Stats506: hw2_3

David Li

October 22, 2017

Stats506: Problem 3 Code
Data Used:
Submitted by: David Li

```r
# Some initialization of parameters
sample_iterations = 1000 # 1000 Sample Iterations
points = data.frame(x=numeric(sample_iterations), y=numeric(sample_iterations))
sample_estimate_pi = numeric(sample_iterations) # Stores estimates of pi from the iterations
ci_estimate = data.frame(lowerbound = numeric(sample_iterations), upperbound = numeric(sample_iteration
error = c(numeric(sample_iterations))
proportion_within = c(numeric(sample_iterations))
within_circle = 0 # Start a counter of inner circle points
outof_circle = 0 # Start a counter of outer circle points
multiplier = qnorm(0.975)

## Perform Simulation: Square with X and Y between -1 and 1
for(i in 1:sample_iterations)
{
  ## Pick a Random point
  points$x[i] <- runif(1, min=-1, max=1)
  points$y[i] <- runif(1, min=-1, max=1)

  ## Classify if it is within the unit circle
  if((points$x[i])^2 + (points$y[i])^2 <= 1)
  {
    within_circle = within_circle +1
  }
  else
  {
    outof_circle = outof_circle +1
  }

  ## Attempt to simulate pi, iteration times
  sample_estimate_pi[i] <- (within_circle/(within_circle + outof_circle)) / (0.25)
  proportion_within[i] = within_circle / sample_iterations;
  error[i] = multiplier * sqrt((proportion_within[i] * (1-proportion_within[i])) / sample_iterations) #
  prop_inside_CI = c(sample_estimate_pi[i] - error[i], sample_estimate_pi[i] + error[i])
  pi_estimate_CI = 4 * prop_inside_CI # CI for pi estimate
  ci_estimate[i,1] = prop_inside_CI[1]
  ci_estimate[i,2] = prop_inside_CI[2]
}
compareresult = matrix(data = c(proportion_within,sample_estimate_pi), ncol = 2)
head(compareresult)
```

```
##       [,1]     [,2]
## [1,] 0.001 4.000000
## [2,] 0.002 4.000000
## [3,] 0.002 2.666667
## [4,] 0.003 3.000000
## [5,] 0.004 3.200000
## [6,] 0.004 2.666667
```

```
tail(compareresult)
```

```
##          [,1]     [,2]
##  [995,] 0.793 3.187940
##  [996,] 0.794 3.188755
##  [997,] 0.795 3.189569
##  [998,] 0.796 3.190381
##  [999,] 0.797 3.191191
## [1000,] 0.797 3.188000
```

The constant we are estimating is pi, namely 3.1415. From this simulation, low sample iterations cause an inaccurate estimate but higher sample iterations
a.k.a. picking more points leads to a better estimation of pi.

```
accuracy_n = function(n, conflevel = 0.995){ # Doing a new simulation to determine sample size needed f
  # Setting up question similar to before
  x1 = runif(n, min = -1, max = 1)
  x2 = runif(n, min = -1, max = 1)
  new1 = (x1)^2 + (x2)^2
  new2 = sapply(new1, circlereturn)
  proportion = sum(new2) / n
  variance = (16/n^2) * var(new2) # General Variance Relationship
  lowerbound = 4*proportion - (qnorm(conflevel) * sqrt(variance))
  upperbound = 4*proportion + (qnorm(conflevel) * sqrt(variance))
  return(data.frame("n ="= n, "prop ="= proportion, "mean="= 4*proportion, "standarddev ="= sqrt(varianc
}
multiple_n = c(10, 50, 100, 250, 1000, 5000, 10000, 1e5)
t(sapply(multiple_n, accuracy_n))
```

```
##      n..   prop..  mean.   standarddev.. CI_Lower.. CI_Upper..
## [1,] 10    0.9     3.6     0.1264911     3.274181   3.925819
## [2,] 50    0.74    2.96    0.035447      2.868695   3.051305
## [3,] 100   0.78    3.12    0.01665333    3.077104   3.162896
## [4,] 250   0.8     3.2     0.006412839   3.183482   3.216518
## [5,] 1000  0.809   3.236   0.001573143   3.231948   3.240052
## [6,] 5000  0.7872  3.1488  0.0003274627  3.147957   3.149643
## [7,] 10000 0.7813  3.1252  0.0001653541  3.124774   3.125626
## [8,] 1e+05 0.78583 3.14332 1.640989e-05  3.143278   3.143362
```

