# Probability and Statistics Final Project Report
## Part 1
## Course Code: 201-DDD-05 STATISTICS

Author:
Hoang Anh, Nguyen
*John Abbott College*
*Montreal, CANADA*

**Abstract:**

The objective of this project is to equip the author with the knowledge of using R functional programming language in a statistical and probabilistic approach to simulate various hypothetical scenarios and incentivize real-world discussions around the obtained results. One of those scenarios is to model the Most Recent Common Ancestor (MRCA) in accordance to Dr. Joseph T. Chang's (Yale University) paper published on Nature magazine[1].

**Forewords:**

Even though the scripts are coded to satisfy the immediate need for a result, they are strictly coded base on the fact that the user studiously feed in valid inputs. In other words, to be essentially frank, the scripts do not catch unexpected errors such as invalid data type inputs or intential attempts to break the program by injecting range-exceeding inputs (e.g. overflowing the memory buffer, overflowing the range of float numbers).

'Terminal', 'console', 'shell', 'tty' are terms that all refer to one thing in the context of this report. Sometimes, in whosever case it may be, it can also undertake another *heinous* synonym: command prompt (or '*cmd*'). Be duly noted that, however, they will refer to different things in a more specific context.

At times, different codeblocks would have different color theme; however, the differences do not carry any significant meanings.

[1] http://www.stat.yale.edu/~jtc5/papers/CommonAncestors/NatureCommonAncestors-Article.pdf

For reference purposes, the source code can be found at my GitHub repository: https://github.com/zasshuwu/r-project-2019/

## Part 1. Installing and Learning R

### Exercise 1. Simulating Probabilities

In this exercise, I learned how to define a variable (or in R case, a vector) and assign a value to it using various methods, operations on those variables set up a dataset/dataframe, drawing bar graphs and pie charts, and randomly sample a tuple with R's built-in function. Aside from that, the exercise also implements some basic programming concepts, i.e. data types, loops, or a counter variable.

Below are the R scripts for each question in this exercise.

*Codeblock. 1. Exercise 1a*

```
### Exercise 1: Simulating Probabilities

# (a) Rolling a die
#(i) 12-sided die, 500 times, pie chart

die <- (1:12)
dieRoll <- sample(die, 500, replace=TRUE)
tb <- table(dieRoll)
piepercent <- round(100*dieRoll/sum(dieRoll), 1)
pie(tb, labels=(1:12), radius=1, col=rainbow(12))

# (ii) 12-sided die, 500 times, pie chart, 1 is x3 more likely

die <- (1:12)
probability <- c(0.3, rep(0.1, 11))
dieRoll <- sample(die,500,replace=TRUE, prob=probability)
tb <- table(dieRoll)
pie(tb, labels=(1:12), radius=1, col=rainbow(12))

# (iii) 2x(12-sided dice), 10000 times, P(sum=10)=?

die <- (1:12)
fair_prob <- rep(0.1,12)
dieRoll1 <- sample(die, 10000, replace=TRUE, prob=fair_prob)
dieRoll2 <- sample(die, 10000, replace=TRUE, prob=fair_prob)
sumn <- function(){
  counter <- 0
  for (i in 1:10000){
```

```
    if ((dieRoll1[i] + dieRoll2[i]) == 10){
      counter = counter + 1
    }
    i = i + 1
  }
  return(counter)
}
desiredoutcomes <- sumn()
estprob <- c(desiredoutcomes/10000)
theoprob <- 0.0625
perror <- (abs(estprob-theoprob))/mean(estprob,theoprob)*100
cat('Estimated probability (experimental): ', estprob, '\n')
cat('Theoretical probability: ', theoprob,'\n')
cat('Percentage error: ', perror, '\n')
cat(rep('\n',5))

################################
```

