The number π, while being an *irrational* and *transcendental* number is a central number to which much of mathematics and science at large relies on for many calculations.

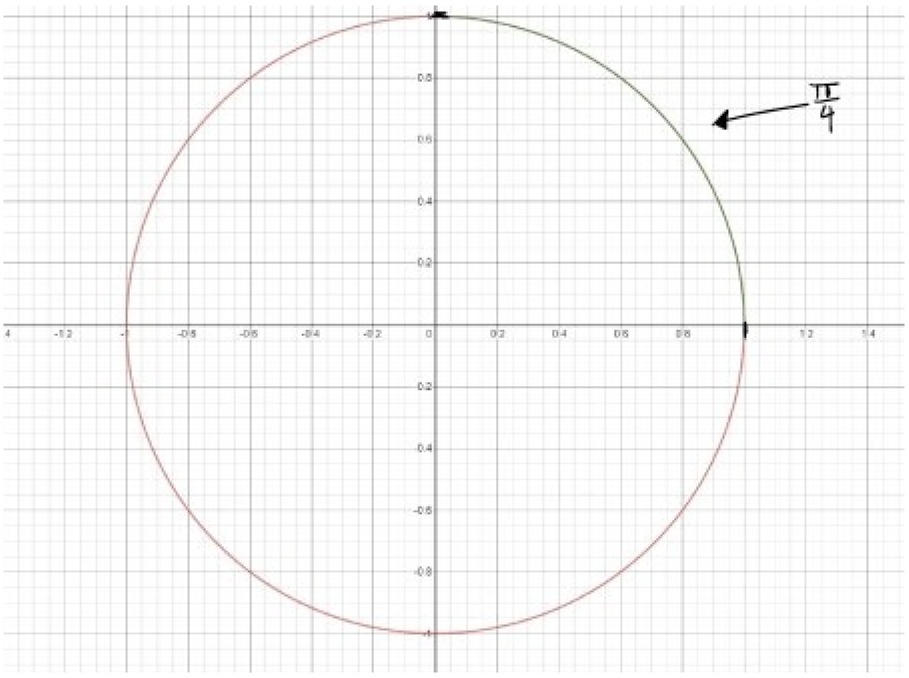
A whole book can be written on this matter alone, but today we are going to focus on approximating the true value of π using Monte Carlo simulations in R and Python!

Disclaimer

This problem has been covered extensively across the internet and serves as a benchmark example of what Monte Carlo can do. What we are going to do is highlight how this method preforms in both R and Python.

The General Algorithm

The formula for the unit circle is:



The Unit Circle; Graphed with Desmos Graphing Calculator

$$x^2 + y^2 = 1$$

$$\iff y = \sqrt{1 - x^2}$$

For $(x, y) \in [0, 1]$, the length of a quarter of the unit circle $\frac{\pi}{4}$. Thus, to approximate π, the function that we will be using is:

$$Y_i = 4\sqrt{1 - X_i^2}$$

To approximate π, we use the following Algorithm:

1. Generate $X_1, X_2 \ldots, X_n \sim U(0, 1)$
2. Calculate $Y_i = 4\sqrt{1 - X_i^2}$
3. Take the mean of $Y_1, \ldots, Y_n$: $\bar{Y} = \frac{\sum_{i=1}^{n} Y_i}{n}$ – this is our approximation

The code for this is relatively straight forward. But the question is, which code will run faster?

Let’s go!

The Test

For our challenge we are going to be writing code which is as intuitive as possible in each language. We are going to seek to approximate the value of πi using the above algorithm. For R we will be using `lapply` to implement the Monte Carlo algorithm and for Python we will using `for` loops.

The Solution with R

```
#' Define Number of points we want to estimate
n<-c(10,100,1000,10000,100000,1000000)

#' Generate our random uniform variables
x<-sapply(n,runif)

#' Our Transformation function

y<- function(u) {
  4*sqrt(1-u^2)
}



startTime<-Sys.time()
yvals<-lapply(x,y)
endTime<-Sys.time()-startTime
avgs<-lapply(yvals,mean)

endTime
```

```
## Time difference of 0.01399588585 secs
```

```
data.frame(n, "MC Estimate"=unlist(avgs), "Difference from True Pi"=
abs(unlist(avgs)-pi))
```

```
##         n MC.Estimate Difference.from.True.Pi
## 1      10 3.281637132         0.1400444782036
## 2     100 3.391190973         0.2495983193740
## 3    1000 3.090265904         0.0513267494211
## 4   10000 3.143465663         0.0018730098616
## 5  100000 3.141027069         0.0005655842822
## 6 1000000 3.141768899         0.0001762457079
```

The Solution with Python

```
import numpy as np
import pandas as pd
import time
```

```
# Define Number of points we want to estimate

n = [10, 100, 1000, 10000, 100000, 1000000]

# Generate our random uniform variables

x = [np.random.uniform(size=n) for n in n]


# Our Transformation function

def y(x):
    return 4 * np.sqrt(1 - x ** 2)


startTime= time.time()
yvals = []
for array in x:
    yval=[]
    for i in array:
        yval.append(y(i))
    yvals.append(yval)

avgs=[]

for array in yvals:
  avgs.append(np.mean(array))

endTime= time.time()-startTime

# How long it took to run our code
print("Time difference of "+ str(endTime) + " secs\n")


# Output


## Time difference of 3.146182060241699 secs
## Estimated Values of Pi



pd.DataFrame({"n":n,
              "MC Estimate":avgs,
              "Difference from True Pi": [np.abs(avg-np.pi) for avg in
avgs]})


##          n  MC Estimate  Difference from True Pi
## 0       10     3.320525                 0.178933
## 1      100     3.172290                 0.030698
## 2     1000     3.156044                 0.014451
## 3    10000     3.141675                          0.000083
## 4   100000     3.147255                          0.005662
## 5  1000000     3.141400                          0.000193
```

Comparing R with Python

From the following ratio we can see how much faster R is than Python:

```
library(reticulate)

reticulate::py$endTime/as.numeric(endTime)
```

```
## [1] 224.7933496
```

Woah! Using my machine- **R is over 220 times faster than Python!**

I think it’s pretty clear to see who the winner is as far as speed is concerned.

Concluding Remarks

While R most of the time sits on the sidelines in the Python-dominant world of Data Science- we need to keep in mind where Python’s weaknesses lie and when to pivot from and use R.