MT5763 Individual Assignment 2

Erna Kuginyte, 220013309

30/10/2022

Find the R Markdown file at **the GitHub Repository**

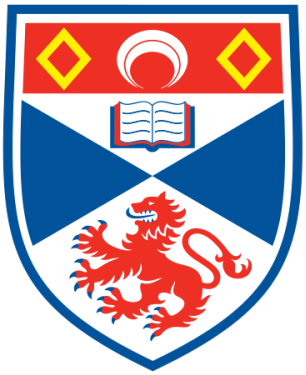
Or copy and paste the link: https://github.com/ekuginyte/MT5763_2_220013309.git

Abstract

This report summarises the functions created to replicate Monte Carlo simulation and Bootstrapping methods.

Problem A includes simple Monte Carlo simulation to calculate Pr(X > Y) with given X Normal distribution and Y Uniform distribution. The plotted sample variance distribution shows that the variance gets close to zero as number of MC simulation increases.

Problem B applies MC simulation and bootstrapping to replicate the tournament format and the effect of total number of games played from probability of winning. The effect driven by this tournament format comes out to be unfair.



University of St Andrews

Problem A

Include Libraries

```
# Include libraries
library(tidyverse)
library(ggpubr)
library(knitr)
library(doParallel)
```

Monte Carlo Simulation to compute the probability Pr(X > Y), with X variable being sampled form normal distribution with mean = 4, standard deviation of square root of 10, and Y variable being sampled from uniform distribution with minimum value of 2, maximum of 8. [1] The function my_mc() takes in value n, the sample size for each variables X and Y. Refer to Pr(X > Y) result in Table 1. The result is reasonable.

```
# Problem A
# Sample size
n <- 10000
# Pre-allocate memory for storing random samples
X <- rep(NA, n)
Y <- rep(NA, n)
prob <- NA
# For reproducibility
set.seed(777)
# Simple Monte Carlo function to calculate probability Pr(X > Y)
# Input:
#  X - samples from normal distribution,
#  Y - samples from uniform distribution.
# Output:
#  prob - probability Pr(X > Y).
my_mc <- function(n) {
  # Generate samples with given boundaries
  X <- rnorm(n, mean = 4, sd = sqrt(10))
  Y <- runif(n, min = 2, max = 8)
  # Compute probability Pr(X > Y)
  prob <- sum(X > Y) / length(X + Y)
  # I wouldn't use return command normally in this place, since the Tidyverse
  # Style Guide advice against it, unless I wanted the return earlier in the
  # function. However, I have heard this sometimes gets penalised at the
  # university, hence, the return command at the end of the function.
  return(prob)
}
# Return results in a table
kable(my_mc(n), caption = "Pr(X > Y)")
```

Table 1: Pr(X > Y)