Notice as n gets large, the accuracy of the estimate and the CI increases. When n is small, the accuracy is very low and usually the estimate and CI are not
good measures or indicators of the true value of pi.
n was chosen from small to large numbers, with large numbers hopefully supporting a better accuracy of estimator.
Generally, around my sample size of 100000 shows decent accuracy to two decimal places.

```
# Now same simulation, but limiting bounds to min = 0 and max = 1 representing new square

# Some initialization of parameters
points2 = data.frame(x=numeric(sample_iterations), y=numeric(sample_iterations))
sample_estimate_pi2 = numeric(sample_iterations) # Stores estimates of pi from the iterations
ci_estimate2 = data.frame(lowerbound = numeric(sample_iterations), upperbound = numeric(sample_iteratio
error2 = c(numeric(sample_iterations))
```

```
proportion_within2 = c(numeric(sample_iterations))
within_circle = 0 # Start a counter of inner circle points
outof_circle = 0 # Start a counter of outer circle points
multiplier = qnorm(0.975)


## Perform Simulation: Square with X and Y between 0 and 1

for(i in 1:sample_iterations)
{
  ## Pick a Random point
  points2$x[i] <- runif(1, min=-1, max=1)
  points2$y[i] <- runif(1, min=-1, max=1)

  ## Classify if it is within the unit circle
  if( (points2$x[i])^2 + (points2$y[i])^2 <= 1)
  {
    within_circle = within_circle +1
  }
  else
  {
    outof_circle = outof_circle +1
  }

  ## Attempt to simulate pi, iteration times
  sample_estimate_pi2[i] <- (within_circle/(within_circle + outof_circle)) / (0.25)
  proportion_within2[i] = within_circle / sample_iterations;
  error2[i] = multiplier * sqrt((proportion_within2[i] * (1-proportion_within2[i])) / sample_iterations
  prop_inside_CI2 = c(sample_estimate_pi2[i] - error2[i], sample_estimate_pi2[i] + error2[i])
  pi_estimate_CI2 = 4 * prop_inside_CI2 # CI for pi estimate
  ci_estimate2[i,1] = prop_inside_CI2[1]
  ci_estimate2[i,2] = prop_inside_CI2[2]
}

compareresult2 = matrix(data = c(proportion_within2, sample_estimate_pi2), ncol = 2)
head(compareresult2)
```

```
##       [,1]     [,2]
## [1,] 0.001 4.000000
## [2,] 0.002 4.000000
## [3,] 0.002 2.666667
## [4,] 0.003 3.000000
## [5,] 0.004 3.200000
## [6,] 0.005 3.333333
```

```
tail(compareresult2)
```

```
##          [,1]     [,2]
##  [995,] 0.780 3.135678
##  [996,] 0.781 3.136546
##  [997,] 0.782 3.137412
##  [998,] 0.783 3.138277
##  [999,] 0.784 3.139139
## [1000,] 0.785 3.140000
```

```
accuracy_n2 = function(n, conflevel = 0.995){ # Doing a new simulation to determine sample size needed
  # Setting up question similar to before
  x1 = runif(n, min = 0, max = 1)
  x2 = runif(n, min = 0, max = 1)
  new1 = (x1)^2 + (x2)^2
  new2 = sapply(new1, circlereturn)
  proportion = sum(new2) / n
  variance = 16 * (16/n^2) * var(new2) # General Variance Relationship, variance property states that n
  lowerbound = 4*proportion - (qnorm(conflevel) * sqrt(variance))
  upperbound = 4*proportion + (qnorm(conflevel) * sqrt(variance))
  return(data.frame("n ="= n, "prop ="= proportion, "mean="= 4*proportion, "standarddev ="= sqrt(varian
}
t(sapply(multiple_n, accuracy_n2))
```

```
##      n..   prop..  mean.   standarddev.. CI_Lower.. CI_Upper..
## [1,] 10    0.5     2       0.843274      -0.17213   4.17213
## [2,] 50    0.76    3.04    0.1380541     2.684396   3.395604
## [3,] 100   0.78    3.12    0.06661331    2.948415   3.291585
## [4,] 250   0.78    3.12    0.02656495    3.051573   3.188427
## [5,] 1000  0.779   3.116   0.006642049   3.098891   3.133109
## [6,] 5000  0.79    3.16    0.001303517   3.156642   3.163358
## [7,] 10000 0.7833  3.1332  0.0006592274  3.131502   3.134898
## [8,] 1e+05 0.78681 3.14724 6.553005e-05  3.147071   3.147409
```

Recreating this table shows that even though we cut the area and placement of the square, the patterns in estimator and CI accuracy are pretty much the same.
We can likely attribute this as since we are calculating proportions, the area of the square is proportional to the proportion of points that will fall in the
area. Thus the patterns displayed by the estimator and their accuracies will not change by much, if at all.
In regard to variance, if you compare the standard deviations in the two tables between the two simulations you will see that when the
area was cut into 1/4, the standard deviation increased by a factor of 4. This supports the variance property that var(a*x) = a^2 times var(x), where a is a constant.

```
## Appendix ##
## Packages Utilized ##
library("dplyr", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("ggplot2", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("knitr", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("rmarkdown", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("stringdist", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("stringr", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("tidyr", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")

# Begin Functions Section #
circlereturn = function(b){ # Returns a binary version of whether a point is in the circle or not
  if(b <= 1){
    return(1)
  }
  else{
    return(0)
  }
}
# End Functions Section #
```