W203 Week 3 Supplementary Exercise 1

Mohammad Jawad Habib

January 29, 2016

Question 1: Sieve of Erastothenes

This implementation of Sieve of Erastothenes will make use of R's capability to automatically apply operations to all members in a vector.

Our starting point will be a vector that includes 2 and odd numbers up to our target number n. For example, for $n = 10$ our starting vector will be $(2, 3, 5, 7, 9)$.

We will then construct a vector that includes all numbers of the form $p^2 + 2p$ up to the biggest number in our starting vector. Here p starts with the first elment of our starting vector, and we continue to iterate while p^2 is smaller than the biggest number in our starting vector.

As we iterate over values of p in our starting vector, we will replace the starting vector with a vector that includes items that are not in both starting vector and the vector generated in each iteration over members of starting vector.

Note that we only need to test numbers up to and including $sqrt(n)$. Assume that $n = a * b$ where a and b are prime numbers. Assume that $a > sqrt(n)$ and $b > sqrt(n)$. That would mean that $a * b > n$. But that contradicts our first assumption $n = a * b$. Therefore, either $a <= sqrt(n)$ or $b <= sqrt(n)$. So if n were not a prime number, we only need to test factors up and including to $sqrt(n)$.

```
prime.sieve <- function(n){
  # We know there are no primes below 1
  if(n <= 1){
    return(paste0('No prime numbers exist <= ', n))
  }
  # We know that 2 is the only prime <= 2
  else if(n==2){
    return(n)
  }
  # For n > 2
  else{
    # Generate a vector that includes 2 and odds up to n
    primes <- c(2, seq.int(3, n, 2))

    i <- 1
    # While p^2 <= n
    while(primes[i]**2 <= primes[length(primes)]){
      # Remove numbers of the form p^2 + 2p from primes
      primes <- setdiff(primes,
                        seq(primes[i]**2,
                            primes[length(primes)],
                            2*primes[i]))
            i <- i+1 # Go to next index
    }
    return(primes)
  }
}
```

Below we can test our prime.sieve function for some sample n.

```
prime.sieve(-1)
```

```
## [1] "No prime numbers exist <= -1"
```

```
prime.sieve(0)
```

```
## [1] "No prime numbers exist <= 0"
```

```
prime.sieve(2)
```

```
## [1] 2
```

```
prime.sieve(3)
```

```
## [1] 2 3
```

```
prime.sieve(100)
```

```
##  [1]  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
## [24] 89 97
```

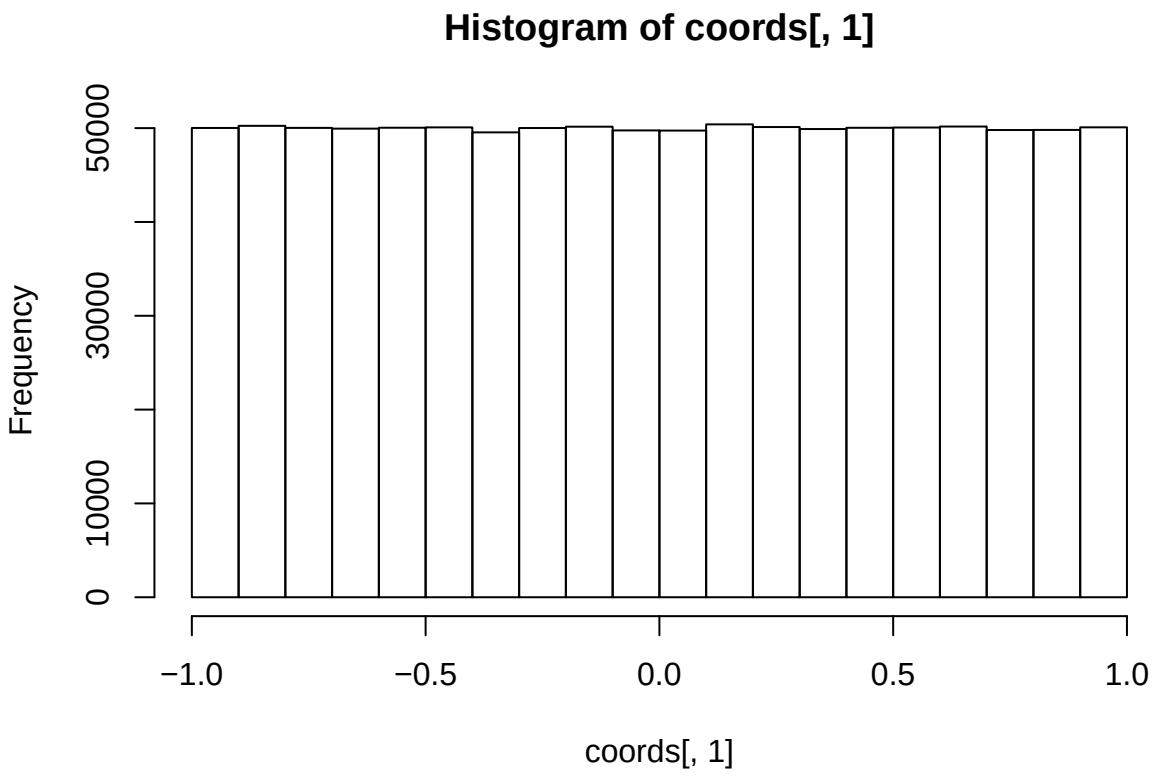
Question 2: Monte Carlo Estimation of Pi

In this question we are asked to estimate Pi using Monte Carlo simulation. We will generate radom x and y coordinates using `runif` and store them in a two column matrix.