*Codeblock 2. Exercise 1b.*

```
### (b): PLAYING THE LOTTERY

# 6 numbers case
play_lotto <- function(){
  n <- 0
  ticket <- c(6,9,4,2,24,7) # here
  draw <- c(0,0,0,0,0,0) # init vect that will contain the numbers drawn
  while (!all(ticket==draw)){
    n <- n+1
    draw <- sample(1:49, 6, replace=FALSE)
    print(draw)
  }
  return(n)
}

total_draws <- play_lotto() # assign return value to a var

# Assume that one year has 52 weeks and leap years are not accounted for.
# Playing once per week -> total_plays / weeks
weeks <- 52
price_ticket <- 3
jackpot <- 16*10^6

cat('Number of total draws: ', total_draws) # print that var
cat('Number of years before winning: ', total_draws/weeks) # print number of years bef
ore winning
cat('Net earning: ', (jackpot - total_draws*price_ticket)) # print net earning

############################ END OF CODEBLOCK ##########################################
```

```r
### (b) PLAYING THE LOTTERY

# 4 numbers case
play_lotto <- function(){
  n <- 0
  ticket <- c(6,9,4,2) # here
  draw <- c(0,0,0,0) # init vect that will contain the numbers drawn
  while (!all(ticket==draw)){
    n <- n+1
    draw <- sample(1:49, 4, replace=FALSE)
    print(draw)
  }
  return(n)
}

total_draws <- play_lotto() # assign return value to a var

# Assume that one year has 52 weeks and leap years are not accounted for.
# Playing once per week -> total_plays / weeks
weeks <- 52
price_ticket <- 3
jackpot <- 16*10^6

cat('Number of total draws: ', total_draws, '\n') # print that var
cat('Number of years before winning: ', total_draws/weeks, '\n') # print number of yea
rs before winning
cat('Net earning: ', (jackpot - total_draws*price_ticket), '\n') # print net earning

############################### END OF CODE BLOCK ######################################
```

Running the codeblock yields the following results:

```
...
> weeks <- 52
> price_ticket <- 3
> jackpot <- 16*10^6
> cat('Number of total draws: ', total_draws, '\n') # print that var
Number of total draws:  670851
> cat('Number of years before winning: ', total_draws/weeks, '\n') # print number of years
before winning
Number of years before winning:  12900.98
> cat('Net earning: ', (jackpot - total_draws*price_ticket), '\n') # print net earning
Net earning:  13987447
```

As you can see, the calculated net earning for this simulation is $13,987,447. This will get you richer in 12900.98 years' time after drawing 670851 times and burning $2,012,553 in the process.

Running these scripts yield the following results in the console/terminal/cmd:

- Exercise 1ai, ii:

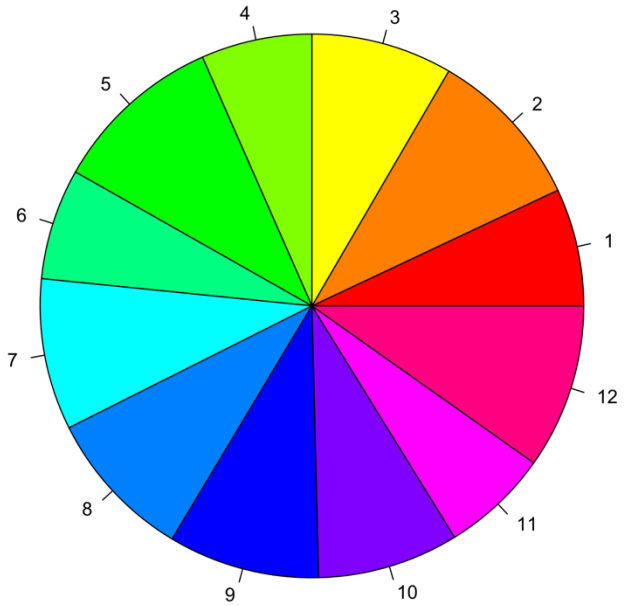



Chart 1i. Simulation of 500 Pseudo-Random Rolls of a 12-sided Die

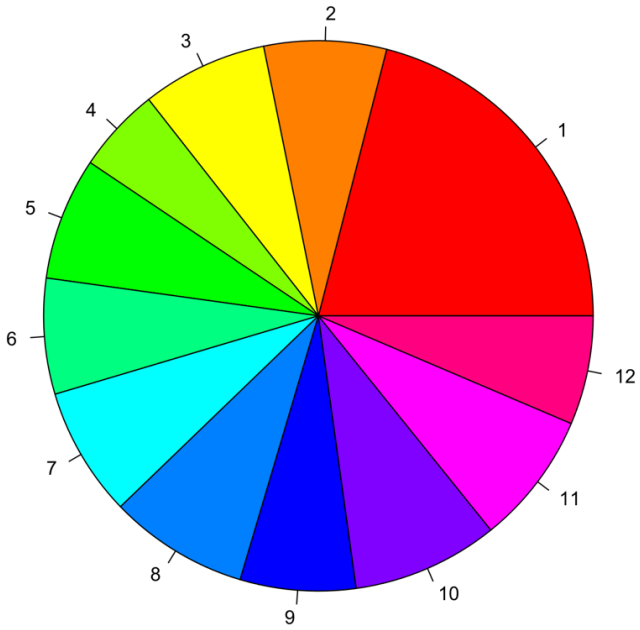



Chart 1. Simulation of 500 Pseudo-Random Rolls of a 12-sided Die with Side 1 has Triple the Probability of Getting Rolled

- Exercise 1iii:

The estimated probability yielded by running the corresponding scripts once is **0.0701**. It is to be noted that this number varies every time the script is rerun so there is no way to replicate this estimated probability recorded here.