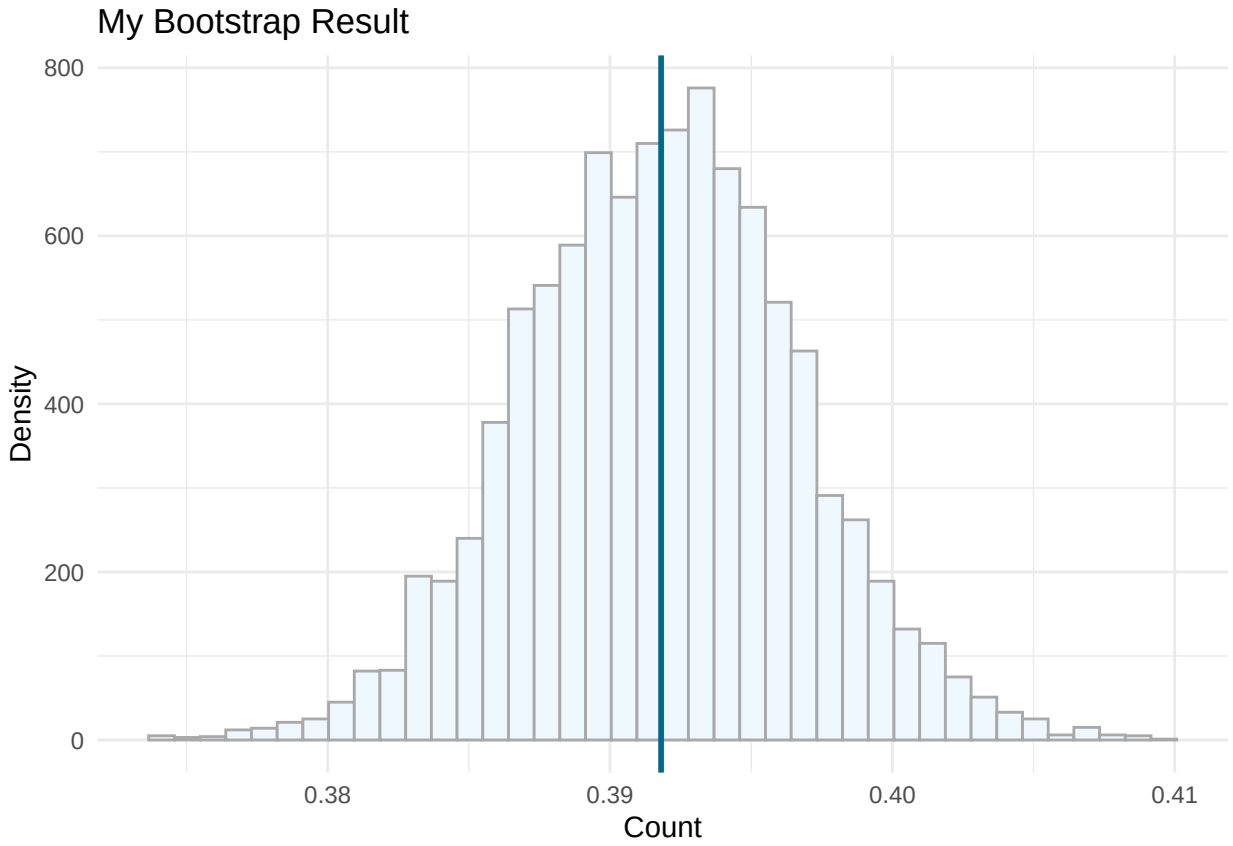
x
0.3891

[1] (Sigal and Chalmers, 2016)

Bootstrapping method to derive the sampling distribution for the estimate of Pr(X > Y). The function my_bts() takes in n_repeat value to return n_repeat number of Pr(X > Y) probabilty values from the my_mc() function. See the plot below of the Bootsrapping distribution. The plot looks reasonable, it does not indicate abnormal distribution.

```
# Bootstrapping to derive the sampling distribution of Pr(X > Y)
# Sample size
n_repeat <- 10000
# Pre-allocate memory for storing results
boot_res <- rep(NA, n_repeat)
prob_res <- rep(NA, n_repeat)
# For reproducibility
set.seed(555)
# Loop across all samples
# Output:
#  boot_res - store vector of the Pr(X > Y) distribution.
my_bts <- function(n_repeat) {
  # Compute vector of the Pr(X > Y) distribution, parallelise
  prob_res <- foreach(i = 1:n_repeat, .combine = "c") %dopar% {
  my_mc(n)
    }
  # Resample with replacement
  boot_res <- sample(x = prob_res, size = n_repeat, replace = TRUE)
  # Store results
  return(boot_res)
}
```

