Probability & Statistics Project

Hana Ali Rashid, hr05940
Tasmiya Malik, Student ID
Ifrah Ilyas, Student ID

April 27, 2021

Q1: Random Walk

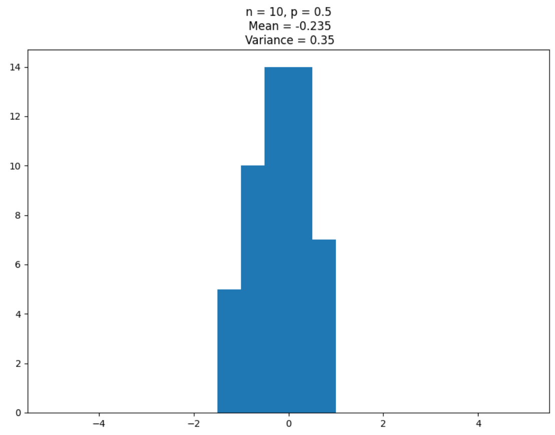
1.1

In order to simulate a random walk, the following function takes in values of n (number of steps to take), and p (the probability of the object moving one step to the right). It then generates a random number and uses the given probability to determine the direction that the object moves in, and returns the final position.

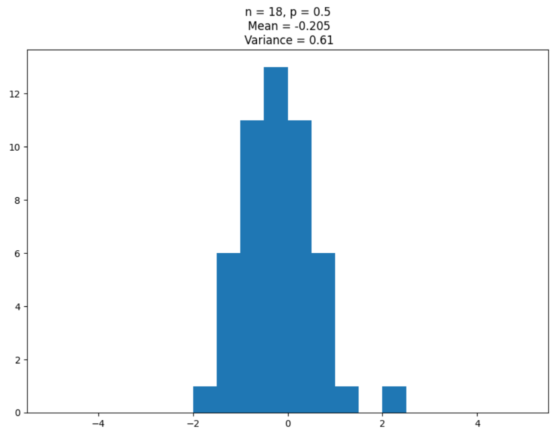
```python
def get_updated_position(n,p):
    pos = 0 #position
    for _ in range(n):
        rand = random.randint(1,100) #generating a random number in the range 1 to 100
        if rand < p*100:
            pos += 1 #move one step right
        else:
            pos -= 1 #move one step left
    return pos #return final position
```

The following function uses the `get_updated_position()` function to get 25 outcomes which are then used to calculate a single expected value for the final position. This is repeated 50 times and the histogram of all these expected values is then plotted.

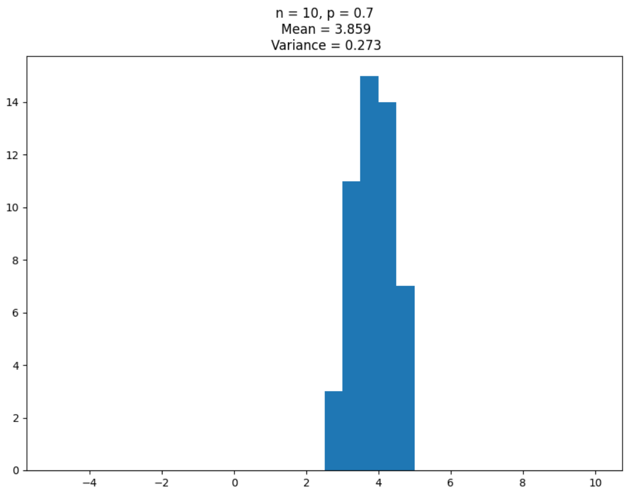
```python
def main_11():
    '''Calculating and plotting expected outcomes for various combinations of n and p'''
    p = 0.7
    n = 10
    expected = []
    for j in range(50): #expected values for each (n,p) for 50 iterations
        outcomes = []
        for i in range(25):
            outcomes.append(get_updated_position(n,p))
        expected.append(sum(outcomes)/25) #appending the expected (average) value for each(n,p
    )
    #plotting and showing a histogram of calculated expected values
    fig, ax = plt.subplots(figsize =(10, 7))
    ax.hist(expected, bins = range(-5,10))
    plt.title('n = '+str(n)+', p = '+str(p)+'\n Mean = '+str(round(statistics.mean(expected)
    ,3))+'\n Variance = '+str(round(statistics.variance(expected),3)))
    plt.savefig("Q1_histograms/q1"+'_n = '+str(n)+'_ p = '+str(p)+'.png')
    plt.show()
```



(a) No. of steps taken = 10 with probability of moving a step right = 0.5.