However, we can calculate the exact theoreotical probability of getting sum of 10 when rolling the two dice. That theoretical probability is ___.

Here is the R-base terminal output from running the scripts in exercise 1aiii:

```
> cat('Estimated probability (experimental): ', estprob, '\n')
Estimated probability (experimental):  0.0643
> cat('Theoretical probability: ', theoprob,'\n')
Theoretical probability:  0.0625
> cat('Percentage error: ', perror, '\n')
Percentage error:  2.799378
```

**Exercise 2. Creating Functions in R**

In this exercise, I learned how to function are created in R. Now, since I have already had a background in C++, Python, and NodeJs, this process went pretty much smoothly. Part (a) and (b) were done in recursion to make them more exciting (and easier on the printer).

*Codeblock 2. Exercise 2*

```
##### Exercise 2: Functions

# Get user input first for n and k parameters

k <- as.integer(readline(prompt="Enter k: "))
n <- as.integer(readline(prompt="Enter n: "))

# a) Sum of integers

# Prompt user for input

sumInt <- function(n) {
  if (n == 1) return (1)
  else return(sumInt(n-1)+(n))
}

cat('Sum of integers to n: ', sumInt(8), '\n')
# Test case: 1+2+3+4+5+6+7+8 = 36

############################## END OF CODEBLOCK ###################################
```

```r
# b) Sum of integers squared

sumSquare <- function(n) {
  if (n == 1) return (1)
  else return (sumSquare(n-1)+n^2)
}

cat('Sum of Square is: ', sumSquare(n))

################################### END OF CODEBLOCK ################################
```

```r
# (c) Sum of squares from k to n

sumSquaredkn <- function(k,n) {
  sum <- 0
  for (i in k:n) {
    sum <- sum + i^2
  }
  return(sum)
}

################################### END OF CODEBLOCK ############################
```

```r
# (d) Rewriting function in (c) to call function in (b)

rewrite <- function(k,n) {
  sum <- 0
  if (n == k) {
    return(k^2)
  }
  else {
    while (k <= n) {
      sum <- sum + k^2
      k <- k+1
    }
    return(sum)
  }
}

cat("Sum of squared k to n = ", rewrite(9,9), "\n")
# Test case 1: rewrite(8,9) = 145
# Test case 2: rewrite(1,9) = 285
# Test case 3: rewrite(9,9) = 81

################################### END OF CODEBLOCK ################################
```

Running the R scripts presented above in exercise 2 with regards to test cases yielded the following output in the console/terminal/cmd:

- Exercise 2a. Sum of Integers:

```
> cat('Sum of integers to n: ', sumInt(8), '\n')
Sum of integers to n:  36
> 1+2+3+4+5+6+7+8
[1] 36
```

- Exercise 2b. Sum of Squared Integers:

```
> cat('Sum of Square is: ', sumSquare(9), '\n')
Sum of Square is:  285
```

- Exercise 2c. Sum of Squared Integers from $k$ to $n$:

```
> cat('Sum is: ', sumSquaredkn(6,9), '\n')
Sum is:  230
```

- Exercise 2d. Rewriting to call (c) in (b):

```
> cat("Sum of squared k to n = ", rewrite(8,9), "\n")
Sum of squared k to n =  145
> cat("Sum of squared k to n = ", rewrite(1,9), "\n")
Sum of squared k to n =  285
> cat("Sum of squared k to n = ", rewrite(9,9), "\n")
Sum of squared k to n =  81
```

**Exercise 3. Probability Distributions**

This exercise deals directly with whatever I have learned throughout the course in the field of probability. The types of distribution I encountered while completing this exercise varied, including but not exclusively, binomial distribution, Poisson distribution, pmf, cdf, pdf, normal distribution, uniform distribution, chi-squared, Fisher's distribution, etc. This was where the stakes were higher since applying these distributions into the code rquired one to know which parameters were permissible in the R library and the *stats* package. One website that I constantly relied on was the *documentation website for the stats package*[2].

*Exercise 3a. Estimating winning Tim Horton cups in a Kirkland store*

[2] https://stat.ethz.ch/R-manual/R-patched/library/stats/html/00Index.html. Accessed on 10/03/2019. This documentation is applicable for R version 3.6.1 from CRAN.

In this exercise, I decided to go with the hypegeometric distribution since it best fits for this problem.

