Lab 3 Report: Probability, Distributions, and the Empirical Rule

Henry Wandover

February 17, 2023

Collaboration Statement:

- I worked on this lab by myself.
- I used scipy docs to reference what quad did.

1 Part I - Regress to Mean

1.1

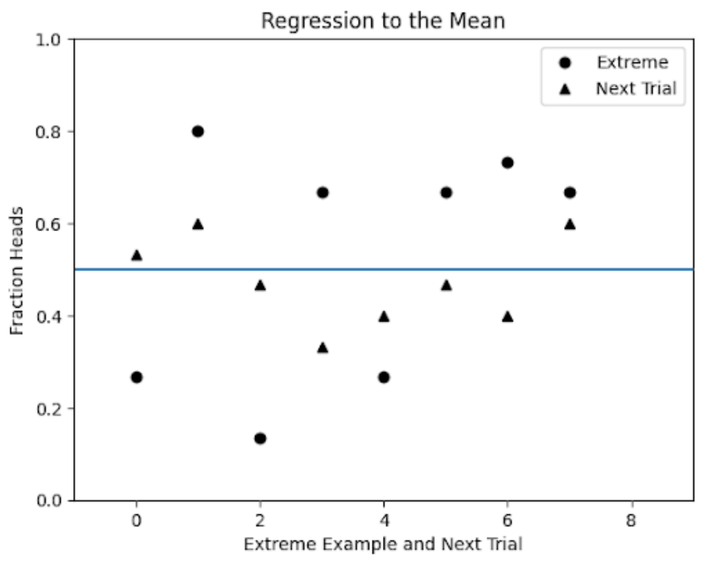

Figure 1: Distribution of ages in population sample.

The blue line represents the mean of the given random data, looking at the graphed extremes and the following trial, each time the next trial appears closer to the mean. Except for the point at 3 on the x-axis which is reflected across the mean and is still the same distance from the mean.

1.2

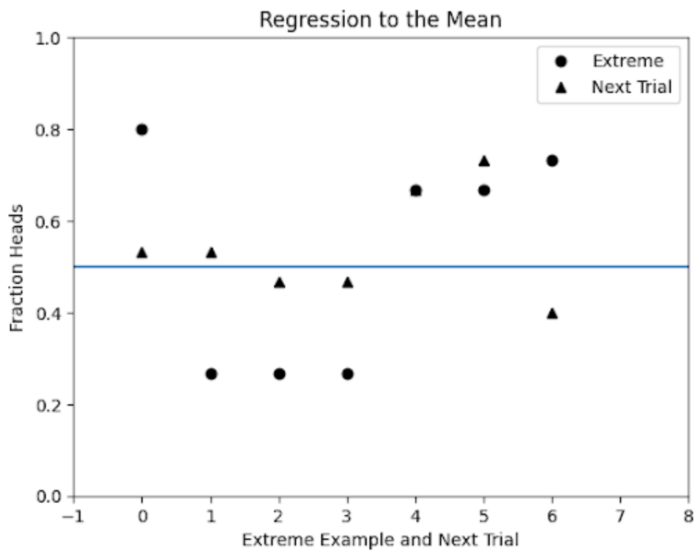

Figure 2: Distribution of ages in population sample.

Here the point at 4 has an extreme and the next trial that is the same fraction of heads, and at 5 the extreme is closer to the mean. I believe the data is showing this way due to the gambler's fallacy, one would assume that after a certain amount of heads, a tail would land and then in the next trail would be closer to the mean. In a coin flip like this, there is no guaranteed outcome that can be mapped so to assume a regression to the mean is false.