(b) No. of steps taken = 18 with probability of moving a step right = 0.5.



(c) No. of steps taken = 10 with probability of moving a step right = 0.7.

Histograms produced by the above code for expected final positions of objects against frequency, where n is the number of steps taken and p is the probability of the object moving one step to the right.

Histograms (a) and (b) have the same value of p and varying number of steps n.
Both histograms appear to follow a normal distribution, with (a) having a mean of -0.235 and a variance of 0.35 and (b) having a mean of -0.205 and a variance of 0.61. Increasing the number of steps has not had a significant effect on the mean but did increase the variance.

Histograms (a) and (c) have the same number of steps n and varying values of p.
Histogram (c) also appears to follow a normal distribution, having a mean of 3.859 and a variance of 0.273. Increasing p appears to have affected the mean of the distribution but not the variance as such.

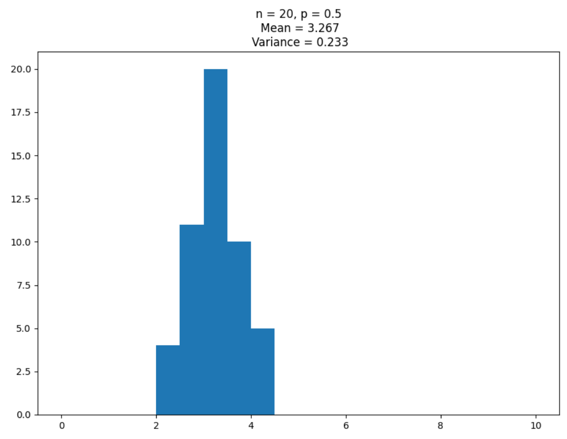
1.2

The following function is the modified version of the `get_updated_position()` function from the previous part such that the object cannot move into the negative part of the number line.

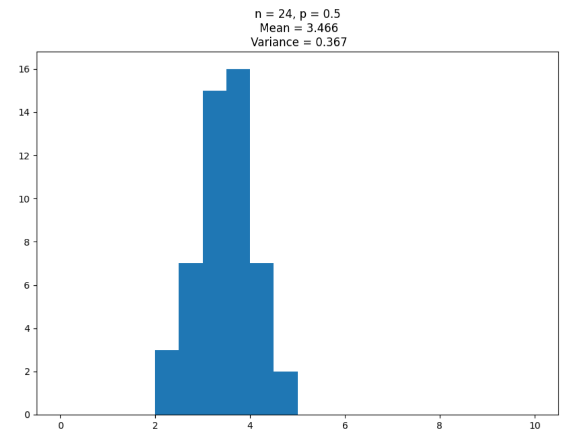
```python
def get_updated_position_restricted(n,p):
    pos = 0 #position
    for _ in range(n):
        rand = random.randint(1,100) #generating a random number in the range 1 to 100
        if rand < p*100 or pos <= 0: #move one step right if pos == 0
            pos += 1
        else:
            pos -= 1 #move one step left
    return pos #return final position
```

The following function uses the `get_updated_position_restricted()` function to get 25 outcomes which are then used to calculate a single expected value for the final position. This is repeated 50 times and the histogram of all these expected values is then plotted.

