt.title("Histogram of 300 samples of success - counting random
variable", alpha = 0.75)

    plt.xlabel('# of successes in 50 fair Bernoulli trials')
    plt.ylabel('# of samples')
    plt.grid(True)
    #plt.savefig("Success_in_5_Bernoulli.png")
    plt.show()

if __name__ == "__main__":
    main()
```

Result:

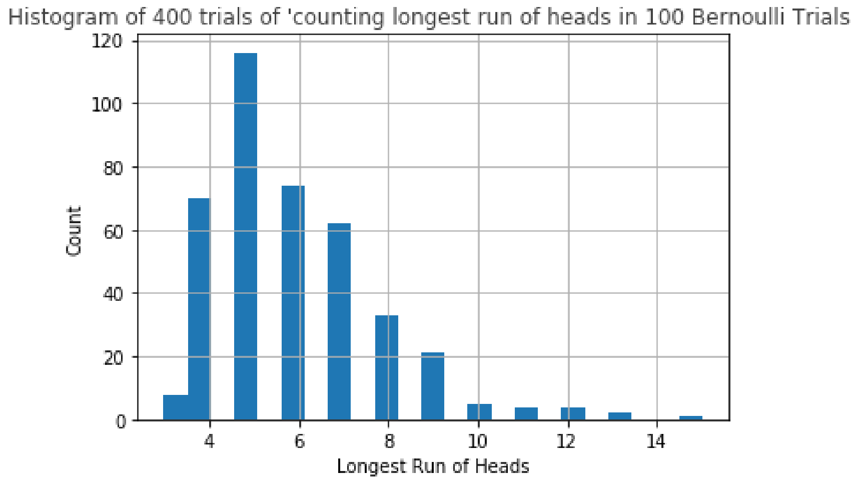

Figure 1.3 Histogram of 400 experiments of - Counting longest run of head in 100 Bernoulli trials

The above histogram can be approximated to be a Poisson distribution curve. The Poisson distribution expresses the probability of a given *number of events occurring* or *number of arrivals* in a fixed interval of time or space. Thus, counting of longest run of heads can be interpreted as *arrival* of continuous of heads in an interval of 100 Bernoulli trials.

2. Problem 2

[Counting successes]

Take your Bernoulli success-counting random variable (the binomial random variable). Generate and sum k=5 samples from this routine. Generate 300 such sums and histogram your results. Repeat for k = {10, 30, 50}. Comment on the histograms you observe for the different values of k.

Approach: This problem is similar to Problem 1 part (b). Instead of generating seven Bernoulli trials, we are generating k = 5, 10, 30, 50 Bernoulli trials. The problem asks to sum these k trials. Since these are Bernoulli trials, sum of these trials is nothing but counting the number of successes in k trials. Thus, this success-counting random variable can be thought of as i.i.d sum of k Bernoulli trials (as stated previously in Problem 1 part (b)). And this experiment needs to be repeated 300 times to plot the histogram.

Once again, we will use the inverse transform method to generate a Bernoulli trial.

Code:

```
import numpy as np
import random as rand
import matplotlib.pyplot as plt

def bernoulliTrials(numTrials):
    bernoulli_arr = np.empty([numTrials])

    for i in range(0,numTrials):
    # Return random floats in the half-open interval [0.0, 1.0).
    # Results are from the “continuous uniform” distribution over the stated
# interval
        x = rand.random()
        if(x >= 0.5):
            bernoulli_arr[i] = 1
        else:
            bernoulli_arr[i] = 0

    return bernoulli_arr

def countLongestRun(bernoulli_arr):

    longestRun = 1
    count = 1

    for i in range(0, np.size(bernoulli_arr) - 1):
        if(bernoulli_arr[i] == 1 and bernoulli_arr[i + 1] == 1):
            count = count + 1
        else:
            count = 1

        if(count >= longestRun):
            longestRun = count
        else:
            longestRun = longestRun

    return longestRun

def main():

    longestRun = np.empty([400])

    for i in range(0,400):
        bernoulli_arr = bernoulliTrials(100)
        longestRun[i] = countLongestRun(bernoulli_arr)

    plt.hist(longestRun, bins = 'auto')
    plt.title("Histogram of 400 trials of 'counting longest run of heads in
100 Bernoulli Trials ", alpha = 0.75)