1.3

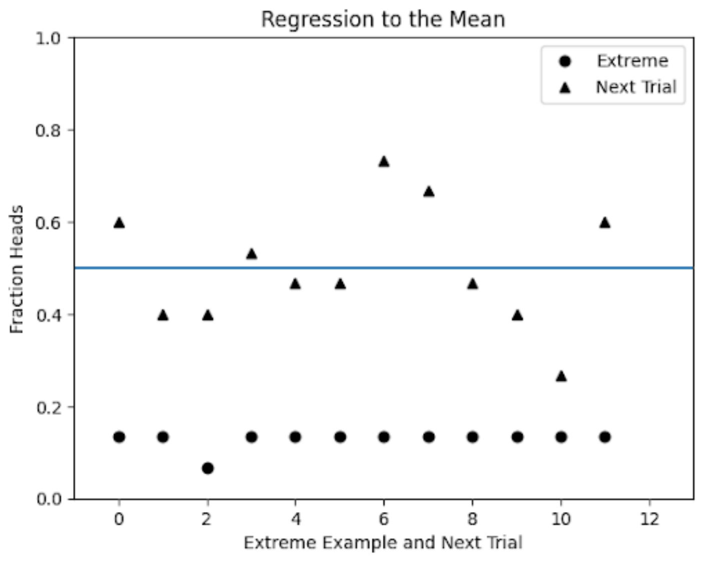

Figure 3: Distribution of ages in population sample.

In this case I only accounted for extremes below 0.15, I had to run significantly more trials as it was unlikely that one would show with merely 40 trials. But when running more trials and after a couple of cycles it appears that with a lower extreme the frequency of regression to the mean was higher than previously.

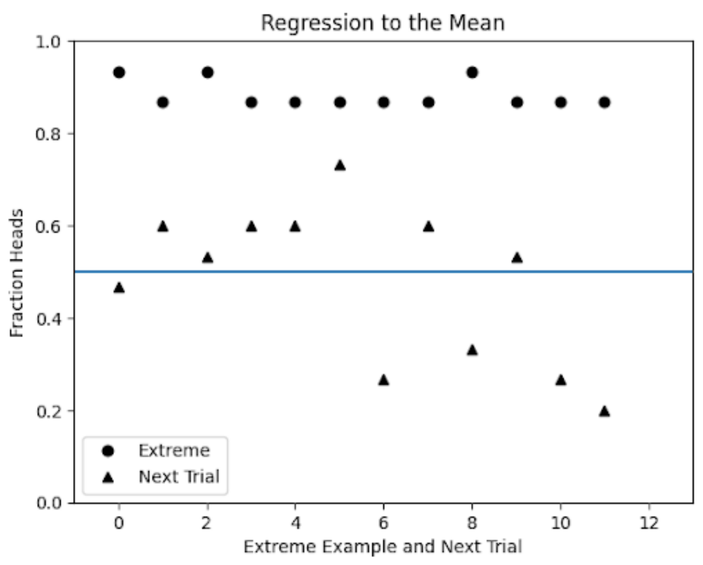

Figure 4: Distribution of ages in population sample.

After looking at both, I can't say that there was a significant enough difference in frequencies between an increase or decrease in the extreme value. Looking at the spread of patterns above, where the extreme accounted for was ¿ 0.8, does not look to dissimilar from the former. The lower extremes has a slightly tighter patter but increasing or decreasing show cases regression better than the normal model.

2 Part II - Empirical Rule

2.1

```
For mu = -3 and sigma = 3
  Fraction within 1 std = 0.6827
  Fraction within 1.96 std = 0.95
  Fraction within 2 std = 0.9545
  Fraction within 2.576 std = 0.99
For mu = -7 and sigma = 3
  Fraction within 1 std = 0.6827
  Fraction within 1.96 std = 0.95
  Fraction within 2 std = 0.9545
  Fraction within 2.576 std = 0.99
```

2.2

The two different trials don't differ at all, since both examples are going to be normally distributed the mean = median = mode, and each time no matter if the mean and standard deviation differ the standard deviations are going to be the same. There is going to be complete symmetry around the center and 50

2.3

```
    scipy.integrate.quad(gaussian,-3,3,(0,1))[0])
```

The code *scipy.integrate.quad* computes a definite integral based on a few parameters. In this case we are passing the Gaussian function which in other terms is referred to as the probability density function, essentially it will return the frequency of points within a given range of standard deviations. The -3 and 3 are the lower and upper bounds of the quad function and work in the context of Gaussian as -3 standard deviations and 3 standard deviations as the bounds of out prompt. Passing (0,1) as arguments in quad are interpreted by Gaussian as the mean and standard deviation respectively. The answer should be 99.7 and after running said code the result is 0.9973002039367399. That checks out based on the 68-95-99 rule which states for within 3 standard deviations of the mean there should be 99.7 of the population. And looking at out inputs with the range of -3 and 3, as well as the mean being 0 with a standard deviation of 1, it's a stock standard Gaussian distribution so said rule applies.

