Monte Carlo Simulation: Estimating Pi

Aditya Gupta

February 2023

Introduction

In this assignment, we find an estimate for the value of π using the Monte Carlo method and compare it against other historical methods for computing π. We will also compare our estimate against the actual value of π.

Monte Carlo Method

Description

If we take a circle with radius r inscribed inside a square of side 2r, we have Area of Circle = πr^2 and Area of Square = $4r^2$. Now imagine that we throw a bunch of darts at this square, then the probability that a dart would hit in the circle inscribed inside the square is:

$$\text{P(dart)} = \text{Area of Circle/Area of Square} = \pi/4$$

Thus we can get an estimate for the value of π using:

$$4*\text{P(dart)} = \pi$$

Implementation

We first write a function that is used for simulating the act of throwing a single random dart at the square containing the circle inside of it. The function dart/1 is used for this purpose. It takes the radius of the circle as an argument. After this, it generates random values for a point at which the dart might hit using *Enum.random(0..r)*. Then it is checked if the distance of this point from the centre of the circle is greater than the radius of the circle or not to determine if the dart hits at a point inside the circle or outside it in the square.

Now to throw a large number of darts we create another function called round/3. This function returns the number of darts that actually hit inside the circular

region. It takes the number of darts thrown, the radius of the circle and an accumulator for storing dart hits in the area of interest as arguments. Thus in this function, we increment the value of the accumulator whenever a dart hits inside of the circle otherwise the function is called again recursively with the dart count being decremented by one. When we run out of darts we just return the number of darts that hit inside the circle.

```
 def dart(r) do
    x = Enum.random(0..r)
    y = Enum.random(0..r)
    :math.pow(r,2) >  :math.pow(x,2) + :math.pow(y,2)
end

def round(0, _, a) do a end
def round(k, r, a) do
    if dart(r) do
      round(k - 1, r, a+1)
    else
      round(k - 1, r, a)
    end
end
```

After we have these functions set up we can finally set up a test in order to estimate the value of π. We create another function called rounds/5.

```
#The function below can also be written with just 4 arguments -> k, j, r, a
# For the case where we double "j" each iteration

def rounds(k, j, r) do
    rounds(k, j , 0, r, 0)
end
def rounds(0, _, t, _, a) do 4*a/t end # def rounds(0, n, _, a) do 4*a/n end
def rounds(k, j, t, r, a) do # def rounds(k, n, r, a) do
    a = round(j, r, a)  # a = round(n, r, a)
    t = t + j   # n = n*2
    pi = 4*a/t  # pi = 4*a/n
    IO.inspect("Pi estimate = #{pi}")
    IO.inspect("Difference from actual value of Pi = #{pi - :math.pi()}")
    rounds(k-1, j, t, r, a) # rounds(k-1, n, r, a)
end
```

Result

I tested the above functions for values between 1000 to 40 million. Even with 40 million we only manage to get an accuracy of up to 4 decimal places. One observation that I made was that the accuracy of the pi value increased as we raised the value of the radius. This is probably because with a smaller radius

the points where the darts could hit is somewhat limited. I also attempted to do the rounds/5 function such that it increases the number of darts thrown each time but it seemed to take far too long and didn't provide better results than the previous approach for me. The best approximation I could get for that approach was 3.1385 which is only accurate up to 1 decimal place.

```
iex(12)> MC.test
For 1,000 darts and the same radius: 3.1516
For 10,000 darts and the same radius: 3.14036
For 100,000 darts and the same radius: 3.142984
For 1,000,000 darts and the same radius: 3.1418588
For 10,000,000 darts and the same radius: 3.14166872
For 40,000,000 darts and the same radius: 3.14156535
:ok
```