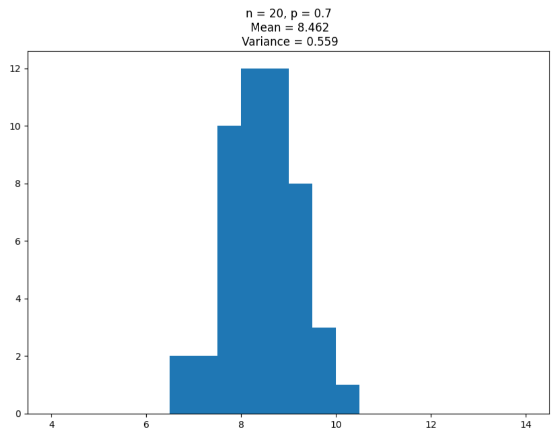
```python
def main_12():
    '''Calculating and plotting expected outcomes for various combinations of n and p'''
    p = 0.7
    n = 24
    expected = []
    for j in range(50): #expected value for each (n,p) for 25 iterations
        outcomes = []
        for i in range(25):
            outcomes.append(get_updated_position_restricted(n,p))
        expected.append(sum(outcomes)/25) #appending the expected (average) value for each(n,p
    )
    #plotting and showing a histogram of calculated expected values
    fig, ax = plt.subplots(figsize =(10, 7))
    ax.hist(expected, bins =
    [4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,10,10.5,11,11.5,12,12.5,13,13.5,14])
    plt.title('n = '+str(n)+', p = '+str(p)+'\n Mean = '+str(round(statistics.mean(expected)
    ,3))+'\n Variance = '+str(round(statistics.variance(expected),3)))
    plt.savefig("Q1_histograms/Q1.22 "+'_n = '+str(n)+'_p = '+str(p)+'.png')
    plt.show()
```



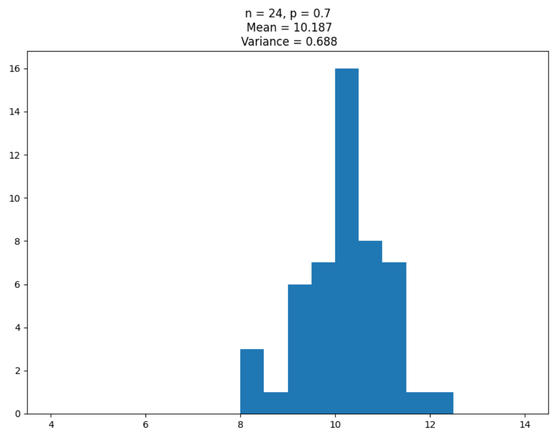
(d) No. of steps taken = 20 with probability of moving a step right = 0.5.



(e) No. of steps taken = 24 with probability of moving a step right = 0.5.



(f) No. of steps taken = 20 with probability of moving a step right = 0.7.



(g) No. of steps taken = 24 with probability of moving a step right = 0.7.

Histograms produced by the above code for expected final positions of objects against frequency, where n is the number of steps taken and p is the probability of the object moving one step to the right.

Histograms (d) and (e) have the same value of p and varying number of steps n.
Both histograms appear to follow a normal distribution, with (d) having a mean of 3.267 and a variance of 0.233 and (e) having a mean of 3.466 and a variance of 0.367. Again, increasing the number of steps has not had a significant effect on the mean but did increase the variance.

Histograms (f) and (g) have the same number of steps n respectively as (d) and (e) and varying values of p. They also appear to follow a normal distribution, with (f) having a mean of 8.462 and a variance of 0.559 and (g) having a mean of 10.187 and a variance of 0.668. Increasing p appears to have increased the mean and variance, but the mean has increased more significantly.

Additionally, due to the added constraint in this part, we see that no expected value for the final position is negative.

1.3

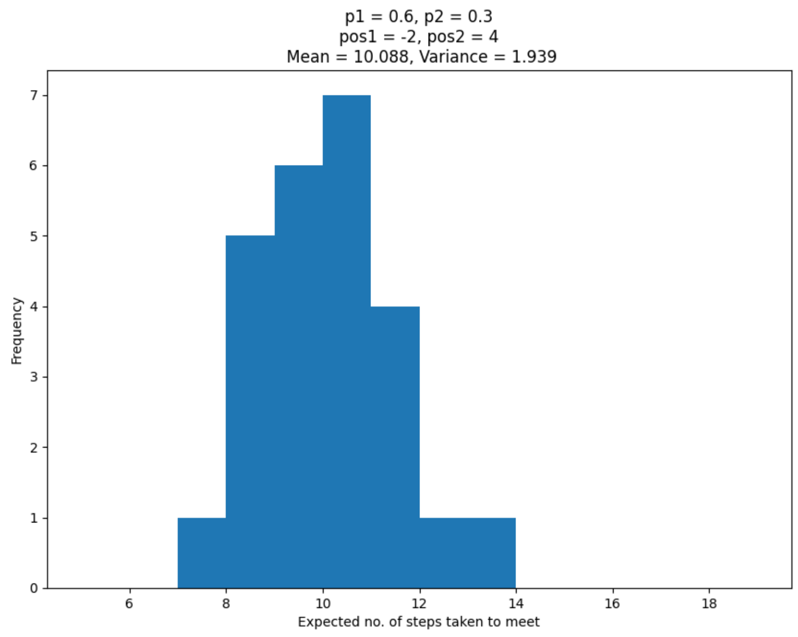
In this part, we are required to find the number of steps taken by two objects initialized at different points, and moving randomly to meet. The following function takes in the initial position of both objects $pos1$ and $pos2$ as well as their probabilities $p1$ and $p2$, generates random numbers and determines the number of steps taken to meet.

```python
def stepsToMeet(pos1,pos2,p1,p2):
    count = 0 #keeps count of number of steps taken for objects to meet
    while pos1 != pos2:
        rand = random.randint(1,100) #generating a random number in the range 1 to 100 to determine outcome
        if rand < p1*100:
            pos1 += 1 #move one step right
        else:
            pos1 -= 1 #move one step left
        rand = random.randint(1,100) #generating a random number in the range 1 to 100
        if rand < p2*100:
            pos2 += 1 #move one step right
        else:
            pos2 -= 1 #move one step left
        count += 1
    return count
```

