Pi Calculator

Maxim Veytsman

June 7, 2001

1 Introduction and Calculations

1.1 The Monte Carlo Method

This article aims to create a program that can calculate π using the Monte Carlo[1] method. The Monte Carlo method is a simple way of calculating values (such as a value like π) by using random numbers. For Pi, it works like this, there is a square dart board with each side 2 units in length, on which is a circle with a radius of 1 unit. A person starts throwing darts at the board, if they miss the board it is not counted, if they hit the circle they get a point. e

1.2 Calculating Pi

First of all, let's calculate the probability of the dart landing in the circle. The area of the circle is calculated by the formula $A_1 = \pi r^2$ Since $r = 1$, we get $A_1 = \pi$. The formula for the area of a square is $A_2 = l^2$, since $l = 2$, $A_2 = 4$. Now the probability of the dart landing in the circle[2] is $\frac{A_1}{A_2}$, this can also be written as $\frac{\pi}{4}$. We now know that the probability of the dart landing in the square is 4π, so we can calculate π by simulating this action and multiplying the probability by 4.

[1] Monte Carlo is a large casino city in Europe similar to Las Vegas.

[2] This is assuming we only count the shots that land in the square

1.3 Did it Land in the Circle

In order to calculate the dart's coordinates, we must imagine the square board as a coordinate plane, the center of the square is (0,0). In order to see if the dart is in the circle we must measure the distance between the coordinates of the dart and (0,0), the radius. The distance can be calculated using the Pyphagorian Theorem. Let's draw a rectangle whose 2 sides are the distances between the coordinate of the dart and (0,0) on x and y. We then draw a line from the point where the dart landed to (0,0), and use this as the hypotenuse of a triangle formed by 2 of the rectangles sides. The Pyphagorian Theorem states that $A^2 + B^2 = C^2$. Which in terms or coordinates would mean that $r^2 = x^2 + y^2$. So to see if the dart has landed in the circle, we put both of it's coordinates in the power of 2, add the results and see if it is less than 1.

2 Simulating the Dart Throwing

2.1 Commented Code

In the previous section we figured out how to calculate π using the method. Here we will write a perl script to simulate throwing darts at a board.

```
1 #!/usr/bin/perl
2 use strict;
3 my ($cycles, $i, $yespi, $pi) = 0;
```

Here we tell the computer we are using perl, then declare a few variables[3]. '$cycles' is the amount of times we throw the dart, '$i' is a counter for the amount of times we throw the dart, '$yespi' records the amount of times the dart landed in the circle, and '$pi' is the value of π. For now, all of these are set to 0.

```
4 srand;
5 print "Please enter the amount of cycles:";
6 chomp($cycles = <STDIN>);
```

We start out by telling the computer that each time we ask for a random number, it will do an opperation to the current time to get a number in the range we set, this is called by 'srand;'. We then ask the the user for the amount of times the dart will be thrown and record this number as '$cycles'.

[3]For the purpose of this article, all variables begin with '$'.

```
7  while ($i <= $cycles) {
8      my ($x, $y, $cdnt) = 0;
9      $x = rand;
10     $y = rand;
11     $cdnt = $x**2 + $y**2;
12     if ($cdnt <= 1) {
13         ++$yespi;
14 }
15 ++$i;
16 }
```

The first part of this chunk of code starts a while loop, this means that everything between the '{' and '}' will be repeated until the conditions between the "*and*" are no longer met. We declare the variable '$cdnt' which is the distance between the dart and (0,0), '$x' which is the x coordinate of the dart, and '$y' which is the y coordinate of the dart. The x and why coordinates are assigned random values between 0 and 1[4], and calculate the distance between the dart's coordinate and (0,0) using the Pyphagorian Theorem. Then we call an if statement, if the variable '$cdnt' is less than or equal to 1, we add 1 to the number of times the dart landed in the circle using the statement '++$yespi'. The last statement adds 1 to the counter. This is so the loop will only repeat itself the amount of times the user specified, because '$i' will eventually be greater than '$cycles' and the loop won't occur.

```
17 $pi = ($yespi / $cycles ) * 4;
18 print "Pi is $pi\n";
```

This last chunk assign $4\frac{A_1}{A_2}$ to '$pi' to get π. It then prints the phrase "Pi is [the value of the variable '$pi']" to standard output (the screen).

2.2 Uncommented Code

Here is the full, uninterrupted code:

```
1 #!/usr/bin/perl
2 use strict ;
3 my ($cycles, $i , $yespi , $pi) = 0;
4 srand;
```

[4]This limits the results to only the I quadrant, but since this is a fraction, dividing each side by 4 will not effect the result.

```
5  print "Please enter the amount of cycles:";
6  chomp($cycles = <STDIN>);
7  while ($i <= $cycles) {
8      my ($x, $y, $cdnt) = 0;
9      $x = rand;
10     $y = rand;
11     $cdnt = $x**2 + $y**2;
12     if ($cdnt <= 1) {
13         ++$yespi;
14 }
15 ++$i;
16 }
17 $pi = ($yespi / $cycles ) * 4;
18 print "Pi is $pi\n";
```

3 Conclusion

3.1 Accuracy

I found this method to be accurate at over 10,000 cycles.

3.2 The value of Pi

After 1,000,000 cycles, the program said that π is equal to 3.14202[5], as compared to the real value: 3.14159.

3.3 To Do

- Add graphic of radius and of the dart board
- Add graph of the change in accuracy as the amount of cycles changes

[5]This value changes every time the program is run, because the program uses random values.