Calculating Pi Using the Monte-Carlo Method

Emil Göransson

Spring Term 2023

Introduction

This report will cover the implementation of how you can calculate pi using the Monte-Carlo method.

The implementation

The Monte-Carlo method is basically a way of calculating things based on a radius and a random. If random is inside the radius we increment a counter counting the amount of times random is "inside" the radius. To calculate pi three functions is used: dart/1, rounds/3 and rounds/5.

dart/1 is the function used to determine if a random is inside the radius. It either returns true or false based on the previous statement. The code looks like the following:

```
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
```

It utilizes the Pythagoras theorem to determine if its outside or inside the radius.

round/5 is what is used to calculate & print the approximated Pi-value as well as its Error compared to the more accurate representation :math.pi.

```
def rounds(0, _, t, _, a) do
  4 * a / t
end
def rounds(k, j, t, r, a) do
  a = round(j, r, a)
  t = t + j
```

```
    pi = 4 * a / t
    :io.format("pi = ~14.10f, error = ~14.10f\n", [pi, pi - :math.pi()])
    rounds(k - 1, j, t, r, a)
  end
```

The next part of the solution is the actual function we call when we want to run the program. It's basically a helper function that calls rounds/5 with some arguments for us with some differences based on if dart/1 hits or not. rounds/3 is called like the following: rounds("number of rounds", "number of randoms per round", "radius") The code is very simple and is the following:

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

Running the code and some conclusions

After running the code with some difference sized arguments I can tell that its very inefficient at what it does. While it does give a decent approximation it is by no means a very precise one unless you run extremely high amounts of rounds and a high radius, however that takes ages. I found that running a high radius gives a decent approximation up to the 4th decimal. E.g running Carlo.rounds(1000, 10000, 1000000) gives 3.1416804 which is not especially precise however it doesn't take a very long of a time since only 1000 rounds are ran.