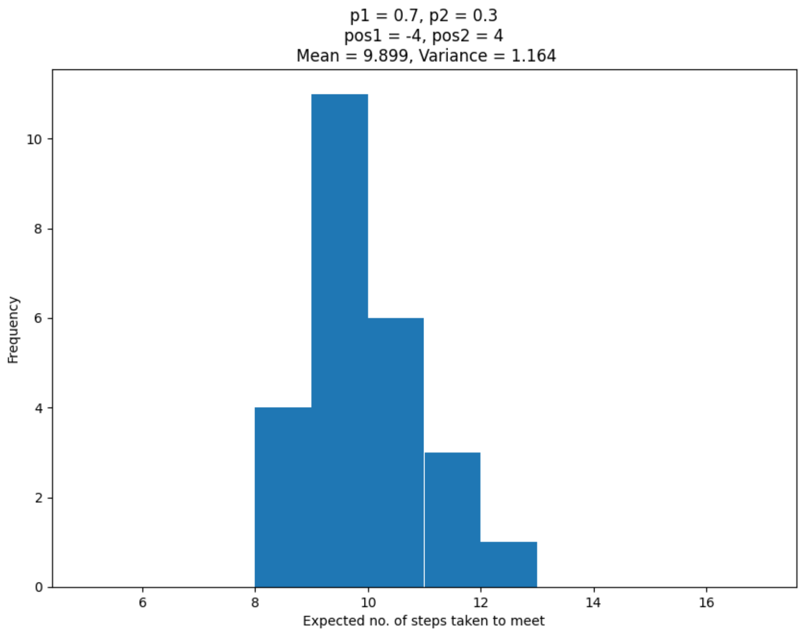
The following function uses the `stepsToMeet` function and calculates 25 expected values using 25 outcomes based on the values of $pos1, pos2, p1$ and $p2$. It then plots a histogram of these expected values as well.

```python
def main_13():
    '''Calculating and plotting expected outcomes for various combinations of n & p'''
    expected = []
    p1 = 0.7
    p2 = 0.3
    pos1 = -4
    pos2 = 4
    for i in range(25):  #calculating the expected value for each (n,p) for 25 iterations
        outcomes = []
        for j in range(25):
            outcomes.append(stepsToMeet(pos1,pos2,p1,p2))
        #calculating the average expected value for each(n,p)
        expected.append(sum(outcomes)/25) #appending the expected (average) value for each(n,p)
    #plotting a histogram of calculated expected values
    fig, ax = plt.subplots(figsize =(10, 7))
    ax.set_xlabel('Expected no. of steps taken to meet')
    ax.set_ylabel('Frequency')
    ax.hist(expected, bins = range(5,18))
    plt.title('p1 = '+str(p1)+', p2 = '+str(p2)+'\npos1 = '+str(pos1)+', pos2 = '+str(pos2)+'\n Mean = '+str(round(statistics.mean(expected),3))+', Variance = '+str(round(statistics.variance(expected),3)))
    plt.savefig("Q1_histograms/Q1.3 "+'_p1 = '+str(p1)+'_p2 = '+str(p2)+'_pos1 = '+str(pos1)+'_pos2 = '+str(pos2)+'(6).png')
    plt.show()
```

Histograms produced by the above code for expected number of steps taken for two objects walking randomly to meet, against frequency:



The above histogram shows that the expected number of steps to meet for two objects follows a normal distribution of mean 10.09 and variance 1.94 when they begin 6 steps apart with the given probabilities.