    plt.xlabel('Longest Run of Heads')
    plt.ylabel('Count')
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    main()
```

Results:

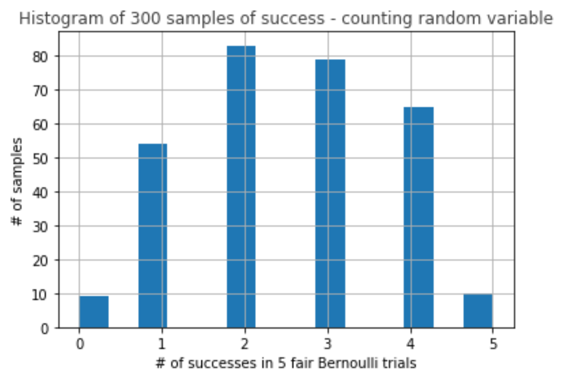

Figure 2.1 Histogram for sum of k = 5 trials

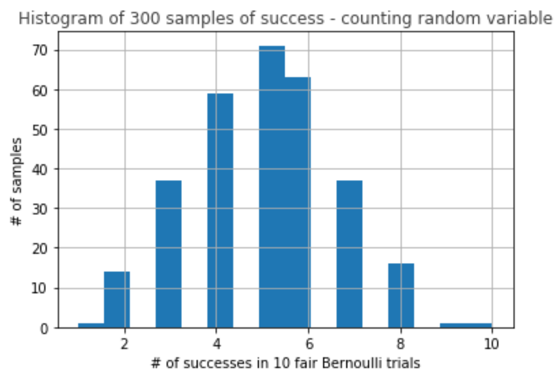

Figure 2.2 Histogram for sum of k = 10 trials

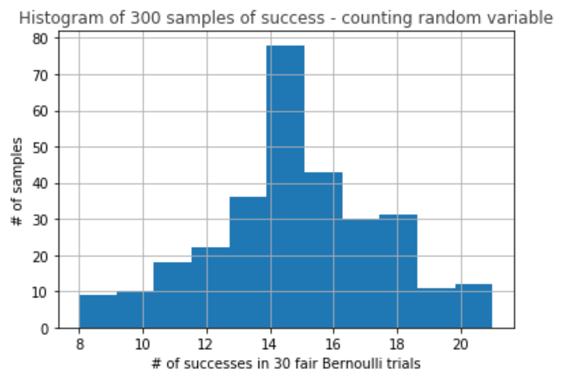

Figure 2.1 Histogram for sum of k = 30 trials

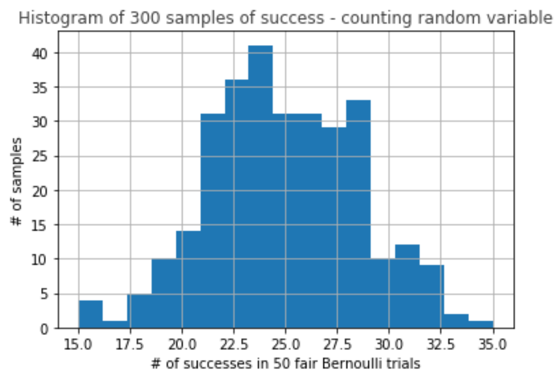

Figure 2.1 Histogram for sum of k = 50 trials

The above random variable of *sum of k Bernoulli trials* can be represented as

$$S_n = X_1 + X_2 + X_3 + \cdots + X_k$$

where $X_i = 1 \; or \; 0$ according as the ith outcome is a success or failure and k = {5, 10, 30, 50}. Once again, according to Central Limit Theorem, the distribution curve of S_n (S_n being an i.i.d sum) approaches to a Gaussian random variable with mean given by kp and variance given by is kpq. Thus, the above histograms look almost like Gaussian curves.

The histogram in figure 2.1 can be approximated by a Gaussian curve with mean kp = 5x0.5 = 2.5 and variance kpq = 1.25. The histogram in figure 2.2 can be approximated by a Gaussian curve with mean kp = 10x0.5 = 5 and variance kpq = 2.5. The histogram in figure 2.3 can be approximated by a Gaussian curve with mean kp = 30x0.5 = 15 and variance kpq = 7.5. The histogram in figure 2.4 can be approximated by a Gaussian curve with mean kp = 50x0.5 = 25 and variance kpq = 12.5.

3. Problem 3

[Networking: part 1]

Given n=20 people in a social network. Imagine that any given unordered pair of two people are connected at random and independently with success probability p = 0.05.
• How many possible edges or connections, N, exist in a group of n = 20 people?
• Write a routine to select edges with probability p=0.05 out of the N candidate edges. (think of the presence or absence of each distinct candidate edge as a Bernoulli trial)
• What is the distribution of the random number of edges selected in this way? Generate histograms to support your answer.

Approach: The above problem can be thought of as performing *N* Bernoulli trials and counting the number of successes.
Firstly, in a network of n = 20 people, the total number of possible edges is equal to

$$\mathbf{N} = \binom{20}{2} = 20\ x\ 19\ x\ 0.5 = 190$$

Again, using the inverse transform method we must generate 190 Bernoulli trials and count the number of successes in all the trials. And this experiment can be repeated for 500 times

Code:
```
import numpy as np
import random as rand
import matplotlib.pyplot as plt