3 Part III - Simulation of Rolling a D3

```
def roll():
   """Assumes numRolls a positive int"""
   return random.choice((1, 2, 3))


def rollSim(numTrials, numRollsPerTrial):
   ones, twos, threes = 0, 0, 0
   for i in range(numTrials):
       for i in range(numRollsPerTrial):
           r = roll()
           if r == 1:
               ones += 1
           elif r == 2:
               twos += 1
           else:
               threes += 1
   return (ones, twos, threes)
```

3.1

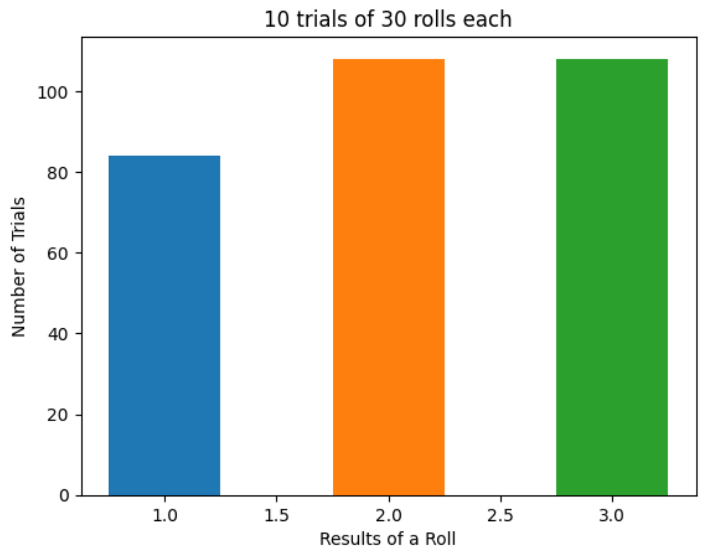

Figure 5: Frequency of 1s, 2s, and 3s after 10 trials with 30 rolls each.

I would say with the trial of 30 rolls it's closest to a bimodal. Often it seems there is two more likely chances while one side is significantly less likely to appear. While not exactly bimodal it's the closest description because of the appearance of two modes at both 1 and 3.

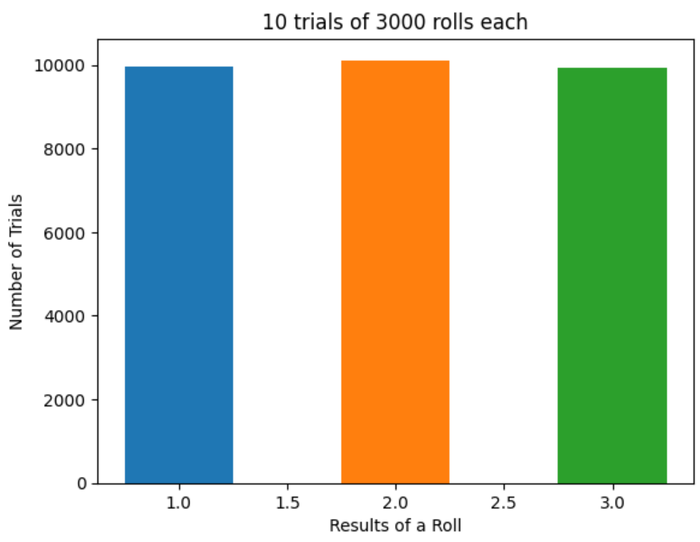

Figure 6: Frequency of 1s, 2s, and 3s after 10 trials with 3000 rolls each.

With the increase in trials to 3000 it's clear to see things even out and become much more of an uniform distribution as the number of trials has increased and it's safe to say the more trials the more uniform it will become.

