Statistic Programming

STATS 202A: Homework #1

Zachary Lacey

October 6, 2021

Problem 1

Problem 1 Prompt

Assessing estimates of the 90th percentile of 100 independent, identically-distributed (iid) uniform(0, 1) random variables. The R function *quantile()* implements a somewhat complex interpolation method in order to estimate a particular quantile, such as the 90th percentile. We will compare the estimate in *quantile()* with simpler estimates.

(A) Write a function that takes as input a vector of length 100 and outputs the 90th of the 100 values sorted from smallest to largest. Note that the input vector might not be sorted.

(B) Write a function to find the 91st of the sorted vector of 100 values.

(C) Write a function that outputs the average of the 90th and 91st of the sorted vector of 100 values.

(D) For each of your functions in parts a-c, as well as the function *quantile(x, 0.9)*, do the following:

  (a) Generate 100 independent, identically-distributed (iid) uniform (0, 1) random variables, and calculate your estimate of the 90th percentile.

  (b) Repeat step (a) 100,000 times

  (c) Plot the sample mean of the first m of your estimates, as a function of m.

(E) Report the ultimate sample mean of your 100,000 estimates, for each of the four estimates. In 1-2 sentences, indicate which of the four estimates appears to be the best, and why.

Problem 1 Output

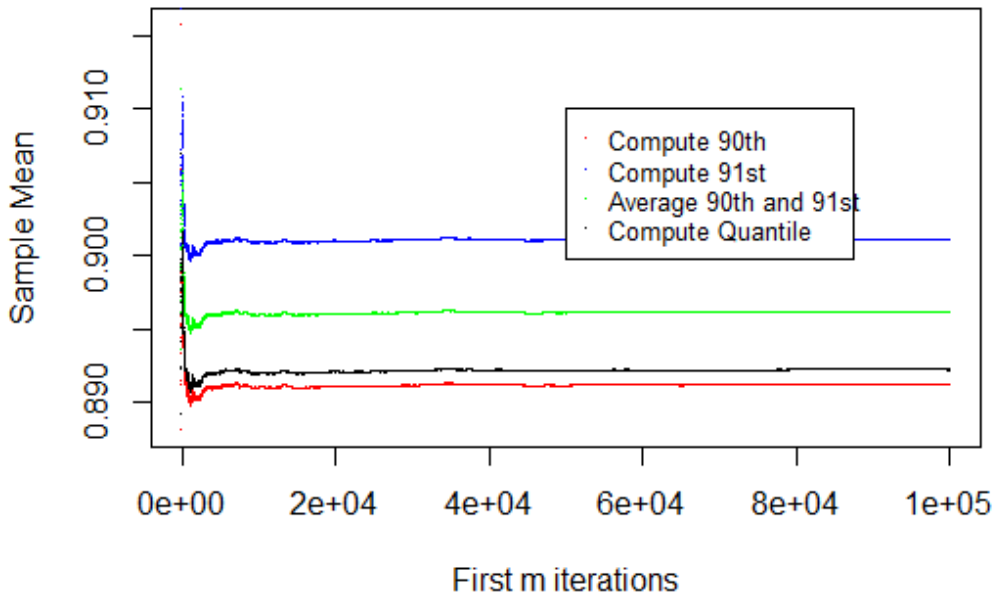

Figure 1: Sample Mean vs First m Iterations.

Results for ultimate sample mean of the 100,000 iterations:

1. Computed 90th of the Sorted 100 values (red): $SampleMean = 0.891185421675972$
2. Computed 91st of the Sorted 100 values (blue): $SampleMean = 0.901058744117816$
3. Computed the Average of the 90th and 91st (green): $SampleMean = 0.896122082896894$
4. Computed 90th from Quantile (black): $SampleMean = 0.892172753920164$

It seems as if the estimation from **Computing the 91st of the 100 values (blue)** yields the best estimate of the 90th percentile as we can see that over 100,000 repeated iterations, it converges the closest to the 90th percentile value (i.e., ultimate sample mean is the closest to 0.90).

Problem 2

Problem 2 Prompt

Functions to approximate π.

(A) Write a function called *pi2(n)* that approximates π as a function of n, using the approximation:

$$\pi = \lim_{n\to\infty} \sqrt{6\sum_{k=1}^{n} k^2}$$

Evaluate *pi2(*10^j*)* for $j = 0, 1, 2, \ldots, 6$.

