Approximating the Value of Pi using the Monte Carlo Method implemented in Python

Kyla Bouldin

Abstract

The purpose of the experiment is to approximate the value of π by using the Monte Carlo Method. The hypothesis was that if the number of points increases, then the accuracy would improve. The methods and materials used made the experiment simple. After the Pseudo-Random Number Generator is written, the experiment is complete. The data and results supported the hypothesis. According to the results as more points were thrown, the more accurate the approximation to π got.

Introduction

In this experiment the Monte Carlo Method will be implemented using a pseudo-random number generator written in Python. "The Monte Carlo Method" is a method of using random numbers, or statistics, to solve problems. A Pseudo-Random Number Generator is an algorithm that makes up numbers that should have no pattern; being "random". An algorithm is a set of rules that is followed to do a certain thing, in this case, to generate the series of random numbers.

The question being asked is **how does the accuracy improve as more points are used in the Monte Carlo Simulation.** The independent variable is the number of points being used. The dependent variable is the calculated value of pi (π). The constants in this experiment are the areas of the square and the circle, the random generator code, and the actual number of pi. The hypothesis is that if more points are used, then the accuracy will improve. To get the approximation of π, the number of points in the circle is divided by the total number of points, and then is multiplied by four. There should be about 79% of the points in the circle, but the

random generator will make numbers above and below it. **Figure 1** shows an example with 900 dots total, and 709 of them are inside the circle. The ratio of 709/900 should be near 0.79 (or 79%), but with the random number generator, it won't be exactly 79%.

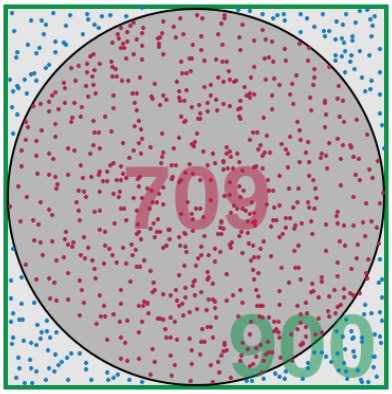

Figure 1 – Schematic diagram of calculating π using circle to square area ratios approximated by randomly generated points.

https://upload.wikimedia.org/wikipedia/commons/thumb/1/18/Circle_area_Monte_Carlo_integration.svg/420px-Circle_area_Monte_Carlo_integration.svg.png

The Pseudo-Random Number Generator Code is used to pick random numbers for x and y coordinates and plot them on a square with an area of $4r^2$. Inside the square will be a circle with an area of πr^2. The Monte Carlo Method puts dots in the square, TotalDots, and of those, CircleDots are also inside the circle. The area of the square is $4r^2$, which is proportional to TotalDots. The area of the circle is πr^2, which is proportional to CircleDots. So, the ratio of the areas is

(1) TotalDots/CircleDots = $4r^2/\pi r^2$,

which simplifies to

(2) π=4[CircleDots/TotalDots].

So, if the Pseudo-Random Number Generator is run to thrown a total number of points, TotalDots, the value of π can be calculated.

Materials and Methods

The first thing required to do this project is to write the Pseduo-Random Number Generator Code on any computer that has Python. The programs used in this experiment were TextWrangler, a text editor used to write the Python code, and the pygame graphics library to add animations and pictures. The Pseudo-Random Number Generator Code used in the experiment was as follows:

```
import random
import math
import pygame
import sys


def roundone():
        test_runoutoffive = 1
        numoftotalpoints = 0
        totalnumberoftestruns = 5
        testpointsincircle = 0
        numberofpointsthrown = 10 #starting number
        while numberofpointsthrown <= 100000: # the 100000 is the number to go up to in increments of 10
                while test_runoutoffive <= totalnumberoftestruns: #each test run this happens
                        w = 500
                        h = 500
                        screen = pygame.display.set_mode((w, h))
                        circle = pygame.draw.circle(screen, (0,0,255), (w/2,h/2), 250, 1)
                        pygame.display.flip()
                        while numoftotalpoints < numberofpointsthrown: #the 100 has to multiply for each thing
                                x = 500 * random.random()
                                y = 500 * random.random()
                                d = math.hypot(x-(w/2), y-(h/2))
```

```
                    if d < 250:
                        point = pygame.draw.circle(screen, (0,0,255), (x,y), 0, 0) #inside circle (blue)
                        totalpointsincircle += 1
                        numoftotalpoints += 1
                    elif d > 250:
                        point = pygame.draw.circle(screen, (255,0,0), (x,y), 0, 0) #outside circle (red)
                        numoftotalpoints += 1
                    pygame.display.flip()
                sys.stdout.write('\a')
                sys.stdout.flush()
                print "This is trial number", test_runoutoffive
                print "There were", numoftotalpoints, "thrown"
                print "Approx to pi is", 4.0 * totalpointsincircle/numoftotalpoints
                pygame.time.wait(1000) #makes the screen pause for 5 seconds
                numoftotalpoints = 0
                totalpointsincircle = 0
                test_runoutoffive += 1
            numberofpointsthrown *= 10
            test_runoutoffive = 1
print roundone()
```