3.2

```
def rollMF(numRolls):
    """Assumes numRolls a positive int"""
    ones = 0
    for i in range(numRolls):
        if random.choice((1,2,3)) == 1:
            ones += 1
    return ones/float(numRolls)

def rollSimMF(numTrials, numRollsPerTrial):
    fracOnes = []
    for i in range(numTrials):
        fracOnes.append(rollMF(numRollsPerTrial))
    mean = sum(fracOnes) / len(fracOnes)
    sd = stdDev(fracOnes)
    return (fracOnes, mean, sd)

def showErrorBars(minExp, maxExp, numTrials):
    """Assumes minExp and maxExp positive ints; minExp < maxExp
         numTrials a positive integer
       Plots mean fraction of heads with error bars"""
    means, sds, xVals = [], [], []
    for exp in range(minExp, maxExp + 1):
        xVals.append(2**exp)
        fracOnes, mean, sd = rollSimMF(2**exp, numTrials)
        means.append(mean)
        sds.append(sd)
    pylab.errorbar(xVals, means, yerr=1.96*pylab.array(sds))
    pylab.semilogx()
    pylab.title('Mean Fraction of Ones ('
                + str(numTrials) + ' trials)')
    pylab.xlabel('Number of rolls per trial')
    pylab.ylabel('Fraction of Ones & 95% confidence')
```

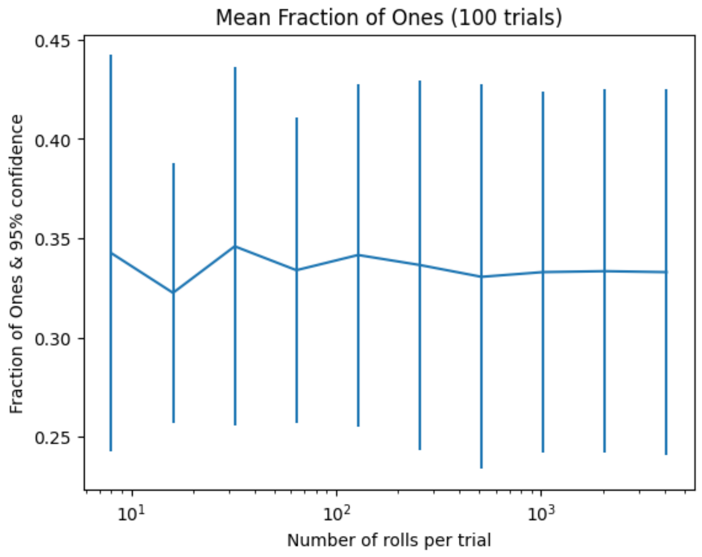

Figure 7: Mean Fraction of Ones rolled with a three-sided die after 100 trials.

This is my base case with 100 trials.

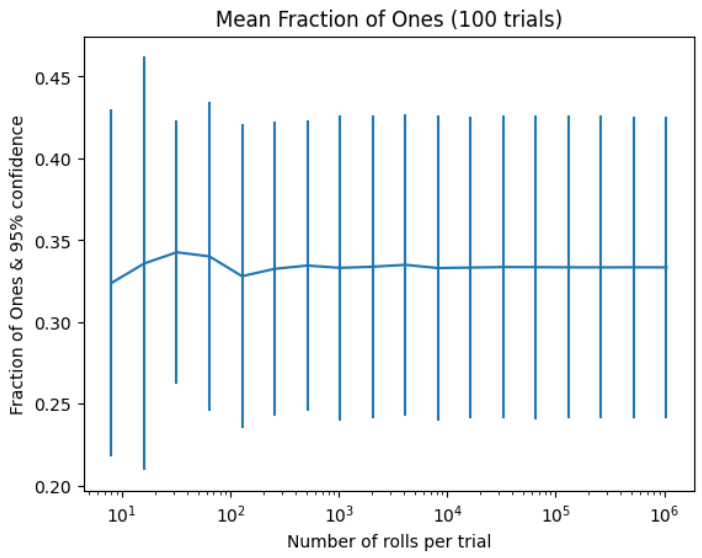

Figure 8: Mean Fraction of Ones rolled with a three-sided die after 100 trials, doubled the amount of rolls from last figure.

After doubling the amount of rolls per trial it's clear to see that there are more points on the graph. However, it looks to be that aside from one outlier at the beginning the error bars remain the same size with an increased amount of rolls.

