Group Simulation Activity 1

Group 48

09/28/23

Luke Major: 20225654

Julia Dyck:

Sabrina Button:

Abigail Magno: 20267004

# Introduction

The primary objective of this activity was to build a comprehensive understanding of discrete random variables and their associated distributions. Using Python and the NumPy, Pandas, and Matplotlib libraries theoretical concepts, such as PMFs, CDFs, and various statistical distributions, were explored in a practical application. Through this lab, our team was able to further our ability to analyze and interpret data, recognize underlying patterns, and draw meaningful conclusions from observed outcomes.

# Results

## Question 1

### I

Method

* + x: Since each side of a fair dice has an equal probability of 1/6, the PMF was directly assigned as 1/6 for each value in x.
  + y: The code iterated through all possible combinations of dice rolls (from 1 to 6 for each dice). For each combination, it checked if the sum of the rolls matched a value in y. If it did, a counter for that value was incremented. The final count for each value was then divided by the total number of combinations (6^3) to get the probability.

Results:

* + x: The PMF was uniformly distributed, as expected. Each value from 1 to 6 had an equal probability.
  + y: The PMF displayed a bell-shaped distribution. The most probable sums were those around the middle values (like 10), which makes sense as there are more combinations of dice rolls that can result in these sums.

A graph of blue rectangular bars

Description automatically generated with medium confidenceA graph of a number of blue bars

Description automatically generated with medium confidence

### II

Method

* + For x: The CDF was computed by adding the probability 1/6 for each successive value in x.
  + For y: The code iterated through all possible combinations of dice rolls. For each value in y, it counted how many combinations resulted in a sum less than or equal to that value. This count was then divided by the total number of combinations (6^3) to get the cumulative probability.

Results:

* + y: The CDF had a sigmoid-like shape, starting slow, increasing rapidly in the middle, and then slowing down again. This shape reflects the cumulative probabilities of the sums, with middle sums being more probable.
  + x: The CDF showed a linear increase, reflecting the uniform distribution of a single dice roll. Each step increased by 1/6.

A graph of a number of blue bars

Description automatically generated with medium confidenceA graph of a bar graph

Description automatically generated with medium confidence

### IV

Method

* + A counter function was defined to count the occurrences of each number in each vector.
  + This function was applied to x1, x2, and x3 to get the frequencies H1, H2, and H3, respectively.
  + The frequencies were then plotted and compared with the PMF of x.

Results

* + When visualized, these frequencies showed some variation due to the randomness of the dice rolls and the limited sample size of 100 trials. However, the general trend was consistent with the theoretical PMF of x

A graph of different colored bars

Description automatically generated

### V

Method

* + The same counter function used in Part IV was applied to the y vector to get the frequency of each possible outcome.
  + The frequencies were then plotted and compared with the PMF of y.

Result

* + When plotted, the distribution of these frequencies resembled the bell-shaped PMF of y but with some variations due to the randomness and limited sample size.

A graph of blue bars

Description automatically generated

### Vi

Result

* + When the number of trials was increased to 1 million, the observed frequencies in H1, H2, H3, and H closely matched the theoretical PMFs for both x and y.
  + This convergence demonstrated the law of large numbers in action. The larger the sample size, the closer the observed frequencies came to the expected probabilities.

## Question 2

### I

Method

* + The mean function works by summing up all the values in the input array and then dividing by the number of elements in the array.
  + The variance function calculates the average of the squared differences from the mean

Results

* + Mean of RV2: 30.003087
  + Mean of RV3: 30.007068
  + Variance of RV1: 100.168461071856
  + Variance of RV2: 25.129345470430998
  + Variance of RV3: 25.14194204337601

### II

Method

* + The counter function counts the frequency of each discrete value in the input data.
  + It initializes an array of zeros, H, with a length equal to the range of possible values (from min to max).
  + For each data point in the input, the function increments the corresponding index in the H array.
  + The index is determined by subtracting the min value and an additional 1 from the data point.
  + This results in a histogram-like array where each index represents a value and the content at that index represents the frequency of that value.

Results

* + All three sets appear to follow a normal distribution.

