Andrew Branum

CSE 622: HW 2

Problem 1:

Given the formulas, an estimation of pi can be derived from this formula: by multiplying 4 to the other side of the equation resulting in . To simplify this further we can assume that the area of the circle will consist of all the points that reside inside the circle and the area of the square will be all the points. We can then simplify the assumption of pi to . Finding the points that reside within a circle that has a radius of 1 can be found using the equation . Using these two equations in conjunction with N number of uniform numbers, the value of pi can be estimated.

For my experiment, I opted to use a wide range of different values of N consisting of [100, 1000, 10000, 100000, 1000000, 10000000, 100000000]. I created a function that accepts the number of points as a parameter, calculates the given number of random points, calculates the distance from the origin of the circle, keeps count of the points inside of the circle, and returns the final estimate. The x and y values of the points are randomly generated and are between -1 and 1 since we will assume the square’s corners reside at (1, 1), (-1, 1), (1, -1), (-1, -1). If then the point is within the circle and the value is incremented. After the loop runs N times, the final estimate is calculated and plotted.

Here is an example of one simulation run:

|  |  |
| --- | --- |
| N | π |
| 100 | 3.4 |
| 1000 | 3.012 |
| 10000 | 3.1408 |
| 100000 | 3.14504 |
| 1000000 | 3.142096 |
| 10000000 | 3.1411744 |
| 100000000 | 3.14173872 |

After running the simulation a few times, it becomes evident that with larger numbers of N, the value of pi is closer to its true value. Furthermore, the time it takes to run the simulation increases and it needs more computation power to do so. Smaller values of N tend to vary while the larger numbers of N tend to stay close to the true value of pi. The cutoff for spontaneous values seems to be N = 10,000. This gives some insight into how tradeoffs must be considered pertaining to each individual use case. Is the extra computation and time worth the extra decimal places?

A graph with a line and a point

Description automatically generated

Problem 2:

To complete a Chi-Squared Test on the given 20 numbers, 10 subintervals must be created from [0,0.1) to [0.9,1.0) and the count of how many numbers fall within each must be computed. After this is complete, we assume our Null Hypothesis (grades are uniformly distributed) and the Alternate Hypothesis (grades are not uniform). From here the expected frequency of each interval must be computed with where N is the total number of numbers. Using the expected frequency with the observed frequency we can compute the Chi-square statistic with this formula where O is the observed frequency. This value is then compared to the critical value at α = 0.05 via the Critical Values Chart. If the calculated Chi-square statistic is less than 0.05, the Null Hypothesis is rejected, and the data is not uniformly distributed. Elsewise, the data is uniformly distributed.

Setting up this simulation consisted of putting the given numbers into an array, creating 10 subintervals between 0 and 1, using numpy to dynamically count the numbers in each bin, calculating the expected frequency per bin in a uniform distribution, then using the chisquare function from scipy.stats to perform the Goodness-of-Fit test and plotting the results. The Kolmogrov-Smirnov Goodness-of-Fit test is also implemented with scipy using its built in function ks\_1samp() and passing in the given numbers.

For this experiment, the expected frequency ends up being 2. The Chi-Square statistic ends up being 6 and the p-value for that is 0.73992 (> 0.05) meaning that the null hypothesis cannot be rejecting and that the data is uniformly distributed. The same analysis occurs with the Kolmogorov-Smirnov test. The KS statistic end up being 0.14 and the p-value 0.77798.

Example Output:

A screenshot of a computer screen

Description automatically generated

For the second portion of this problem, the steps are slightly different. First, we must calculate the expected frequency for each grade with the formula where Oi is sum of a grade across semesters, Oj is the sum of the grades of the semester we are evaluating, and O is the sum of all grades in the dataset. For example, the expected frequency of grade A during this semester is (30\*59)/118 = 15. We then must compute the Chi-Square statistic using this formula where Oji is the observed frequency for grade I in semester j, Eij is expected frequency for grade I in semester j, and k is the number of grade categories. The Chi-Square statistic for each semester is then combined and compared to the critical value that has 4 degrees of freedom and alpha = 0.05.

The implementation of this simulation was relatively straight forward, using mainly mathematical equations to calculate values used in subsequent operations. Scipy’s chi2 function was used to find the critical value and p value dynamically.

After running the numbers through the equations, the Chi-Squared statistic comes out to be 8.34, the degrees of freedom are 4 (Number of Grades -1). With these two values we can find the critical value to be 9.488 for df = 4 and α=0.05. Since 8.34 < 9.488 we fail to reject the null hypothesis, and the grades aren’t seen as different.

A white background with black text

Description automatically generated

Problem 3:

If I had more time to work on the first assignment, I would improve my evasion technique, add a health attribute, vary the speed of the spaceships, and add more ships. The first thing I would do is change the evasion technique to not be random. Right now, it is activated depending on the time and randomly changes the direction of the ship in evasion mode. I would modify it so the ship changes direction when entering a “danger zone” rather than random time intervals to ensure that it attempts to evade as best as possible while positioning itself to attack as well. Furthermore, I would give each ship either health or lives to see how the ships interact with each other when it isn’t a “one hit kill”. Changing the speed of each ship dynamically can add interesting nuances to the evading and attacking techniques of the ships as well. Finally, adding more than two ships would add some more chaos to the simulation and would be interesting to see how the battle turns out.