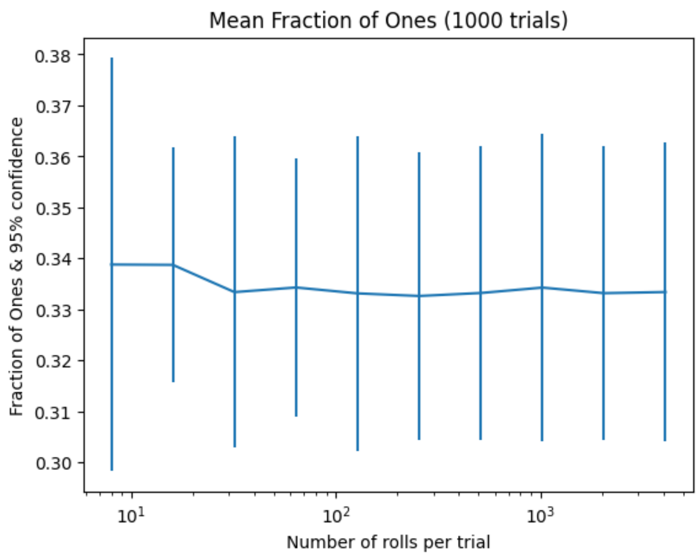

Figure 9: Mean Fraction of Ones rolled with a three-sided die after 1000 trials.

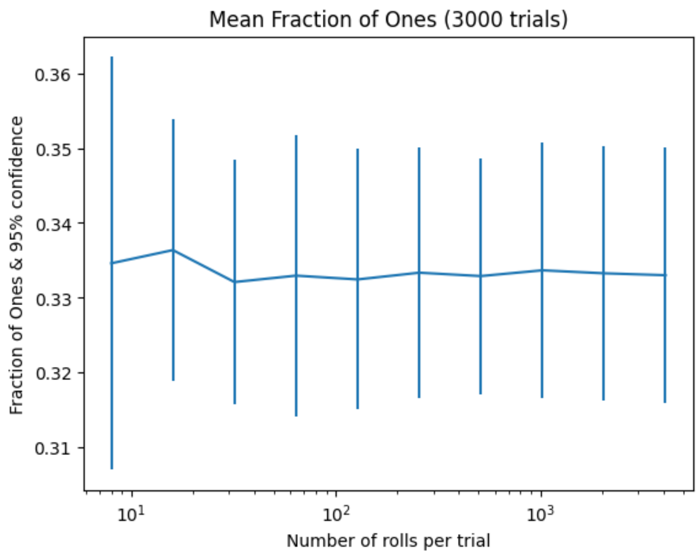

Figure 10: Mean Fraction of Ones rolled with a three-sided die after 3000 trials.

Looking at figure 9 and 10, however, show a stark change in the size of error bars. Going all the way down from between .45 and .25 to between .35 and .32 in the 3000 trials example. While the sizes aren't uniform as in figure 8, the size themselves are definitely smaller than the former two examples with only 100 trials. With addition rolls and an increased trial count they would be smaller and more consistent compared to the base case graph.

This is the clearly an example of the law of large numbers. As we increased the number of trials the average of the trial results becomes more clear. So in other words the margin of error, as showcased by the graphs, decreases significantly as more trials are performed.

3.3

```
def rollTwoD3(numRolls):
"""Assumes numRolls a positive int"""
rolledResults = [0, 0, 0, 0, 0,]
```

```
for i in range(numRolls):
    val1 = random.choice((1, 2, 3))
    val2 = random.choice((1, 2, 3))
    result = val1 + val2
    if (result == 2):
        rolledResults[0] += 1
    elif (result == 3):
        rolledResults[1] += 1
    elif (result == 4):
        rolledResults[2] += 1
    elif (result == 5):
        rolledResults[3] += 1
    elif (result == 6):
        rolledResults[4] += 1
return rolledResults
```