Next, execute the program and record the number it prints out which is the approximate number to pi. As an addition also take screenshots after each trial. With the results make a graph.

Results

The data table shows the results from each trial in the experiment. The data table shows how many points were thrown, what trial number it was, and what the approximation of pi was. The approximation of pi was based on the equation (2) above: 4*(total points inside circle/number of total points). In figure 2, the values of Y get closer showing that the number for the approximated values of pi get more exact to the red line which is pi. In figure 3, the values of

Y decrease towards a relative error of zero. At first the accuracy improves quickly, but then the relative error only improves slowly.

Number of Points Thrown	Trial Number	Approx to pi	Average	Relative error (π-average/π)
10	1	3.6		
10	2	2.4		
10	3	2.8		
10	4	2.8		
10	5	2.8	2.88	0.083268563
100	1	2.96		
100	2	3.04		
100	3	3.28		
100	4	2.92		
100	5	3.4	3.12	0.006874276
1000	1	3.164		
1000	2	3.24		
1000	3	3.204		
1000	4	3.16		
1000	5	3.192	3.192	-0.01604401
10000	1	3.1444		
10000	2	3.1064		
10000	3	3.1364		
10000	4	3.1364		
10000	5	3.1536	3.13504	0.002086901

100000	1	3.14768		
100000	2	3.14332		
100000	3	3.1298		
100000	4	3.14684		
100000	5	3.13888	3.143304	-0.000543609

