Lab 9 - Monte carlo

Weichen Chai

Spring 2023

Introduction

This paper seeks to answer the questions presented in"Monte carlo" for ID1019. In this assignment, we will explore how the functional language Elixir can be utilized to explore methods in which to find the mathematical value of pi in a detailed manner using the Monte carlo method.

Tasks 1

The idea of the Monte Carlo method is to have a large amount of generated numbers, or in the assignment "darts" which falls on a generated circle, pi can then be calculated through analysing the area of a circle in an area of a square.

To begin with, we should implement a function that "throws darts" at a square and continuously estimates the value of pi, this dart function should take in number which is the radius, of course, the more darts we throw the more accurate the result will be. In this assignment, we are asked to beat have an estimation that exceeds six decimals. In order to implement random variables in the dart function, we can use the Enum.random function which generates a random number. Then we would check if the numbers generated are inside the arch, shown below:

```
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)
   //to the power of 2
 end
```

Calling this function as it is will result in a true or false return variable.

As stated before, to get as close to the estimated value as possible, we want to throw a large number of darts, which requires for us to define a **round** function. The function in question throws k number of darts. This can be done with a basic accumulator. The function is as shown below:

```
def round(0, _, a) do a end
// return a
def round(k, r, a) do
    if dart(r) == false do
        round(k - 1, r, a)
// when it is outside the arch, k number of dart decreases
    else
        round(k - 1, r, a + 1)
// when it is inside the arch acculumator increments
    end
  end
```

Next, we will attempt to estimate the value for pi after each round. And we will find the error when comparing to pi by using the function math.pi() and subtract it to the value that we have received. Furthermore, from evaluation, we know that 4*a/t should be used due to the fact that we are comparing the area of the circle to the area of the square, which is:

$\frac{\pi \cdot r^2}{4r^2}$ After simplification, we get $\frac{\pi}{4}$ This ratio is representative of number of points generated inside the circle compared to total number of points generated. Of course, to find π we simply rearrange the equation to 4 multiply to the comparison stated above: $4 * \frac{Acculumatedhits}{TotalDarts}$.

In code form, it is shown below.

```
...
// t is the total amount of darts thrown
// k is amount of times the round function is called
  def rounds(0, _, t, _, a) do 4*a/t end
  //from equation derived above, we know a is number of hits
  // and t is the total number of darts
  def rounds(k, j, t, r, a) do
    a = round(j, r, a)
    t = t + j
    pi = 4*a/t
    // This is our estimated pi
    :io.format("Estimated = ~.6f, Difference = ~.6f\n", [pi, (pi - :math.pi())])
    // comparison with value of estimated pi to actual pi
    // print to 6th digit
    rounds(k - 1, j, t, r, a)
  end
end
```

When executing the code we realize that when the radius is too small, it will not capture an accurate representation of pi, so when testing we can have :

```
Monte.rounds(5, 10000000, 100000)
```

This gives quite an accurate result closing on the 6th decimal of pi, however, it runs very slowly, taking a long time to return the result.

Summary

The exercise proposed under the summary task is for us to approximate pi with Leibniz formula, which is faster executed than the Monte Carlo method, as Monte Carlo is not initially used to estimate the value of pi. From the Leibniz formula:

```
4 * Enum.reduce(0..1000, 0, fn(k,a) -> a + 1/(4*k + 1) - 1/(4*k + 3) end)
```

We get a rough estimate of pi, even though not too accurate, however, if we change the equation given to us and alter the values from 1000 to 10000000, it will become more accurate but still quite fast, or at least faster than using the Monte Carlo method.