Monte Carlo and Π

Axel Månson Lokrantz

Spring Term 2023

Introduction

The task for this assignment was to compute an approximation of pi using the Monte Carlo method. The program was written together with Daniel Dahlberg at KTH.

The task

One way of estimating the value of pi (3.141592...) is by using the Monte Carlo method. The idea is to draw a square with an inner circle, generate a large number of random points (darts, in this program) within the square and count how many hit the enclosed circle. The area of a circle is $A = 2\pi r$. The area for the outer rectangle then becomes $A = (r2)^2 = 4r^2$. If we now divide the area of the circle with the area of the triangle we get the expression $2\pi r/4r^2$ which can be simplified to $\pi/4$. Similarly the ratio between the darts that hit the inner circle and all the darts that has been thrown can be used. Hence, we get the following formula to estimate pi, $\pi = 4*$ (the number of darts in the circle/total number of darts).

To randomly generate a dart that hits either within the arc of the circle or the rectangle we use `Enum.random` which will generate a number between 0 and radius of the circle. Normally when calculating the hypotenuse we use the Pythagorean theorem $a = \sqrt{b^2 + c^2}$, however to avoid calculating the expensive root calculation for every dart we can rewrite the expression to $a^2 = b^2 + c^2$.

```
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
```

Next we define a function for one round, where `k` is the number of darts, `r` the radius and `acc` our accumulated number of hits.

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

The dart function will either return true or false. If returned true, we decrease the number of darts by 1 and increase our accumulator by 1. If returned false we only decrease the number of darts by 1. The function repeats recursively until our base case is met where we do not have any darts left and return the accumulator. To set up a benchmark for testing we implemented another function `rounds`.

```
 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(k, j, t, r, a) do
    a = round(j, r, a)
    t = t + j
    pi = 4*(a/t)
    :io.format("Estimate: ~w  Difference: ~w \n",
    [pi, (pi - :math.pi())])
    rounds(k-1, j*2, t, r, a)
  end
```

The benchmark runs `k` number of rounds, with `j` amount of darts per round and a circle radius of `r`. For every round we get a new estimate value and the difference between our estimated value and the actual value of pi. We managed to reliably get an estimation with a precision of five decimals with this program, however the testing is time consuming and take too long to be justified as a solution for finding more precise values of pi. A more efficient way of calculating pi is by summing the Leibniz formula.

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

Which gave us an estimated value of pi with a precision of six decimals within a couple of seconds.