(B) Write a function *pi3(n)* that approximates π as a function of n, by simulating random points in the square vertices $(-1, -1)$, $(-1, 1)$, $(1, 1)$, and $(1, -1)$, seeing what fraction of them are in the unit circle [the circle with radius 1 centered at the origin], and then converting this fraction into an estimate of π.

Evaluate *pi2(*10^j*)* for $j = 0, 1, 2, \ldots, 6$.

For $j = 6$, plot your simulated points, using different plotting symbols for the simulated points inside and outside the unit circle.

Problem 2 Output

Results for evaluating *pi2(*10^j*)* for $j = 0, 1, 2, \ldots, 6$:

1. *pi2(*10^0*)* = 2.44948974278318
2. *pi2(*10^1*)* = 3.04936163598207
3. *pi2(*10^2*)* = 3.13207653180911
4. *pi2(*10^3*)* = 3.14063805620599
5. *pi2(*10^4*)* = 3.14149716394721
6. *pi2(*10^5*)* = 3.14158310432644
7. *pi2(*10^6*)* = 3.14159169866047

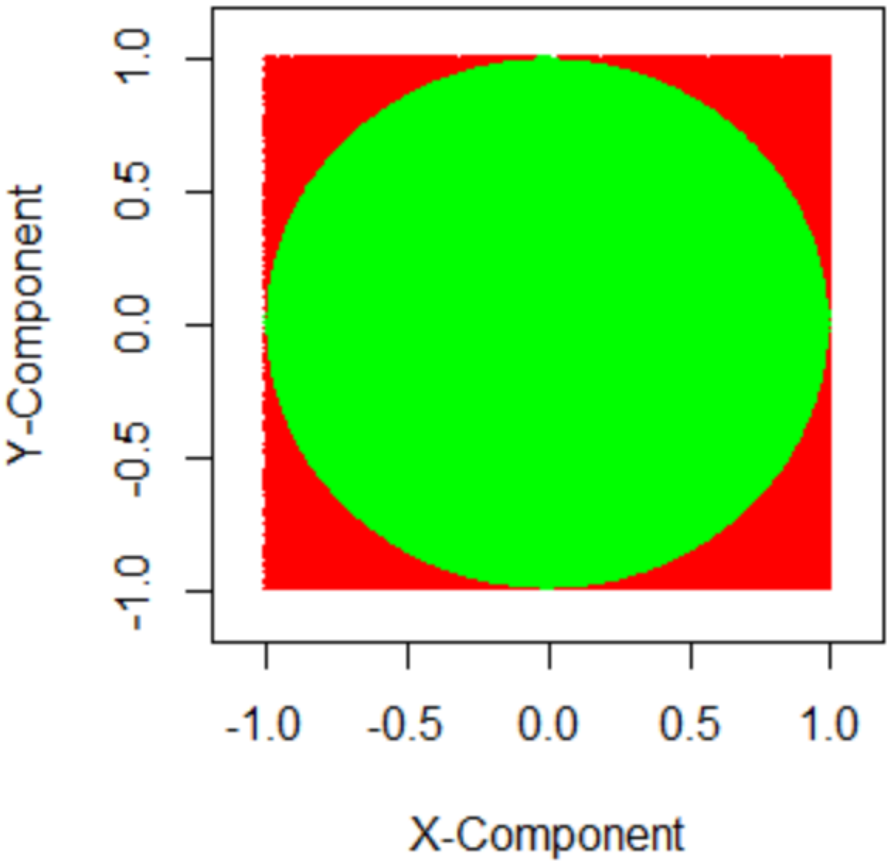

Figure 2: Plot of Simulated Random Points: points within the unit circle (green), points outside the unit circle (red).

To determine an estimated value π, we approximate the fraction of n simulated points in the square vertices ($(-1,-1)$, $(-1,1)$, $(1,1)$, and $(1,-1)$) that fall within the unit circle. This is essentially simulating the estimation of the fraction of the area that the unit circle covers over the square vertices region. This can be approximated as follows:

$$fraction = \frac{area\ unit\ circle}{area\ square\ vertices\ region} = \frac{\#\ of\ points\ in\ unit\ circle}{n}$$

where n is the total number of simulated random points generated in the square vertices ($(-1,-1)$, $(-1,1)$, $(1,1)$, and $(1,-1)$) region.