The above histogram shows that the expected number of steps to meet for two objects follows a normal distribution of mean 9.90 and variance 1.16 when they begin 8 steps apart with the given probabilities.

Q5: Hypothesis Testing

5.1

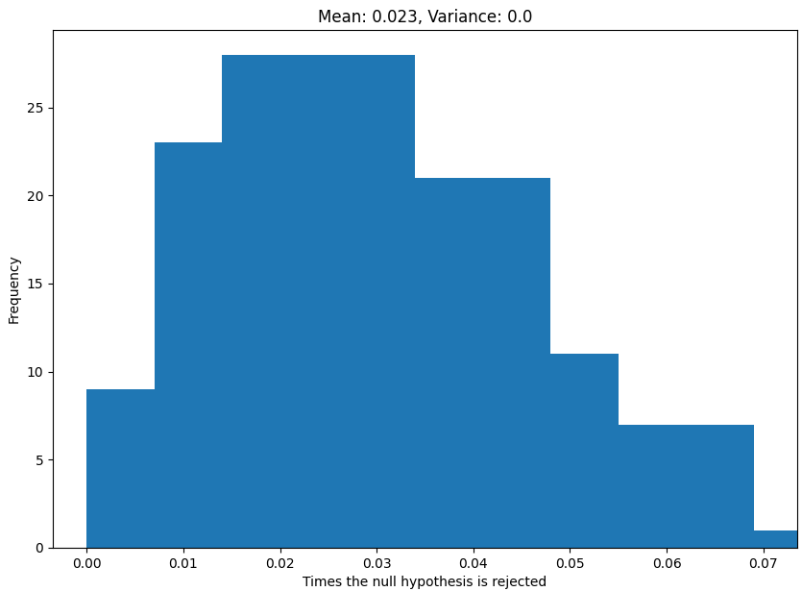
This question requires us to use hypothesis testing to determine whether the null hypothesis that a coin is fair is true or not.
The following function implements the simulation of a fair coin:

```python
def cointoss():
    #assuming 0 = T & 1 = H
    return(random.randint(0,1))
```

This function then uses the above function to simulate 10 coin tosses 100 times and finds the expected number of times the null hypothesis is rejected for each iteration:

```python
def simulate_tosses():
    expected = []
    for n in range(100): #to get 100 expected values
        rejected = []
        for j in range(100): #to get 1 expected value
            outcomes = []
            for i in range(10): #to get 10 outcomes/1 rejected value
                outcomes.append(cointoss())
            #check if hypothesis is accepted or rejected
            if sum(outcomes) == 0 or sum(outcomes) == 1 or sum(outcomes) == 10 or sum(outcomes
    ) == 9:
                rejected.append(1) #hypothesis rejected
            else:
                rejected.append(0) #hypothesis accepted
        expected.append(sum(rejected)/100)
    #plotting histogram
    fig, ax = plt.subplots(figsize =(10, 7))
    ax.hist(expected,width = 0.01)
    ax.set_xlabel('Times the null hypothesis is rejected')
    ax.set_ylabel('Frequency')
    plt.title('Mean: '+str(round(statistics.mean(expected),3))+', Variance: '+str(round(
    statistics.variance(expected),3)))
    plt.savefig("Q5_histograms/Q5.1.png")
    plt.show()
```

Histogram of expected values:



Probability we will reject the null hypothesis even though it is true:
Simulation-wise: According to the mean of the expected values, the probability is 2.3%.
Mathematically: We reject the null hypothesis if the probability of the outcomes is below the threshold. Since the probability of the outcomes being accepted is 0.95 and the probability of them being rejected is 0.05, the probability that we reject the null hypothesis even though it is true is also 0.05 which is equal to the threshold.