A blue and white graph

Description automatically generated

A graph of a number of values

Description automatically generated

A graph of a number of values

Description automatically generated with medium confidence

### III

Method

* + The prob function estimates the probability of the input data taking values between two specified bounds (min and max).
  + It works by summing the frequencies of the values in the specified range and then dividing by the total number of data points.

Results

* + Probability of RV1 between 10 and 40: 0.170853
  + Probability of RV2 between 10 and 40: 0.982133
  + Probability of RV3 between 10 and 40: 0.981952

# Code

## Question 1

'''

ELEC 326 - Group 48

Assignment 1 -> Question 1 🏎️🏎️🏋️‍♂️🦀🦀

Sabrina Button, Luke Major, Julia Dyck, Abby Magno

'''

import numpy as np

import scipy as sp

import matplotlib.pyplot as plt

# Declare the range of RVs

x = np.arange(1,7)

y = np.arange(3,19) # y is the RV sum of three dice rolls

# Print RVs as a test

print("RV x: ", x)

print("RV y: ", y)

# Function for quickly making plots

def plot(name, x, y, y\_label):

    plt.figure()

    plt.bar(x, y, align='center')

    plt.title(name)

    plt.ylabel(y\_label)

    plt.show()

# PMF of y

def PMF\_of\_y(y, num\_rolls, num\_sides=6):

    # Number of elements in PMF is cardinality of y

    pmf = np.zeros(y.size)

    denominator = num\_sides\*\*num\_rolls

    print(denominator)

    for i in range(y.size):

        # Figure out how many ways we can get y[i] with num\_rolls dice

        # This is the numerator of the PMF

        numerator = 0

        for j in range(1, num\_sides+1):

            for k in range(1, num\_sides+1):

                for l in range(1, num\_sides+1):

                    if j+k+l == y[i]:

                        numerator += 1

        pmf[i] = numerator/denominator

    return pmf

# PMF for y

def PMF\_of\_x(x, num\_sides=6):

    # Number of elements in PMF is cardinality of x

    pmf = np.zeros(x.size)

    for i in range(x.size):

        pmf[i] = 1/num\_sides

    return pmf

# PMF for y

def CDF\_of\_y(y, num\_rolls, num\_sides=6):

    # Number of elements in CDF is cardinality of y

    cdf = np.zeros(y.size)

    denominator = num\_sides\*\*num\_rolls

    # iterate through possible sums and find all methods of getting that sum

    for i in range(y.size):

        # Figure out how many ways we can get y[i] with num\_rolls dice

        # This is the numerator of the CDF

        numerator = 0

        for j in range(1, num\_sides+1):

            for k in range(1, num\_sides+1):

                for l in range(1, num\_sides+1):

                    if j+k+l <= y[i]:

                        # If this adds to our desired value, that means there's one more

                        # way to get that value, so increment numerator

                        numerator += 1

        cdf[i] = numerator/denominator

    return cdf

def CDF\_of\_x(x):

    # Number of elements in CDF is cardinality of x

    cdf = np.zeros(x.size)

    for i in range(x.size):

        cdf[i] = (i+1)/6

    return cdf

def do\_trials(num):

    # Try 100 dice rolls for each

    rng = np.random.default\_rng()

    x1 = rng.integers(low=1, high=7, size=(num))

    x2 = rng.integers(low=1, high=7, size=(num))

    x3 = rng.integers(low=1, high=7, size=(num))

    # Store sum of three rolls in y\_out

    y\_out = np.sum([x1, x2, x3], axis=0)

    # print all variables

    print("x1: ", x1)

    print("x2: ", x2)

    print("x3: ", x3)

    print("y: ", y\_out)

    # Count frequency of each value of RV

    def counter(trial, outcomes=6, min=1, max=6):

        H = np.zeros(outcomes)

        for i in range(trial.size):

            H[trial[i] - (min) ] += 1

        return H

    # Count frequency of each value of RV

    H1 = counter(x1)

    H2 = counter(x2)

    H3 = counter(x3)