The fraction relation to the estimate π is obtained as follows:

$$\begin{aligned} fraction &= \frac{\pi r^2}{(2r) * (2r)} \\ &= \frac{\pi r^2}{4r^2} \\ &= \frac{\pi}{4} \end{aligned}$$

so,

$$\pi = 4 * (fraction)$$

Results for evaluating $pi3(10^j)$ for $j = 0, 1, 2, \ldots, 6$:

1. $pi3(10^0) = 4.00$
2. $pi3(10^1) = 3.60$
3. $pi3(10^2) = 3.00$
4. $pi3(10^3) = 3.132$
5. $pi3(10^4) = 3.1624$
6. $pi3(10^5) = 3.14368$
7. $pi3(10^6) = 3.140848$

Code

C++ Code

sample_mean_c.cpp

```cpp
#include <Rcpp.h>
#include <Rmath.h>
using namespace Rcpp;

// [[Rcpp::export]]
DoubleVector sample_mean_c(DoubleVector sample_data_vec, int m) {

  DoubleVector sample_mean(m);
  double sample_sum = 0.0;

  for (int iteration = 0; iteration < m; iteration++) {
    sample_sum += sample_data_vec[iteration];

    sample_mean[iteration] = sample_sum / (iteration + 1);
  }

  return sample_mean;
}
```

Listing 1: sample_mean_c.cpp

R Code

HW1_Question1.R

```r
## Homework 1 Question 1
##
## Author: Zachary G. Lacey
```

```
## Email: zlacey@g.ucla.edu
## Date: September 24th, 2021
##
## Course: STAT 202A - Statistics Programming
## Assignment: Homework 1
## Due Date: October 6th, 2021 (Wednesday), 2:00 PM PST
##
## Question 1
##

library(Rcpp)

## 1. Assessing estimates of the 90th percentile of 100 independent,
##    identically-distributed (iid) uniform (0, 1) random variables.
##
##    The R function quantile() implements a somewhat complex
##    interpolation method in order to estimate a particular quantile,
##    such as the 90th percentile. We will compare the estimate in
##    quantile() with simpler estimates.

## a. Write a function that takes as input a vector of length 100 and
##    outputs the 90th of the 100 values sorted from smallest to largest.
##    Note that the input vector may not be sorted.

## Function that sorts the sample vector and returns the nth element of
## the sorted vector.
computeNSample = function(sampleVector, n) {
  sortedSampleVector = sort(sampleVector)

  sortedSampleVector[n]
}

## Function which returns the 90th element of the sorted vector.
```

```
compute90th = function(sampleVector) {
  computeNSample(sampleVector, 90)
}

## b. Write a function to find the 91st of the sorted vector of 100
##    values.

## Function which returns the 91st element of the sorted vector.
compute91st = function(sampleVector) {
  computeNSample(sampleVector, 91)
}

## c. Write a function that outputs the average of the 90th and 91st of
##    the sorted vector of 100 values.

## Function which returns the average of the 90th and 91st element of the
## sorted vector.
computeAvg90th91st = function(sample90thCurrentCalculation,
                              sample91stCurrentCalculation) {
  (sample90thCurrentCalculation + sample91stCurrentCalculation) / 2
}

## d. For each of your function in part a - c, as well as the function
##    quantile(x, 0.9), do the following:
##
##    (i).   Generate 100 independent, identically-distributed (iid)
##           uniform (0, 1) random variables, and calculate your estimates
##           of the 90th percentile.
##
##    (ii).  Repeat step (i) 100,000 times.
##
##    (iii). Plot the sample mean of the first 'm' of your estimates, as a
##           function of m.
```

```
nIterations = 100000

mIterations = 100000

computedSampleData  = matrix(nrow = 4, ncol = nIterations)

iterations = 1:nIterations

computeSampleN = function(iterationsIdx) {
  sampleIidData = runif(100)

  c1 = compute90th(sampleIidData)
  c2 = compute91st(sampleIidData)
  c3 = computeAvg90th91st(c1, c2)
  c4 = quantile(sampleIidData, 0.9)

  return(c(c1, c2, c3, c4))
}

computedSampleData = sapply(iterations, computeSampleN)

sampleMean90thN = vector("numeric", mIterations)
sampleMean91stN = vector("numeric", mIterations)
sampleMean90th91stN = vector("numeric", mIterations)
sampleMean90thQuantileN = vector("numeric", mIterations)

sourceCpp("sample_mean_c.cpp")

sampleMean90thN = sample_mean_c(
  as.double(computedSampleData[1, ]),
  mIterations)

sampleMean91stN = sample_mean_c(
```

```
  as.double(computedSampleData[2, ]),
  mIterations)

sampleMean90th91stN = sample_mean_c(
  as.double(computedSampleData[3, ]),
  mIterations)

sampleMean90thQuantileN = sample_mean_c(
  as.double(computedSampleData[4, ]),
  mIterations)

plot_sample_mean = function(m_iter) {
  plot(1:m_iter, sampleMean90thN, pch=".", col="red",
       xlab="First m iterations", ylab="Sample Mean")

  points(1:m_iter, sampleMean91stN, pch=".", col="blue")
  points(1:m_iter, sampleMean90th91stN, pch=".", col="green")
  points(1:m_iter, sampleMean90thQuantileN, pch=".")
}

plot_sample_mean(mIterations)

## e. Report the ultimate sample mean of your 100,000 estimates, for each
##    of the four estimates. In 1-2 sentences, indicate which of the 4
##    estimates appears to be the best, and why.

print(c("Computed 90th of Sorted:",
        sampleMean90thN[nIterations]))
print(c("Computed 91st of Sorted:",
        sampleMean91stN[nIterations]))
print(c("Computed Average 90th and 91st:",
        sampleMean90th91stN[nIterations]))
print(c("Computed 90th from Quantile:",
```

```
        sampleMean90thQuantileN[nIterations]))

print("Finished")
```