5.2.1

This question requires us to use hypothesis testing to determine the validity of the null hypothesis that the mean length of fish in a lake is 23.
The following function conducts a single hypothesis test by taking a sample of 30 fish and calculating the mean and standard deviation of their lengths to check the following condition to accept or reject the null hypothesis:

$$P(|S - u_0| \geq a) < 0.05$$

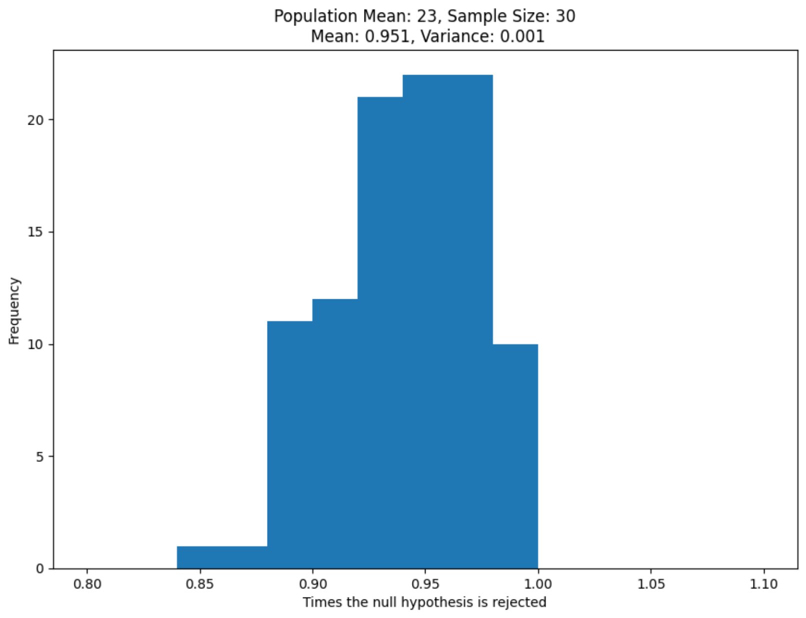
The function returns 1 if the hypothesis is rejected, and 0 if it is accepted.

```
def hypothesis_test(u_0, n):
    s = []
    for i in range(n): #catching a sample of 30 fish
        s.append(f.fish())
    s = np.array(s)
    std = s.std() #sample standard deviation
    mean = s.mean() #sample mean
    a = abs(mean - u_0)
    prob = 2*(stats.norm(u_0, std/math.sqrt(n)).cdf(u_0 - a))
    if prob < 0.05:
        return 1 #rejected
    else:
        return 0 #accepted
```

The following function performs 50 experiments and uses the average of their outcome to calculate a single expected value. It repeats this for 100 iterations and plots a histogram of the expected values.

```
def experiments():
    u_0 = 23 #population mean
    n = 30 #sample size
    expected = []
    for j in range(100): #getting 100 expected values
        outcomes = []
        for i in range(50): #conducting 50 hypothesis tests
            outcomes.append(hypothesis_test(u_0,n))
        expected.append(sum(outcomes)/50) #calculating one expected value from the 50 tests
    #plotting histogram of expected values
    fig, ax = plt.subplots(figsize =(10, 7))
    ax.hist(expected, width = 0.04)
    ax.set_xlabel('Times the null hypothesis is rejected')
    ax.set_ylabel('Frequency')
    plt.title('Population Mean: '+str(u_0)+', Sample Size: '+str(n)+'\n Mean: '+str(round(
    statistics.mean(expected),3))+', Variance: '+str(round(statistics.variance(expected),3)))
    plt.savefig("Q5_histograms/Q5.2.1.png")
    plt.show()
```

Histogram of expected values of the null hypothesis being rejected:



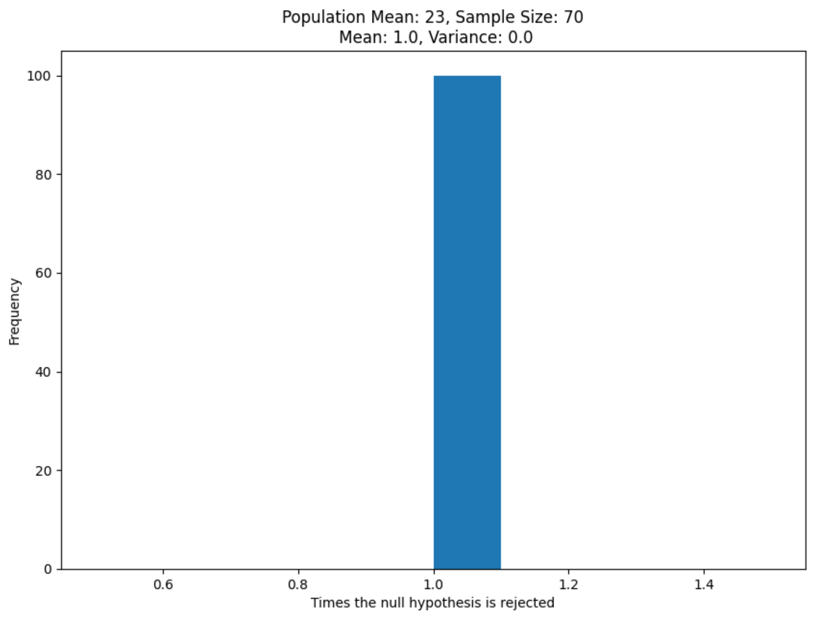
The histogram follows a normal disttribution with mean 0.951 and variance 0.001. According to this, the null hypothesis is rejected 95% of the times so it is false. A single hypothesis test may have been sufficient to accept or reject the null hypothesis as the variance is very low.

5.2.2

The following function calls the `hypothesis_test()` function defined in 5.2.1 using the same value for the population mean u_0 (23) but passing a value of 70 for the sample size n. It then performs 50 experiments and uses the average of their outcome to calculate a single expected value. It repeats this for 100 iterations and plots a histogram of the expected values.

```
def experiments_updated():
    u_0 = 23 #population mean
    n = 70 #sample size
    expected = []
    for j in range(100): #getting 100 expected values
        outcomes = []
        for i in range(50): #conducting 50 hypothesis tests
            outcomes.append(hypothesis_test(u_0,n))
        expected.append(sum(outcomes)/50)
    #plotting histogram
    fig, ax = plt.subplots(figsize =(10, 7))
    ax.hist(expected, width = 0.1)
    ax.set_xlabel('Times the null hypothesis is rejected')
    ax.set_ylabel('Frequency')
    plt.title('Population Mean: '+str(u_0)+', Sample Size: '+str(n)+'\n Mean: '+str(round(
    statistics.mean(expected),3))+', Variance: '+str(round(statistics.variance(expected),3)))
    plt.savefig("Q5_histograms/Q5.2.2.png")
```

Histogram of expected values of the null hypothesis being rejected:



Increasing the value of n made the variance become 0 where it was previously 0.001. In other words, the null hypothesis is rejected 100% of the times. A single hypothesis test may have been sufficient to reject the null hypothesis in this case as there is no variance in the outcome.

5.2.3

Determining the least value of n to ensure that the null hypothesis is not wrongly rejected more than 10 percent of the time using simulations.

The following is a function that returns the length of a fish following a normal distribution with mean 23 and standard deviation 3.

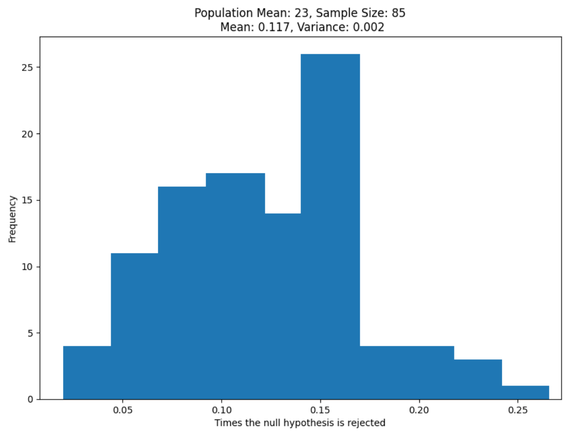
```
def my_fish():
    return np.random.normal(23,3)
```

