ID1019: Approximation of π with the Monte Carlo method

Alexander Lundqvist

Spring Term 2023

Introduction

The mathematical constant π is an irrational number that has been known for thousands of years. It represents the ratio between the circumference and diameter of a circle, and has been of great interest to mathematicians throughout history. Despite its significance, approximating the true value of π has proved challenging due to its irrational nature and the limited computational tools available for much of history.

In this report we are tasked with trying to approximate the value of π by using the Monte Carlo method which is a statistical technique that uses repeated random sampling to simulate complex systems or problems. To achieve this goal, we will create random points inside a square and determine the number of points that lie within a quarter-circle of radius `r`, which is centered at the origin. By comparing the ratio of the number of points that are within the quarter-circle to the total number of points generated, we can come up with an approximation of the value of π.

Method

The instructions for the assignment provided the basic code skeleton for the solution. The basic functions we will need are `dart/1` that takes a radius `r` as input and generates `true` or `false` depending whether the dart hits inside the radius of the circle or not.

We also need to define a function `round/3` which represents an amount of darts thrown at the board. The function takes a number of darts `k`, a radius `r`, and an accumulated number of hits `a` as input and adds the number of hits inside the circle to the accumulator `a`.

Finally we are given a function `rounds/3` which will be used to run the approximations. It is accompanied by the function `rounds/5` which we will implement ourselves. This function has two additional parameters which are two accumulators `t` which are the current amount of darts thrown and `a` which is the amount of darts that has hit inside the circle so far.

Result

How many darts do you need before your can for sure state that you have better value; how would you know if you did not have :math.pi()?

As an initial test, I ran the approximation with 100 darts and a radius of 100. The result was far from ideal but it shows that we can get somewhat close to π even with small numbers.

```
Our approximation: 3.3200000000 | Error = 0.1784073464
Our approximation: 3.2800000000 | Error = 0.1384073464
Our approximation: 3.2266666667 | Error = 0.0850740131
Our approximation: 3.2200000000 | Error = 0.0784073464
Our approximation: 3.2240000000 | Error = 0.0824073464
Our approximation: 3.2333333333 | Error = 0.0917406797
Our approximation: 3.1885714286 | Error = 0.0469787750
Our approximation: 3.1900000000 | Error = 0.0484073464
Our approximation: 3.1866666667 | Error = 0.0450740131
Our approximation: 3.1760000000 | Error = 0.0344073464
```

I then started to increase the different values to see when I would beat Archimedes. I started by just repeatedly increasing the amount of darts while keeping the radius at 1,000 however the results were pretty much the same. The higher amount of darts did however produce a more consistent error margin across all rounds.

```
iex(11)> ApproximationOfPi.rounds(10, 1000, 1000)
Our approximation: 3.1800000000 | Error = 0.0384073464
Our approximation: 3.1680000000 | Error = 0.0264073464
Our approximation: 3.1533333333 | Error = 0.0117406797
Our approximation: 3.1300000000 | Error = 0.0115926536
Our approximation: 3.1368000000 | Error = 0.0047926536
Our approximation: 3.1513333333 | Error = 0.0097406797
Our approximation: 3.1400000000 | Error = 0.0015926536
Our approximation: 3.1410000000 | Error = 0.0005926536
Our approximation: 3.1377777778 | Error = 0.0038148758
Our approximation: 3.1336000000 | Error = 0.0079926536

iex(14)> ApproximationOfPi.rounds(10, 1000000, 1000)
Our approximation: 3.1383360000 | Error = 0.0032566536
Our approximation: 3.1386000000 | Error = 0.0029926536
Our approximation: 3.1394760000 | Error = 0.0021166536
Our approximation: 3.1396470000 | Error = 0.0019456536
Our approximation: 3.1393552000 | Error = 0.0022374536
Our approximation: 3.1396733333 | Error = 0.0019193203
Our approximation: 3.1397720000 | Error = 0.0018206536
```

```
Our approximation: 3.1398585000 | Error = 0.0017341536
Our approximation: 3.1398515556 | Error = 0.0017410980
Our approximation: 3.1395804000 | Error = 0.0020122536
```

I kept the amount of darts to 1,000,000 and started to increase the radius. When having a radius of 100,000 I could reliably beat Archimedes approximations.

```
Our approximation: 3.1422640000 | Error = 0.0006713464
Our approximation: 3.1422500000 | Error = 0.0006573464
Our approximation: 3.1413280000 | Error = 0.0002646536
Our approximation: 3.1413340000 | Error = 0.0002586536
Our approximation: 3.1418120000 | Error = 0.0002193464
Our approximation: 3.1417400000 | Error = 0.0001473464
Our approximation: 3.1417822857 | Error = 0.0001896321
Our approximation: 3.1419135000 | Error = 0.0003208464
Our approximation: 3.1417097778 | Error = 0.0001171242
Our approximation: 3.1416612000 | Error = 0.0000685464
```

For the The assignment suggested modifying the `rounds/5` function to double the amount of darts each round and I did implement it as `rounds2/5` however I didn't consider it necessary since even when running 10,000,000 darts and radius 100,000 I got up to 4 decimals reliably.

```
Our approximation: 3.1412004000 | Error = 0.0003922536
Our approximation: 3.1414800000 | Error = 0.0001126536
Our approximation: 3.1415829333 | Error = 0.0000097203
Our approximation: 3.1416490000 | Error = 0.0000563464
Our approximation: 3.1415564800 | Error = 0.0000361736
Our approximation: 3.1414964000 | Error = 0.0000962536
Our approximation: 3.1414800000 | Error = 0.0001126536
Our approximation: 3.1414396500 | Error = 0.0001530036
Our approximation: 3.1414132444 | Error = 0.0001794091
Our approximation: 3.1415392800 | Error = 0.0000533736
```

The last test at these numbers took several minutes so it is safe to say that this isn't an efficient method for very accurate approximation as is also mentioned in the assignment. I did however get a "personal best" in this run with π `= 3.1415829333 (Error = 0.0000097203)` which is just 1 decimal below Zu Chongzhi.

We can compare it to summing the Leibniz formula which took around 3 seconds and also produced an approximation accurate to 6th decimal I can see why Monte Carlo isn't optimal for this.

```
iex(21)> 4 * Enum.reduce(0..1000000, 0, fn(k,a) -> a + 1/(4*k + 1) - 1/(4*k + 3) end
3.1415921535902243
```

```
iex(22)> :math.pi()
3.141592653589793
```

The entire code is available at this github repository.