DATA 609: Homework 4

Aaron Grzasko

September 15, 2017

```
library(dplyr)
library(knitr)
library(stringr)
```

Page 191: Question 3

Using Monte Carlo simulation, write an algorithm to calculate an approximation to π by considering the number of random points selected inside the quarter circle.

$$Q : x^2 + y^2 = 1, \ x \geq 0, y \geq 0$$

where the quarter circle is taken to be inside the square

$$S : 0 \geq x \geq 1 \ \text{ and } \ 0 \geq y \geq 1$$

Use the equation

$$\frac{\pi}{4} = \frac{\text{area } Q}{\text{area } S}$$

We can simulate x and y values in the first quadrant by assuming both x and y variables are uniformly distributed between 0 and 1, i.e. $U(0, 1)$.

If our simulated pair of values satisfies the inequality $x^2 + y^2 \leq 1$, then we assume the point is inside the quarter circle.

Our estimate of $\frac{\pi}{4}$ is calculated as the number of points simulated in the quarter circle, divided by the total number of simulated points.

We then solve for π by multiplying by 4.

Note: the R language supports vectorized operations; so this problem can be solved without the use of loops.

```
# function to estimate pi using sample size, n
pi_est <- function(n){
    myrand.x <- runif(n) # generate random x values between 0 and 1
    myrand.y <- runif(n) #  gnerate random y values between 0 and 1
    sum(myrand.x^2 + myrand.y^2 <=1) / n * 4
}

# specify various sample sizes, from 10 to 10 million
n <- c(10,100,1000,10000,100000,1000000,10000000)

# apply pi estimate function for specified sample sizes
set.seed(1981)
my_est <- sapply(n, pi_est)
```

```r
# save results in data frame, format and print result
n_text <- formatC(n, format="d",big.mark = ",")
mydf <- data.frame(sample_size = n_text, pi_estimate = my_est, pi = pi,abs_diff = abs(my_est - pi))
kable(mydf, digits = 5, align = "rrrr")
```

sample_size	pi_estimate	pi	abs_diff
10	3.60000	3.14159	0.45841
100	3.36000	3.14159	0.21841
1,000	3.14400	3.14159	0.00241
10,000	3.15280	3.14159	0.01121
100,000	3.15628	3.14159	0.01469
1,000,000	3.14021	3.14159	0.00138
10,000,000	3.14141	3.14159	0.00019

In our results table above, we see that the approximation for π improves as the number of our simulated sample size increases.

Page 194, Question 1

Use the middle-square method to generate the following random number sequences.

Below we construct a function to implement the middle-square method based on seed value and a specified sequence length. Our function is robust enough to handle seed values of varying sizes.

```r
# random number generator function; employs middle square method
# inputs: n: number of random numbers to be generated; x0: seed number of 4 digits or higher
# output: vector of pseudorandom integers
rand_ms <- function(n, x0){

  # initialize with seed value
  x <- x0

  # number of digits required in squared number, assume odd length numbers have leading zero
  num_dig <- (nchar(x) + nchar(x) %% 2) * 2

  # helper function to calculate next number in sequence
  mid_sq <- function(z){

    # square number, add leading zeros if appropriate
    z <- str_sub(paste(c(replicate(num_dig,"0"),z^2), collapse = ""),-num_dig)

    # return middle digits, convert to integer
    as.integer(str_sub(z,start = 1 + num_dig/4 , end = num_dig - num_dig/4))
  }

  # apply repeated iterations of mid_sq() based on previous output; append to vector
  if (n > 1) {
    for (i in 2:n) {
      x <- append(x, mid_sq(x[length(x)]))
    }
  }
```

```
  x
}
```

(A): 10 random numbers using $x_0 = 1009$

```
# 10 random numbers starting with seed 1009
n <- 10
x0 <- 1009
myrand <- rand_ms(n, x0)
myrand
```

```
 [1] 1009  180  324 1049 1004   80   64   40   16    2
```

(b): 20 random numbers using $x_0 = 653217$

```
# 20 random numbers starting with seed 653217
n <- 20
x0 <- 653217
myrand <- rand_ms(n, x0)
myrand
```

```
 [1] 653217 692449 485617 823870 761776 302674 611550 993402 847533 312186
[11] 460098 690169 333248  54229 940784  74534 555317 376970 106380 316704
```

(c): 15 random numbers using $x_0 = 3043$

```
# 15 random numbers starting with seed 3043
n <- 15
x0 <- 3043
myrand <- rand_ms(n, x0)
myrand
```

```
 [1] 3043 2598 7496 1900 6100 2100 4100 8100 6100 2100 4100 8100 6100 2100
[15] 4100
```

(d): Comment about the results of each sequence. Was there cycling? Did each sequence degenerate rapidly?