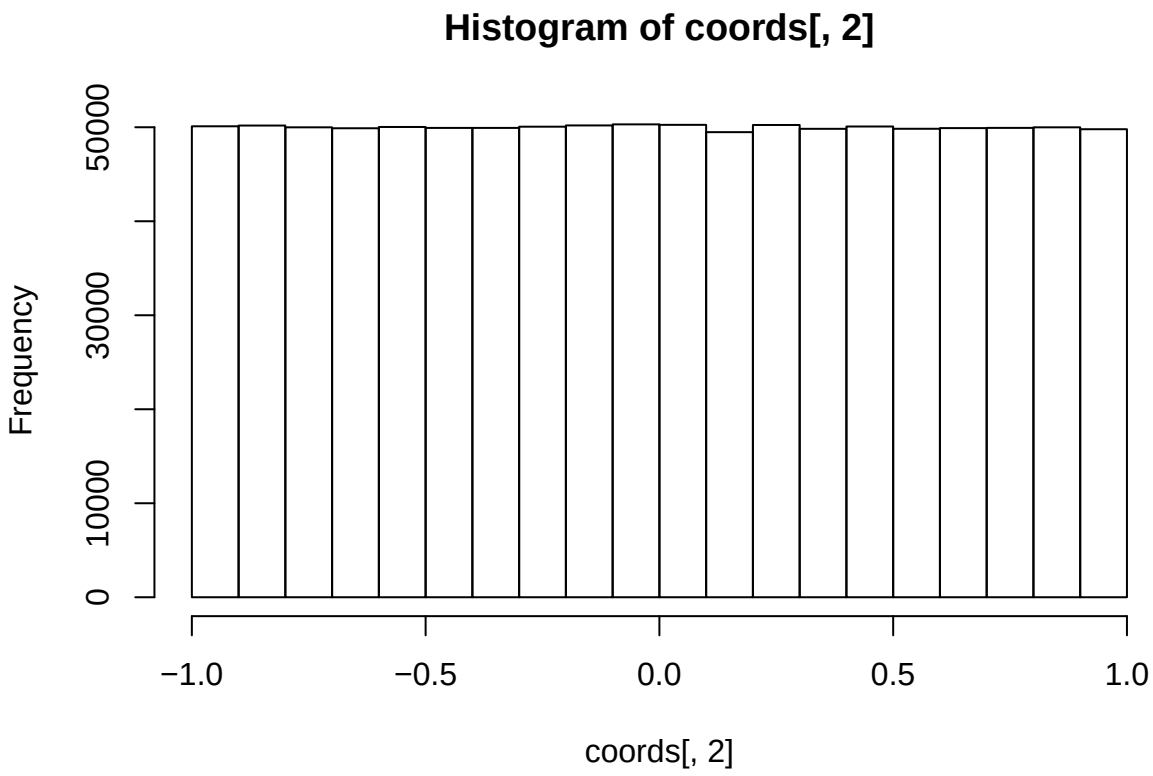
We will write a function that tells us whether a given coordinate is inside a circle of unit radius or not.

We will then calculate the probability of a random coordinate falling inside the circle.