Listing 2: HW1_Question1.R

HW1_Question2.R

```
## Homework 1 Question 2
##
## Author: Zachary G. Lacey
## Email: zlacey@g.ucla.edu
## Date: September 24th, 2021
##
## Course: STAT 202A - Statistics Programming
## Assignment: Homework 1
## Due Date: October 6th, 2021 (Wednesday), 2:00 PM PST
##
## Question 2
##

## 2. Functions to approximate pi.

## a. Write a function called pi2(n) that approximates pi as a function
##    of 'n', using the approximation:
##
##    pi2(n) = lim sqrt(6 * summation (k^-2))
##
##    Evaluate pi2(10^j) for j = 0, 1, 2, ..., 6.

pi2 = function(n) {
  summation = rep(1, n)

```

```
  for (k in 2:n) {
    summation[k] = 1 / k^2
  }

  cumulativeSummation = (6 * cumsum(summation))
  pi2Estimates = sqrt(cumulativeSummation)

  pi2Estimates[n]
}

# Start the clock!
ptm = proc.time()

pi2_0 = pi2(10^0)
pi2_1 = pi2(10^1)
pi2_2 = pi2(10^2)
pi2_3 = pi2(10^3)
pi2_4 = pi2(10^4)
pi2_5 = pi2(10^5)
pi2_6 = pi2(10^6)

# Stop the clock
proc.time() - ptm

## b. Write a function pi3(n) that approximates pi as a function of 'n',
##    by simulating random points in the square with vertices (-1, 1),
##    (-1, 1), (1, 1), and (1, -1), seeing what fraction of them are in
##    the unit circle [the circle with radius 1 centered at the origin],
##    and then convert the fraction into an estimate of pi. Evaluate
##    pi3(10^j) for j = 0, 1, 2, ..., 6. For j = 6, plot your simulated
##    points, using different plotting symbols for simulated points
##    inside and outside the unit circle. There is no need for you to
##    plot the unit circle also.
```

```
59
60 pi3 = function(n, plotBool) {
61   x = runif(n) * 2 - 1
62   y = runif(n) * 2 - 1
63
64   isInUnitCircle = (x^2 + y^2 <= 1)
65
66   if (plotBool) {
67     plot(c(-1.1, 1.1), c(-1.1, 1.1), type = "n",
68          xlab = "X-Component", ylab = "Y-Component")
69
70     points(x[which(isInUnitCircle)],
71            y[which(isInUnitCircle)],
72            cex = 0.8,
73            pch=".", col="green")
74     points(x[which(!isInUnitCircle)],
75            y[which(!isInUnitCircle)],
76            cex = 0.8,
77            pch=".", col="red")
78   }
79
80   pi_estimate = 4 * sum(isInUnitCircle) / n
81 }
82 pi3_0 = pi3(10^0, FALSE)
83 pi3_1 = pi3(10^1, FALSE)
84 pi3_2 = pi3(10^2, FALSE)
85 pi3_3 = pi3(10^3, FALSE)
86 pi3_4 = pi3(10^4, FALSE)
87 pi3_5 = pi3(10^5, FALSE)
88 pi3_6 = pi3(10^6, TRUE)
89
90 print("Finished")
```

Listing 3: HW1_Question2.R