- Part A: No cycling, but the sequence is degenerating rapidly. After the first ten numbers in the sequence, all values are equal to zero.
- Part B: No obvious cycling or degeneration in the first 20 draws.
- Part C: The sequence begins to cycle at the 9th value in the sequence. At this point, we see the following repeating sequence : 6,100, 2,100, 4,100, 8,100

Page 201, Question 4

***Horse Race* - Construct and perform a Monte Carlo simulation of a horse race. Simulate the Mathematical Derby with the entries and odds shown in the following table**

```
# data
horse <- c("Euler's Folly","Leapin' Leibniz","Newton Lobell","Count Cauchy","Pumped up Poisson",
           "Loping L'Hopital","Steamin' Stokes","Dancin' Dantzig")
payoff <- c(7,5,9,12,4,35,15,4)
stake <- rep(1,8)
odds <- paste0(payoff,"-",stake)
```

```
# print table
mydf <- data.frame(number = 1:8,horse=horse, odds=odds)
kable(mydf)
```

number	horse	odds
1	Euler's Folly	7-1
2	Leapin' Leibniz	5-1
3	Newton Lobell	9-1
4	Count Cauchy	12-1
5	Pumped up Poisson	4-1
6	Loping L'Hopital	35-1
7	Steamin' Stokes	15-1
8	Dancin' Dantzig	4-1

Construct and perform a Monte Carlo simulation of 1000 horse races.

First, we'll calculate the implied probability of each horse winning.

```
# probability of each horse winning
probs <- 1 - payoff / (payoff + stake)
probs
```

```
[1] 0.12500000 0.16666667 0.10000000 0.07692308 0.20000000 0.02777778
[7] 0.06250000 0.20000000
```

The implied probabilities do not sum up to 1:

```
sum(probs)
```

```
[1] 0.9588675
```

Generally speaking, implied probabilities sum up to a value greater than one. The difference between the sum and 1 represents the bookie's edge. In this case, the values sum to a value less than one. This is an advantageous situation for the bettor, and is referred to as a "sure bet".

To obtain an estimate of the true probabilities, we'll divide each implied probability by the sum of implied probabilities:

```
true_probs <- probs / sum(probs)
true_probs
```

```
[1] 0.13036212 0.17381616 0.10428969 0.08022284 0.20857939 0.02896936
[7] 0.06518106 0.20857939
```

Now, our probabilities sum to 1:

```
sum(true_probs)
```

```
[1] 1
```

Let's calculate cumulative probabilities as follows:

```
# vector of true, cumulative probabilities
cumu_prob <- cumsum(true_probs)
cumu_prob
```

```
[1] 0.1303621 0.3041783 0.4084680 0.4886908 0.6972702 0.7262396 0.7914206
[8] 1.0000000
```

We're ready for the simulation. We will make 1,000 random draws from a standard uniform distribution. The value of each draw corresponds to a value from our cumulative distribution, and ultimately maps to one of our eight horses. For example, if our first draw is 0.05, we assign a win to horse one because the value is less than the rightmost value of 0.1304 for horse one. If, on the other hand, we draw a 0.85, we map the win to horse eight, as the random draw is higher than the rightmost value for horse seven, but lower than the rightmost value for horse eight.

```
# simulate 1k random U(0,1) variables
sims <- 1000
set.seed(5678)
myrand <- runif(sims)

# match random numbers with horse cumulative probability
winners <- findInterval(myrand,cumu_prob)+1

# summarize simulation results
sim_results <- data.frame(number = winners) %>%
  group_by(number) %>%
  count() %>%
  rename(wins = n) %>%
  mutate(sim_prob = sprintf("%.2f%%", 100* wins/sims))

# combine sim results with original odds data frame; print
mydf <- mydf %>%
  inner_join(sim_results, by="number")
mydf$true_prob <- sprintf("%.2f%%", 100*true_probs)

kable(mydf)
```

number	horse	odds	wins	sim_prob	true_prob
1	Euler's Folly	7-1	136	13.60%	13.04%
2	Leapin' Leibniz	5-1	186	18.60%	17.38%
3	Newton Lobell	9-1	112	11.20%	10.43%
4	Count Cauchy	12-1	90	9.00%	8.02%
5	Pumped up Poisson	4-1	198	19.80%	20.86%
6	Loping L'Hopital	35-1	25	2.50%	2.90%
7	Steamin' Stokes	15-1	52	5.20%	6.52%
8	Dancin' Dantzig	4-1	201	20.10%	20.86%

Which horse won the most races?

Dancin' Dantzig

Which horse won the the fewest races?

Loping L'Hopital

Do these results surprise you?

No, the simulation results are very much in line with our long run expectations.

Given an infinite number of trials, we expect both Pumped up Poisson and Dancin' Dantzig to tie for the most wins. In our 100 simulations, Dancin' Dantzig won 3 more races than Pumped up Poisson; so their win totals were still very close.

We also expected Loping L'Hopital to win the fewest number of races, and this is what our simulation showed us.

Provide the tallies of how many races each horse won with your output.

See table provided above.