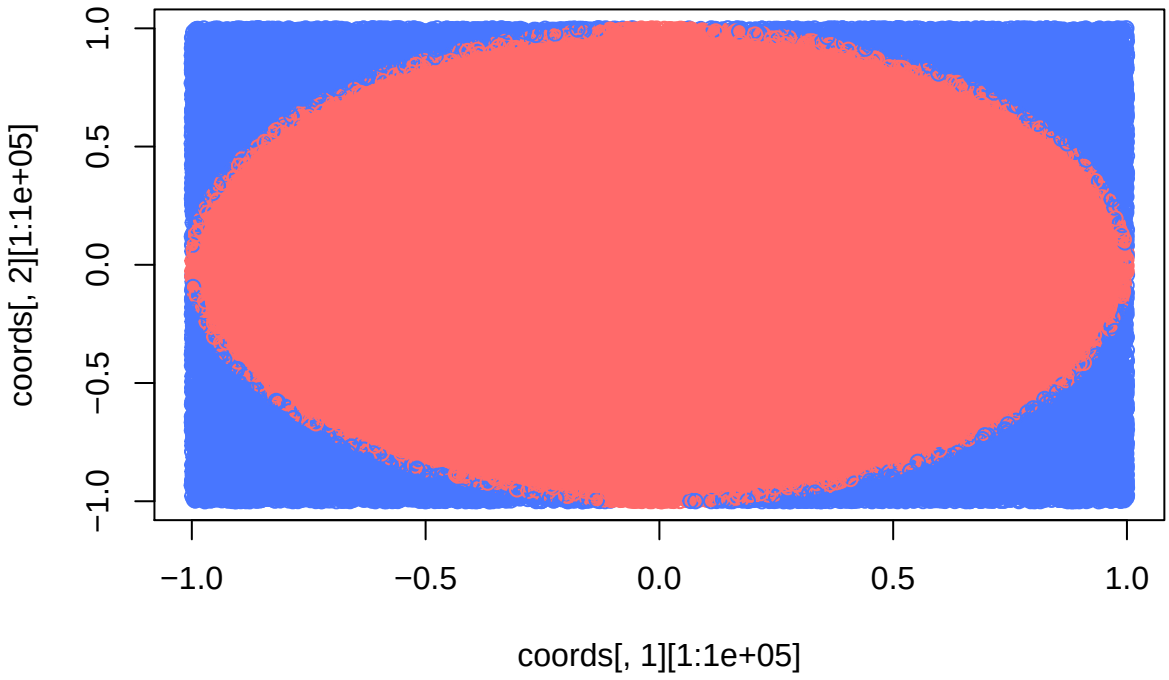
```
# A function that determines whether (x, y) is inside a circle of radius 1
# Returns 1 if true, 0 if false
is.inside.circle <- function(vec){
  ifelse(vec[1]**2 + vec[2]**2 <= 1, 1, 0)
}

# Let number of coordinates equal 1000000
n <- 10**6

# Generate n coordinates using runif as an [x, y] matrix
coords <- cbind(runif(n, -1, 1), runif(n, -1, 1))

# Plot histograms of our random x and y coordinates
hist(coords[,1])
```




```
hist(coords[,2])
```




```r
# Create a plot of our points, red are inside circle, blue are outside
plot(coords[,1][1:100000], coords[,2][1:100000], type="p",
     col = ifelse(coords[,1]**2 + coords[,2]**2 <= 1, "indianred1", "royalblue1"))
```




```
# Pass coords to is.inside.circle and get 1 or 0 for each
# Store this result in a vector
test.vector <- apply(coords, 1, FUN='is.inside.circle')

# Sum of test.vector gives us the number of points inside the circle
# Length gives the total number which is equal to n = 10**6
# Sum divided by Length gives us the probability that a random coordinate falls inside the circle
# We multiply that by the area of a square with side=1 to get estimate of pi
pi.estimate <- sum(test.vector)/length(test.vector)*4

print(pi.estimate)
```

```
## [1] 3.142664
```

We are told that the theoretical variance of a bernoulli random variable is given by $p * (1 - p)$. Here our random variable "r" tells us whether a randomly selected coordiante was inside the circle (1) or outside (0). Therefore, the probability that "r" is inside the circle is given by `sum(test.vector)/length(test.vector)`.

```
# Calculate the theoretical variance of pi.estimate
# Let r be the result of is.inside.circle(coord)
r.expectation = (pi.estimate/4)*(1) + (pi.estimate/4)*(0)
r_squared.expectation = (pi.estimate/4)*(1)**2 + (pi.estimate/4)*(0)**2

#variance = r_squared.expectation - (r.expectation)**2
variance.theoretical <- (pi.estimate/4)*(1-pi.estimate/4)
```

```r
print(variance.theoretical)
```

```
## [1] 0.1683949
```

```r
# Calculate theoretical variance of pi.estimate
variance.theoretical.pi <- variance.theoretical*16

print(variance.theoretical.pi)
```

```
## [1] 2.694319
```

Next we estimate the variance of r by running the simulation over n trials.

```r
# Function for estimating pi n times
# where sample size for each estimate and number of estimates are given
pi.estimates.n <- function(sample.size, trials){
  pi.estimates.array <- c()
  i <- 1
  while(i <= trials){
    i <- i + 1
    test.vector <- apply(cbind(runif(sample.size, -1, 1), runif(sample.size, -1, 1)),
                         1, FUN='is.inside.circle')
    pi.estimates.array <- rbind(pi.estimates.array,
                                sum(test.vector)/length(test.vector)*4)
  }
  return(pi.estimates.array)
}

# Numerically calculate the variance of pi.estimates
# where sample size = 100000 and number of trials = 100

pi.estimates <- pi.estimates.n(100000, 10)

print(head(pi.estimates))
```

```
##         [,1]
## [1,] 3.13776
## [2,] 3.14452
## [3,] 3.14224
## [4,] 3.13704
## [5,] 3.14580
## [6,] 3.13276
```

```r
# Calculate the variance in the probability of a random point failling inside the circle
variance.numerical <- mean((mean(pi.estimates/4)-(pi.estimates/4))^2)

print(variance.numerical)
```

```
## [1] 1.207576e-06
```

```r
# Calculate the variance in our estimates of Pi
variance.numerical.pi <- variance.numerical*16

print(variance.numerical.pi)
```

```
## [1] 1.932122e-05
```

We can see from the above that variance decreases as the sample size of trials increases.