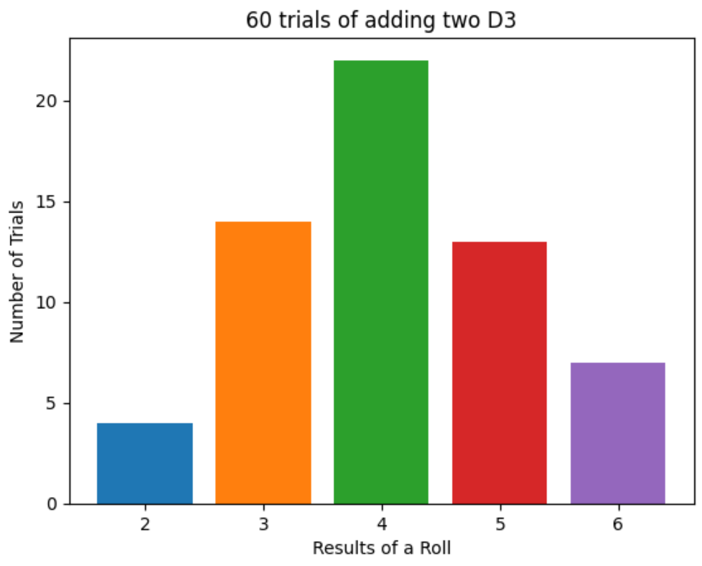

Figure 11: 60 Trials of Rolling Two D3 and adding the result.

Compared to the single die roll results, these are significantly closer to a normal distribution. As seen by the obvious mode, median, and mean with an spread around it. the single die roll showcased both bimodal and with an increase in trials a uniform distribution. At this point it seems to be approaching a normal distribution, while not exactly there yet as with other trials it sometimes came back not as clean.

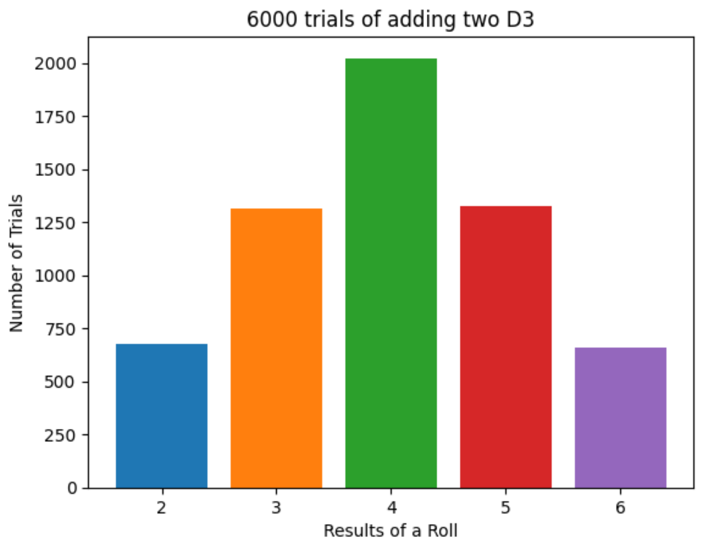

Figure 12: 6,000 Trials of Rolling Two D3 and adding the result.

With an increase to 6,000 trials we can see a very even normal distribution as compared to the 60 trials. With the result of 4 being the clear mean of the data and 2, 3, 5, and 6 evenly spaced out to the edges of the graph. The prior figure 11 resembled a normal distribution but in accordance with the law of large numbers we can now clearly see a normal distribution with an increase in trials.

3.4

Looking back at my results in Lab 1, I found that what I calculated by hand for rolling one D3 was a uniform distribution, and for rolling two D3 and adding the results I plotted a normal distribution. There the data wasn't random and so every run of new trials would be exactly the same each time, while here in Lab 3 it differed in that regard. According again to the law of large numbers it makes sense that after a higher count of trials that things would come to resemble the expected. With an increase in trials for the single die, it went from bimodal to the uniform distribution I had expected in Lab 1, same with the two D3 which went from a resemblance of a normal distribution to a confident and clean normal distribution with 6,000 trials.

So all in all they are as I predicted in Lab 1, with enough of trials they come to resemble the results of the previous work.

3.5 CHALLENGE

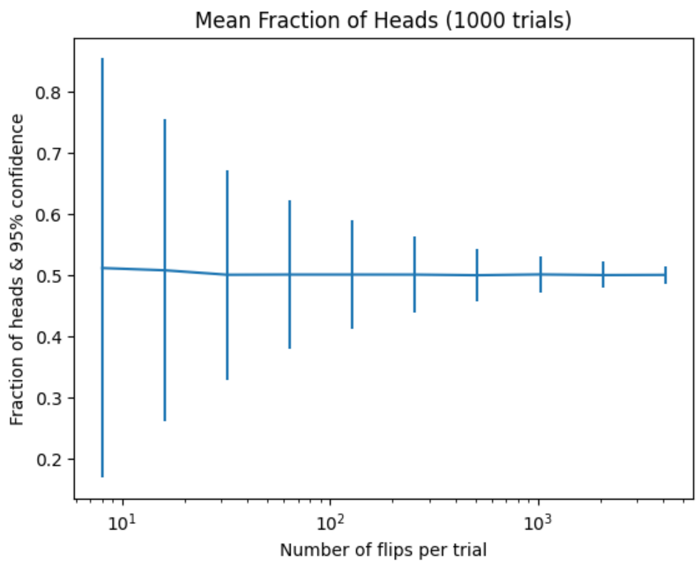

Figure 13: Error bars of Mean Fraction of Heads, 1,000 Trials.

