Pi Estimation - Programming Problem

Paul Sæther Knutson

April 14, 2017

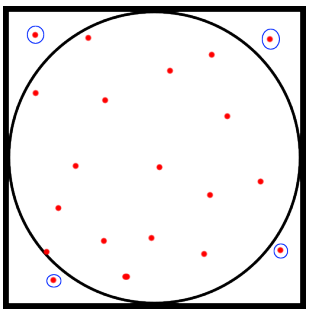
---

1 Problem

Put a circle inside a square, place (random) dots on the square. Use to estimate π, in some way.

2 Solution

Use the probability of random the dots "landing" inside or outside the circle to calculage the area of the circle. Find a formula for π based on the circle's area, and make a program to estimate the probability, thus estimating the value of π.



The dots with blue around them lands outside the circle, and estimates the area outside the circle. The dots without blue estimates the area inside the circle.

2.1 Maths

Here, A_c is the area (A) of the circle (c), and A_s is the area of the square (s).

$A_c = \pi r^2$
$A_s = (2r)^2$

$\frac{A_c}{A_s} = \frac{\pi r^2}{(2r)^2} => \frac{\pi r^2}{2^2 r^2} => \frac{\pi r^2}{4r^2} => \frac{\pi}{4}\frac{r^2}{r^2} => \frac{\pi}{4}$
$\frac{\pi}{4} = \frac{A_c}{A_s}$

$\pi = \frac{4\,A_c}{A_s}$

This gives us that we can get an estimate of π by finding $4\frac{A_c}{A_s}$.
In other words:

$$4 * \frac{\text{amount of dots landing in the circle}}{\text{amount of dots landing in the circle + amount of dots landing in the square}}.$$

Or simply, 4 times the chance a dot will land in the circle.

2.2 Code

We achieve this by writing the following code in Python 3.5

```
import random
import math
```

Importing "random" to allow for randomly placed dots in my square, and "math" to be able to calculate whether the dots are within the circle or not.

```
runs    = 1000000
cir     = 0
sqr     = 0
```

Here, "runs" is how many times we want the simulation to run. "cir" and "sqr" are counters (starting at 0) for how many times the dots land in the circle or ine the square (respectively).

```
for i in range (runs):
```

Here, we start a loop that runs for the set amount of times.

```
point = [(random.uniform(-1, 1)), (random.uniform(-1, 1))]
```

In the loop, we first create a list representing the 2-dimentional point/dot in the space, and assign random values for the axis-values thusly:
point(a, b)
$a, b \in [-1, 1]$
with the centre at origo $(0, 0)$

```
r = math.sqrt(point[0]**2 + point[1]**2)
```

Next, we find the distance from origo and the dots location, or the radius (polar coordinates). We achieve this by using the Pythagorean Theorem:
$a^2 + b^2 = c^2$
in a triangle, or a coordinate system with normalized axis. Solving for the c, or the hypotenuse, gives us:
$c = \sqrt{a^2 + b^2}$
where a and b is represented in code by point[0] and point[1].

```
if (r > 1):
        sqr += 1
    else:
        cir += 1
```

If the value of r, the distance from origo, is larger than 1, then the dot is outside the circle (and would be represented with a blue circle around it. See image). If not, it is within the circle.

```
pi = 4 * (cir) / (cir + sqr)
error = abs(math.pi - pi)
```

Calculate π using our formula, and calculate the absolute error from π's real value.

```
print ("pi:\t"      + str(pi))
print ("error:\t"   + str(error))
print ("runs:\t"    + str(runs))
```

```
print ("cir:\t"      + str(cir))
print ("sqr:\t"      + str(sqr))
```

Finally, print out the results of the runs:

pi	- Our estimated value of π
error	- Absolute error of our estimation
runs	- How many times the simulation ran for
cir	- How many times the dots landed inside the circle
sqr	- How many times the dots landed outside the circle

The more times the simulation is ran, the better estimation of π you get. When I ran it 1'000'000, it gave me the following output in 1.776 seconds:

```
pi:     3.140484
error:  0.0011086535897932848
runs:   1000000
cir:    785121
sqr:    214879
```