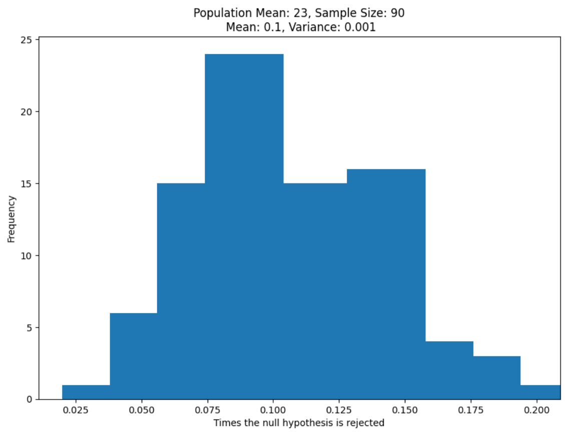
In order to calculate the expected number of times the null hypothesis is rejected, we use the following function which takes in the value of the population mean u_0 and sample size n and uses the `my_fish()` function defined above to create the sample for the test. It then conducts the test itself the same way as the previous parts but with a threshold of 0.1.

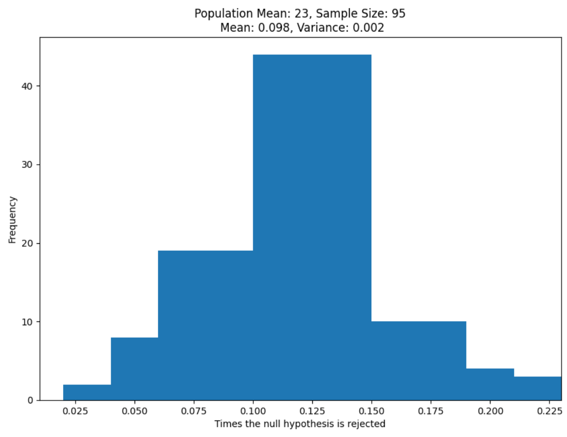
```python
def testing(u_0,n):
    s = []
    for i in range(n): #catching a sample of 30 fish using the new fish function
        s.append(my_fish())
    s = np.array(s)
    std = s.std() #sample standard deviation
    mean = s.mean() #sample mean
    a = abs(mean - u_0)
    prob = 2*(stats.norm(u_0, std/math.sqrt(n)).cdf(u_0 - a))
    if prob < 0.1:
        return 1 #rejected
    else:
        return 0 #accepted
```

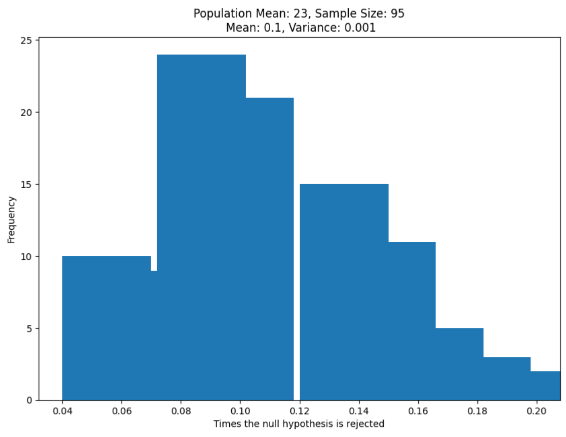
We then use the following function which calls the `testing()` function, performs 50 experiments and uses the average of their outcome to calculate a single expected value. It repeats this for 100 iterations and plots a histogram of the expected values.

```python
def simulation():
    u_0 = 23 #population mean
    n = 90 #sample size
    expected = []
    for j in range(100): #getting 100 expected values
        outcomes = []
        for i in range(50):  #conducting 50 hypothesis tests
            outcomes.append(testing(u_0,n))
        expected.append(sum(outcomes)/50)
    #plotting histogram
    fig, ax = plt.subplots(figsize =(10, 7))
    ax.hist(expected, width = 0.03)
    ax.set_xlabel('Times the null hypothesis is rejected')
    ax.set_ylabel('Frequency')
    plt.title('Population Mean: '+str(u_0)+', Sample Size: '+str(n)+'\n Mean: '+str(round(
    statistics.mean(expected),3))+', Variance: '+str(round(statistics.variance(expected),3)))
    plt.savefig("Q5_histograms/Q5.2.3_"+str(n)+".png")
    plt.show()
```









Conducting simulations with various values of n, we achieve the above histograms. In order to determine the least value of n to ensure that the null hypothesis is not wrongly rejected more than 10 percent of the times, we want a value of n that gives a mean of 0.1. According to the simulations, the value of n satisfying this condition is **90**.