    # plot H1, H2, H3 on one bar chart

    plt.figure()

    plt.bar(np.arange(1,7), H1, align='center')

    plt.bar(np.arange(1,7), H2, align='center')

    plt.bar(np.arange(1,7), H3, align='center')

    plt.title("H1, H2, H3")

    plt.xlabel("x")

    plt.ylabel("Frequency")

    plt.show()

    # Try dividing H1 by N=100

    H1 = np.divide(H1, num)

    # plot H1/N as a bar chrt

    plot("H1/N", x, H1, "Probability")

    H = counter(y\_out, outcomes=16, min=3, max=18)

    # plot H as bar chart

    plot("H", y, H, "Frequency")

# \*\*\*\*\* Get values and figures for question 1 using above funtions: \*\*\*\*\*

# I) plot pmf of y as bar chart

plot("PMF of RV y", y, PMF\_of\_y(y, 3), "Probability")

# I) plot pmf of x as bar chart

plot("PMF of RV x", x, PMF\_of\_x(x), "Probability")

# II) plot cdf of y as bar chart

plot("CDF of RV y", y, CDF\_of\_y(y, 3), "Probability")

# II) plot cdf of x as bar chart

plot("CDF of RV x", x, CDF\_of\_x(x), "Probability")

# III-V) Do 100 trials

do\_trials(100)

# V) Do 1000000 trials

do\_trials(1000000)

## Question 2

'''

ELEC 326 - Group 48

Assignment 1 -> Question 2 🔥🔥🔥🔥💻🧠🤯

Sabrina Button, Luke Major, Julia Dyck, Abby Magno

'''

import scipy.io

import numpy as np

import matplotlib.pyplot as plt

# Load MATLAB files

RV1 = scipy.io.loadmat('C:\\Users\\lukep\\OneDrive\\Serious-Stuff\\3rd-year\\ELEC-326-Probability-and-Random-Processes\\probability\_assignments\\assignment1\\\RV1.mat')

RV2 = scipy.io.loadmat('C:\\Users\\lukep\\OneDrive\\Serious-Stuff\\3rd-year\\ELEC-326-Probability-and-Random-Processes\\probability\_assignments\\assignment1\\\RV2.mat')

RV3 = scipy.io.loadmat('C:\\Users\\lukep\\OneDrive\\Serious-Stuff\\3rd-year\\ELEC-326-Probability-and-Random-Processes\\probability\_assignments\\assignment1\\\RV3.mat')

# Get the RVs from MATLAB files

RV1 = RV1['RV1']

RV2 = RV2['RV2']

RV3 = RV3['RV3']

# Calculate the mean of RVs

def mean(input):

    return np.sum(input)/len(input)

# Calculate the variance of RVs

def variance(input):

    return np.sum((input - mean(input))\*\*2)/len(input)

print("Mean of RV1: ", mean(RV1[0]))

print("Mean of RV2: ", mean(RV2[0]))

print("Mean of RV3: ", mean(RV3[0]))

print("Variance of RV1: ", variance(RV1[0]))

print("Variance of RV2: ", variance(RV2[0]))

print("Variance of RV3: ", variance(RV3[0]))

# Count frequency of each value of RV

def counter(input, min=0, max=None):

    if max is None:

        max = np.max(input)

    H = np.zeros(max - min + 1)

    for i in input:

        H[i - min] += 1

    return H

# Plot bar chart of RV

def plot(name, H):

    plt.figure()

    plt.bar(np.arange(len(H)), H)

    plt.title(name)

    plt.xlabel("Value")

    plt.ylabel("Frequency")

    plt.show()

# Estimate probability of RV landing between 10 and 40

def prob(H, min, max):

    return np.sum(H[min:max+1])/np.sum(H)

# \*\*\*\*\* Get values and figures for question 2 using above funtions: \*\*\*\*\*

# Count frequency of each value of RV

H1 = counter(RV1[0])

H2 = counter(RV2[0])

H3 = counter(RV3[0])

plot("H1", H1)

print("Probability of RV1 between 10 and 40: ", prob(H1, 10, 40))

plot("H2", H2)

print("Probability of RV2 between 10 and 40: ", prob(H2, 10, 40))

plot("H3", H3)

print("Probability of RV3 between 10 and 40: ", prob(H3, 10, 40))