*Codeblock 3a.*

```
# (a) Kirkland Cups problem with hypergeometric distribution
total_cups <- 33357600
good_cups <- total_cups/6 # winning cups
bad_cups <- total_cups - good_cups # non-winning cups
kirkland_cups <- 8000 # cups given to kirkland
desired_cups <- 1300 # desired cups to win

cat("dhyper (quantile only) = ", dhyper(desired_cups, good_cups, bad_cups, kirkland_cups))

cat("phyper (cumulative) = ", phyper(desired_cups, good_cups, bad_cups, kirkland_cups))
```

Results yielded in the terminal when the codes are run:

```
> cat("dhyper (quantile only) = ", dhyper(desired_cups, good_cups, bad_cups, k$
dhyper (quantile only) =  0.007308059
> cat("phyper (cumulative) = ", phyper(desired_cups, good_cups, bad_cups, kirk$
phyper (cumulative) =  0.1623035
```

*whereas '$' denotes output cut-off point*

*Exercise 3b. Number of shots on goal made by the Canadiens exceeding 2400.*

In this exercise, it is clearly stated that the number of shots on goal by the Canadiens during a hockey game follow a Poisson distribution. The notation for this problem is: $Pois \sim \left(2400, \frac{avgscore}{season}\right)$

*Codeblock 3b.*

```
# The hockey game exercise
e_shots_game <- 29
games <- 82
e_shots_season <- e_shots_game * games

# trying to find the probability of getting more than 2400 shots per season...
# P(X>=2400) = P(X=0) + ... + P(X=2400)
interesting_outcome <- 2400
complement <- function(){
    total_probability <- 0
        for (x in 0:(interesting_outcome-1)) {
            total_probability <- total_probability + dpois(x, e_shots_season)
    }
    return(total_probability)
```

```
}

# or alternatively, using the cdf function ppois(x,lambda,lower.tail):
cat('The probability is: ', ppois(2400, lambda = 29*82, lower.tail = FALSE), '\n')

p <- 1 - complement()
print(p)
```

Results yielded in the terminal when the codes are run:

```
> cat('The probability is: ', ppois(2400, lambda = 29*82, lower.tail = FALSE), '\n')
The probability is:  0.3212928
> p <- 1 - complement()
> print(p)
[1] 0.3212928
```

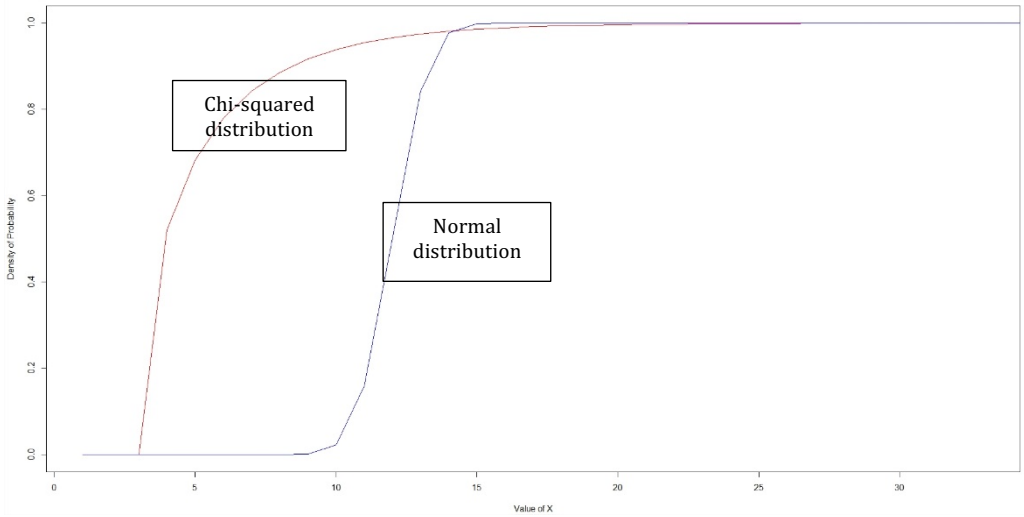
*Exercise 3c. On same graph, plot cdf for 2 continuous distributions (chi-squared and Fisher)*

*Codeblock 3c.*

```
# (c) Plotting the cdf for X^2 and Normal Distribution
xlabel <- "Value of X"
ylabel <- "Density of Probability"
maintitle <- "The probability of obtaining a corresponding x value with probability y
of a normal distribution and chi-squared distribution"
plot(pchisq(-
1:15, df=1), type="line", col="red", xlab=xlabel, ylab=ylabel, sub=maintitle,font.sub=
2)

lines(pnorm(-1:15, mean = 3, sd = 4), col="blue")
```

Running the codeblock above yields the following result:




Graphs of cumulative distribution function of chi-squared distributionand normal distribution plotted using R's built-in plotting functions.