This figure shows similar to what was already seen with the D3, shows the margin of error as the trials go on. Like how with the D3 it was marking the mean fraction of rolling a one, here it tracks the mean fraction of getting heads when flipping a coin. The charts obviously have a different characteristic than the D3. In this case there are only two outcomes, heads or tails, while with the D3 there is either a one, two, or three. Looking at both the 1,000 trials run for D3 rolls and coin flips, the graphs are very different looking. For the most part the D3 maintains the same length of error bars as more trials go on after the first extreme case. Because there is an extra outcome to account for with a D3, the coin has only two outcomes and as the trials go on it slowly approaches the mean with shrinking margin of error. With an increase from 100 to 1,000 trials for the D3 the extremes of error decrease a .2 range to .3 range, generally. While the coin flips error margin decrease significantly without needing to increase trial number. While different due to outcome amounts, they still both represent a decrease in error margin due to more trials in different ways.

3.6 CHALLENGE

```
def rollD6(numRolls):
    rolledResults = [0, 0, 0, 0, 0, 0]
    for i in range(numRolls):
        val = random.choice((1, 2, 3, 4, 5, 6))
        if (val == 1):
            rolledResults[0] += 1
        elif (val == 2):
            rolledResults[1] += 1
        elif (val == 3):
            rolledResults[2] += 1
        elif (val == 4):
            rolledResults[3] += 1
        elif (val == 5):
            rolledResults[4] += 1
        elif (val == 6):
            rolledResults[5] += 1
    return rolledResults
```

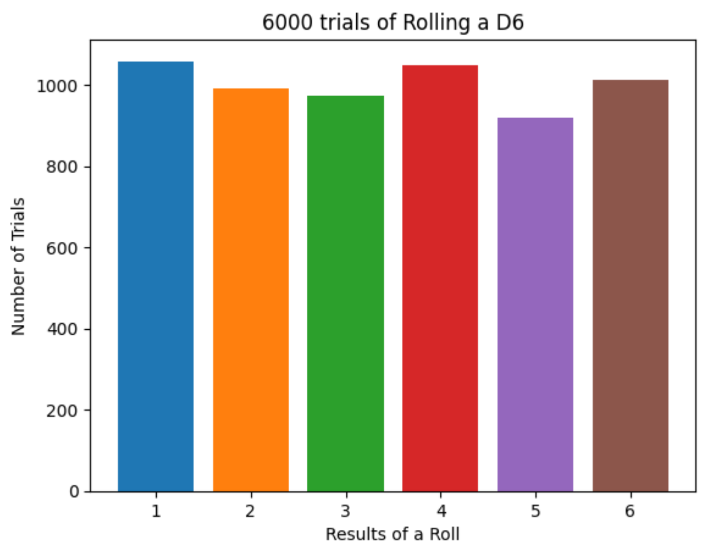

Figure 14: 6,000 Trials of Rolling a D6.

4 Part IV - Graphing Probability Distributions

4.1 Probability Density Function

4.1.1

Using

```
norm.pdf(0, 0, 1)
```

I get 0.3989422804014327.

4.1.2

Using

```
norm.pdf(0.5, 1, 3)
```

I get 0.13114657203397997.