def bernoulliTrialsForEdges(numPeople):

    edgeExist = []  # Empty list
    count = 0

    for i in range(0, numPeople - 1):
        for j in range(i + 1, numPeople):
        # Return random floats in the half-open interval [0.0, 1.0).
        # Results are from the "continuous uniform" distribution over the
stated interval
            x = rand.random()
            if(x <= 0.05):
                result = 1
                edgeExist.append([i, j, result])
                count = count + 1
            else:
                result = 0
                edgeExist.append([i, j, result])

    return count

def main():

    edgeExistTrial = np.empty([500])

    for i in range(0,500):
      edgeExistTrial[i] = bernoulliTrialsForEdges(20)

    plt.hist(edgeExistTrial, bins = 'auto')
    plt.title("Histogram of 500 simulated trials for counting # of edges that
exist in the social network", alpha = 0.75)

    plt.xlabel('# of edges in the social network')
    plt.ylabel('Count')
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    main()
```

Result:

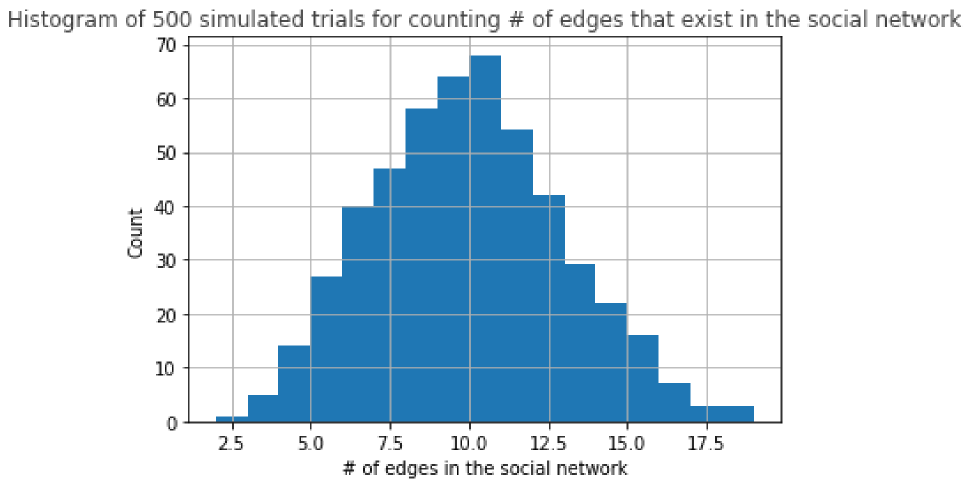

Figure 3 Histogram of 500 simulated Bernoulli trials for counting # of edges in n = 20 network

The above random variable of *counting number of edges* can be represented as

$$S_n = X_1 + X_2 + X_3 + \cdots + X_{190}$$

where $X_i = 1$ *or* 0 according as the ith outcome is a success(with probability p =0.05) or failure (with probability $q = 1\text{-}p$ =0.95).

Once again, according to Central Limit Theorem, the distribution curve of S_n (S_n being an i.i.d sum) approaches to a Gaussian random variable with mean given by np = 190x0.05 = 9.5 and variance given by is $npq = 9.025$. Thus, the above histogram looks almost like Gaussian curve.