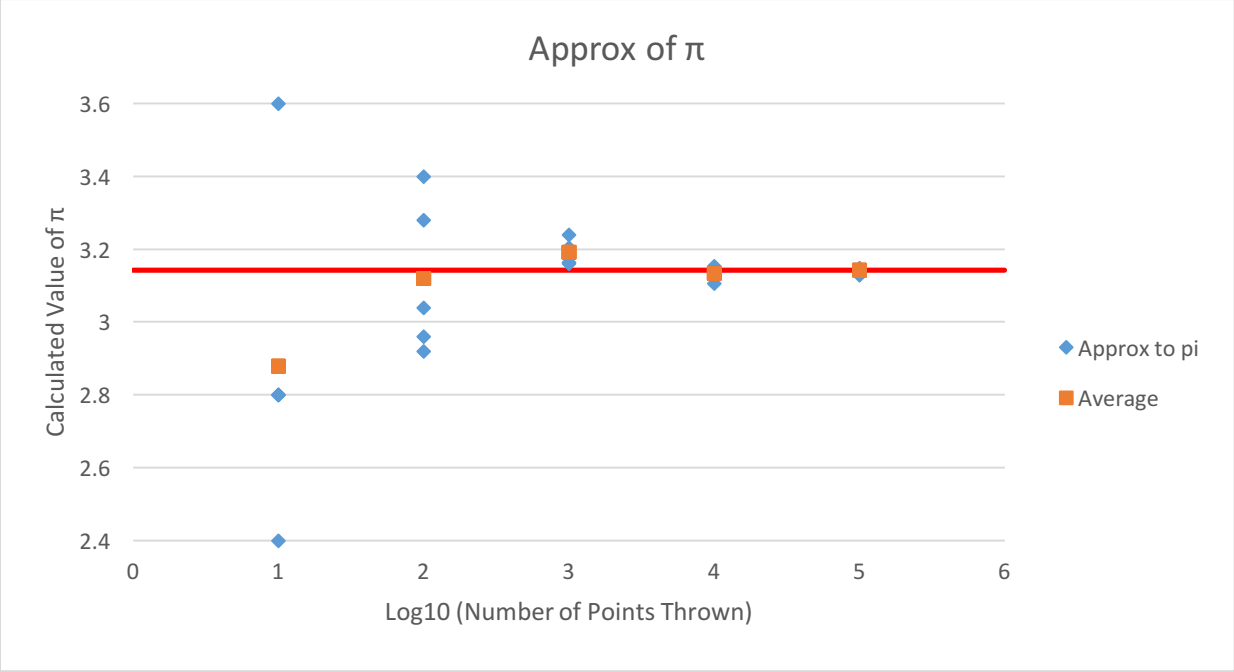

Figure 2 – the calculated value of π versus the log 10 of the number of points thrown. For each number of points thrown there were 5 different trials (blue diamonds), and the average of the 5 trials (red square), but on the graph not all 5 points are visible at each trial because some points overlap.

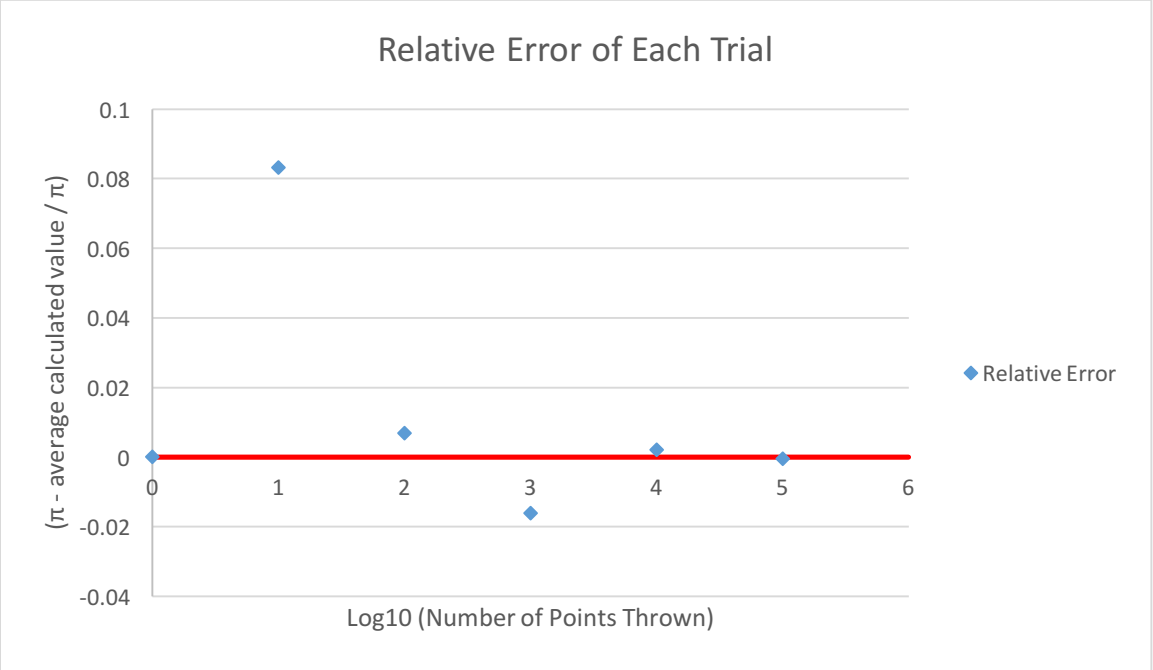

Figure 3 – the relative error of the value π versus the log 10 of the number of points thrown. As the number of points is increased, the relative error decreases, first quickly, and then much more slowly.

Conclusion

The hypothesis states that if more points are used then the accuracy will improve. The results support the hypothesis. As more points were thrown, the approximation to π got more precise. Also as the points increased, the relative error decreases first quickly, then much more slowly. At the trial with 1,000 throws, the error increases, but this is probably due to the random nature of the Monte Carlo method.

Literature Cited

"An Introduction to Monte Carlo Methods." *University of Nebraska – Lincoln Chmeistry.* Web. 19 Oct. 2009. <http://www.chem.unl.edu/zeng/joy/mclab/mcintro.html>.

Paul E. Black, "pseudo-random number generator", in *Dictionary of Algorithms and Data*

Structures[online], Paul E. Black, ed., U.S. National Institude of Standards and Technology. 21 May 2007. (accessed October 13, 2009) Available from: http://www.itl.nist.gov/div897/sqg/dads/HTML/pseudorandomNumberGen.html