# Using for loops in R

I will be changing the method I generated the data by using R’s for loop and will start timing the code from there.

I did my best to make my R code as similar to the Python code in the last blog- if you see an issue, please comment!

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

#' Our Transformation function

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

}

#' Start the timer startTime<-Sys.time()

#' Generate our random uniform variables x<-list()

for(i in 1:length(n)){ x[[i]]<-runif(n[i])

}

#' Transform our uniform variables. yvals<-list()

for (i in 1:length(x)){

yvals[[i]]<-1

for(j in 1:length(x[[i]])){ yvals[[i]][j]<-y(x[[i]][j])

}

}

#' Calculate our approximations of pi avgs<- c()

for(i in 1:length(yvals)){ avgs[i]<-mean(yvals[[i]])

}

endTime<-Sys.time()-startTime endTime

## Time difference of 1.009413958 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 |

# Using for loops in Python (From previous blog)

As I did in the previous blog, here is the code I used to run the Monte Carlo algorithm with for loops. I heard there are more accurate ways to time this code, but since I want it to be similar to my R code- I am doing it this way.

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]

# Our Transformation function

def y(x):

return 4 \* np.sqrt(1 - x \*\* 2)

#Start the timer startTime= time.time()

# Generate our random uniform variables

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

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 2.790393352508545 secs

print("Estimated Values of Pi\n")

## 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 |  | 3.259405 |  | 0.117812 |
| ## | 1 100 |  | 3.351556 |  | 0.209963 |
| ## | 2 1000 |  | 3.130583 |  | 0.011009 |
| ## | 3 10000 |  | 3.126542 |  | 0.015050 |
| ## | 4 100000 |  | 3.144484 |  | 0.002891 |
| ## | 5 1000000 |  | 3.140740 |  | 0.000853 |

library(reticulate) py$endTime/as.numeric(endTime)

## [1] 2.764369693

Ok- so using for loops R isn’t *as fast* as I initally stated. However, based on my machine **R is still over twice as fast as Python with for loops**.

Hey, it ain’t 220 but its something 

# Using R’s “Best Practices” (Using the apply

**family)**

Instead of using for loops, a faster alternative is to use the apply family of functions, namely

sapply and lapply.

#' Start the timer startTime<-Sys.time()

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

yvals<-lapply(x,y)

avgs<-lapply(yvals,mean)

newendTime<-Sys.time()-startTime newendTime

## Time difference of 0.1879060268 secs

#' Speed - for loop vs apply as.numeric(endTime)/as.numeric(newendTime)

## [1] 5.371908366

# Using Python’s best practices

After getting several comments of (constructive) criticism about how the comparison was not fair here’s some new code implementing some of the best practices in writing faster code.

I’m sure there are better ways out there (I have seen in the comments for the last blog a lot of very good solutions), but I found this to be the most readable and follows a structure similar to R’s.

(Let me know if you have something better!)

startTime= time.time()

x = [np.random.uniform(size=n) for n in n] yvals = list(map(y, x))

avgs = list(map(np.mean, yvals))

endTime= time.time()-startTime

# How long it took to run our code

print("Time difference of "+ str(endTime) + " secs\n")

## Time difference of 0.0629582405090332 secs Comparing R with Python now we have: as.numeric(newendTime)/py$endTime

## [1] 2.984613695

**Python is nearly 3 times faster on my machine** using the updated code.

# Conclusion

Well, you live and learn. Best practices can make it or break it for your code and this updated analysis can help give you a better idea.