4.1.3

```
x = np.linspace(-4, 4)
y = norm.pdf(x, 1, 3)
pylab.figure(0)
pylab.plot(x,y)
```

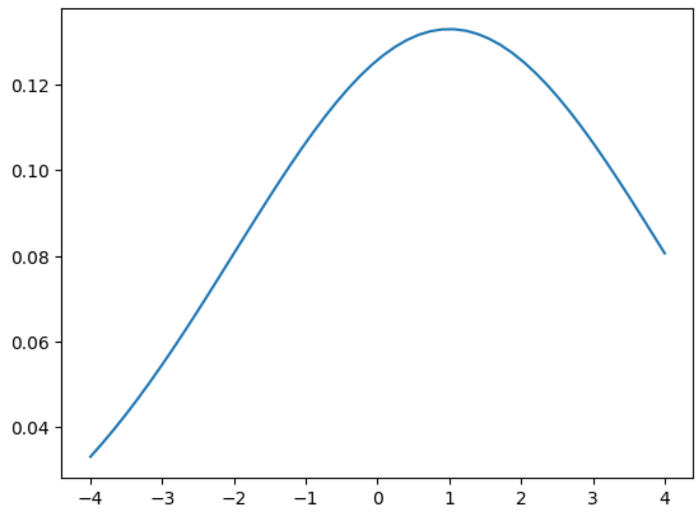

Figure 15: Probability Density Function of a normal distribution with mean 1 and standard deviation of 3, from -4 to 4.

4.2 Cumulative Distribution Function

4.2.1

Using

```
norm.cdf(0, 0, 1)
```

I get 0.5.

4.2.2

Using

```
norm.cdf(0.5, 1, 3)
```

I get 0.43381616738909634.

4.2.3

```
x = np.linspace(-4, 4)
y = norm.cdf(x, 1, 3)
pylab.figure(1)
pylab.plot(x,y)
```

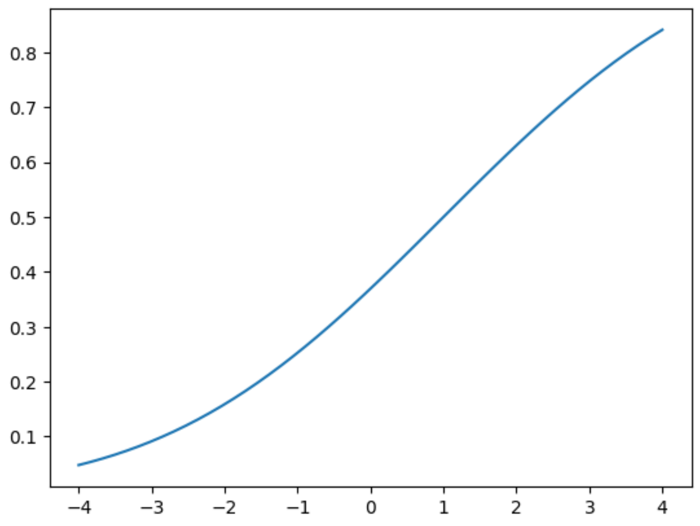

Figure 16: Cumulative Distribution Function of a normal distribution with mean 1 and standard deviation of 3, from -4 to 4.

4.3 Probability Point Function

4.3.1

Using

```
norm.ppf(0.5, 1, 3)
```

I get 1.0.

4.3.2

```
x = np.linspace(-4, 4)
y = norm.ppf(x, 1, 3)
pylab.figure(2)
pylab.plot(x,y)
```

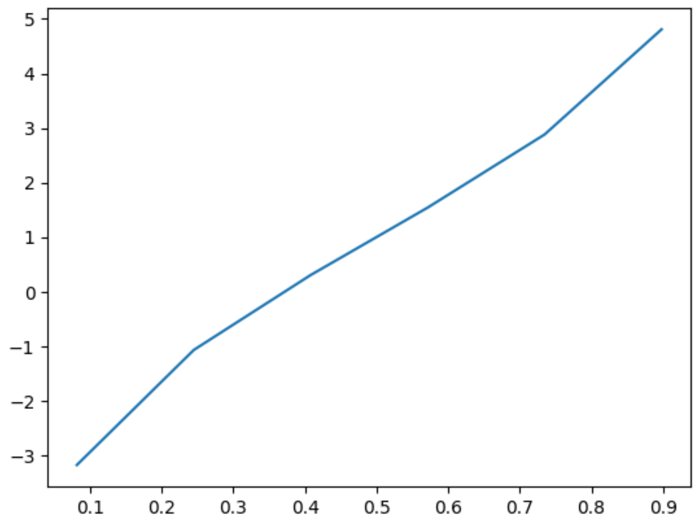

Figure 17: Probability Point Function of a normal distribution with mean 1 and standard deviation of 3, from -4 to 4.