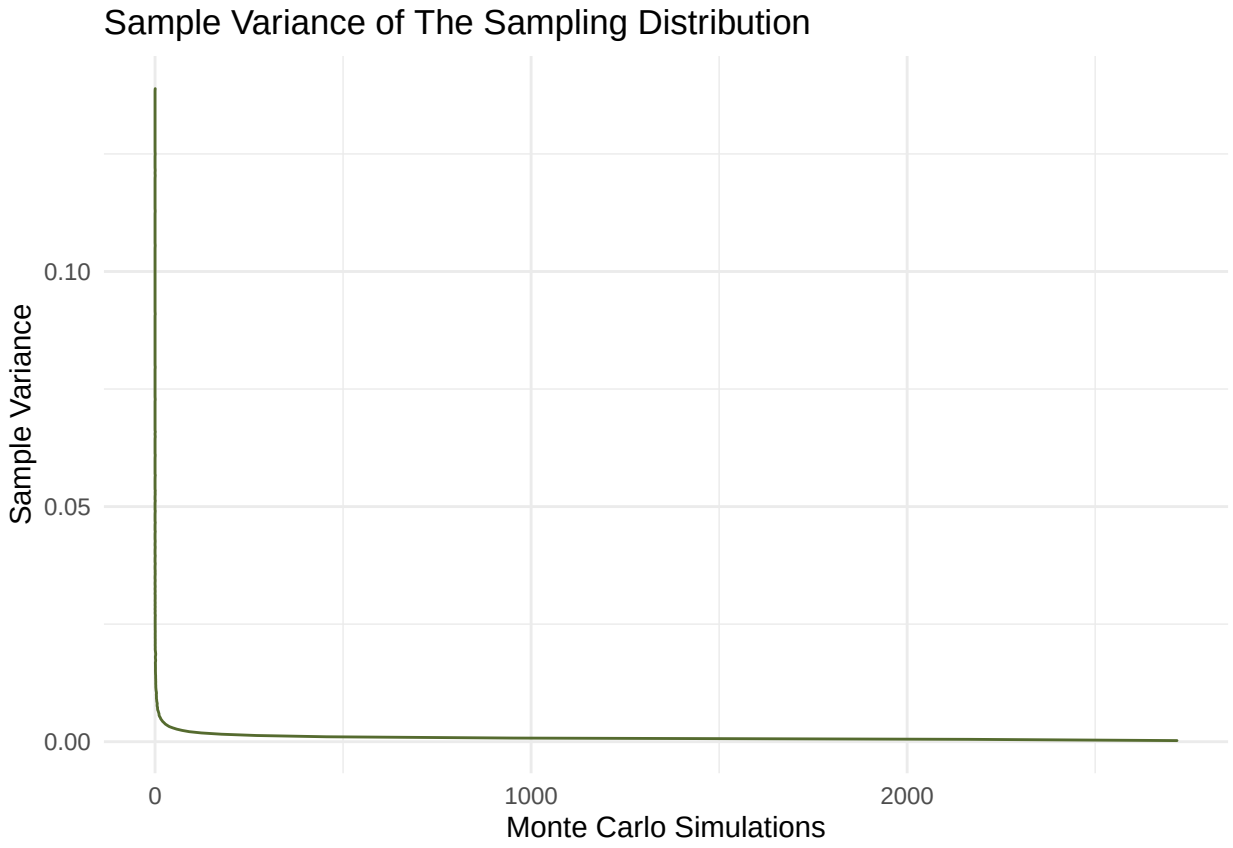
```
# Save bootstrap results
res_bts <- my_bts(n_repeat)
# Save the mean of the bootstrap results
res_bts_mean <- mean(res_bts)
# Plot the bootstrap results
ggplot() +
  geom_histogram(aes(x = res_bts), colour = "darkgrey", fill = "aliceblue",
                 bins = 40) +
  geom_vline(xintercept = res_bts_mean, colour = "deepskyblue4", size = 1) +
  xlab("Count") + ylab("Density") +
  ggtitle("My Bootstrap Result") +
  theme_minimal()
```



Sample variance changes with the number of Monte Carlo Simulations. The variance_change() function takes in value n_sim to repeat the simulation n_sim times. The function extracts probabilities and then computes variance change as the number of simulation increases. See the plot "Sample Variance of The Sampling Distribution" below, it shows that as the number of simulations of Monte Carlo method increases, variance decreases.

```
# Number of simulations to run
n_sim <- 10000
# Pre-allocate memory to store vector of variances
sample_variances <- rep(NULL, n)
simulation <- rep(NULL, n)
# For reproducibility
set.seed(222)

# Create function to produce variances as number of Monte Carlo simulations
# increase.
# Input:
#  n - number of simulations ran.
# Output:
#  sample_variances - vector of Monte Carlo sample variances.
variance_change <- function(n_sim) {
  # Store multiple values of Monte Carlo simulation probabilities Pr(X > Y)
  simulation <- foreach(i = 1:n_sim, .combine = "c") %dopar% {
  my_mc(i)
  }
  # Store each variance of the i:(i + 1) simulations
  sample_variances <- foreach(i = 1:n, .combine = "c") %dopar% {
    var(simulation[1:(i + 1)])
```

```
  }
  return(sample_variances)
}

# Save variance distribution to p
p <- variance_change(n_sim)
# Sample variance changes plot
ggplot() +
  geom_density(aes(y = p), colour = "darkolivegreen") +
  xlab("Monte Carlo Simulations") +
  ylab("Sample Variance") +
  ggtitle("Sample Variance of The Sampling Distribution") +
  theme_minimal()
```



Problem B

Football tournament. A team keeps playing until they accrue 7 wins or 3 losses (whichever comes first - no draws allowed). Assumed fixed win rate p[0, 1] across all rounds (they are paired at random). Function tournament() takes in probabilities p [0, 1] and computes the number of games played to end a match. The p_function() passes through the probabilites to the tournament function and outputs number of games played from probabilities of winning.

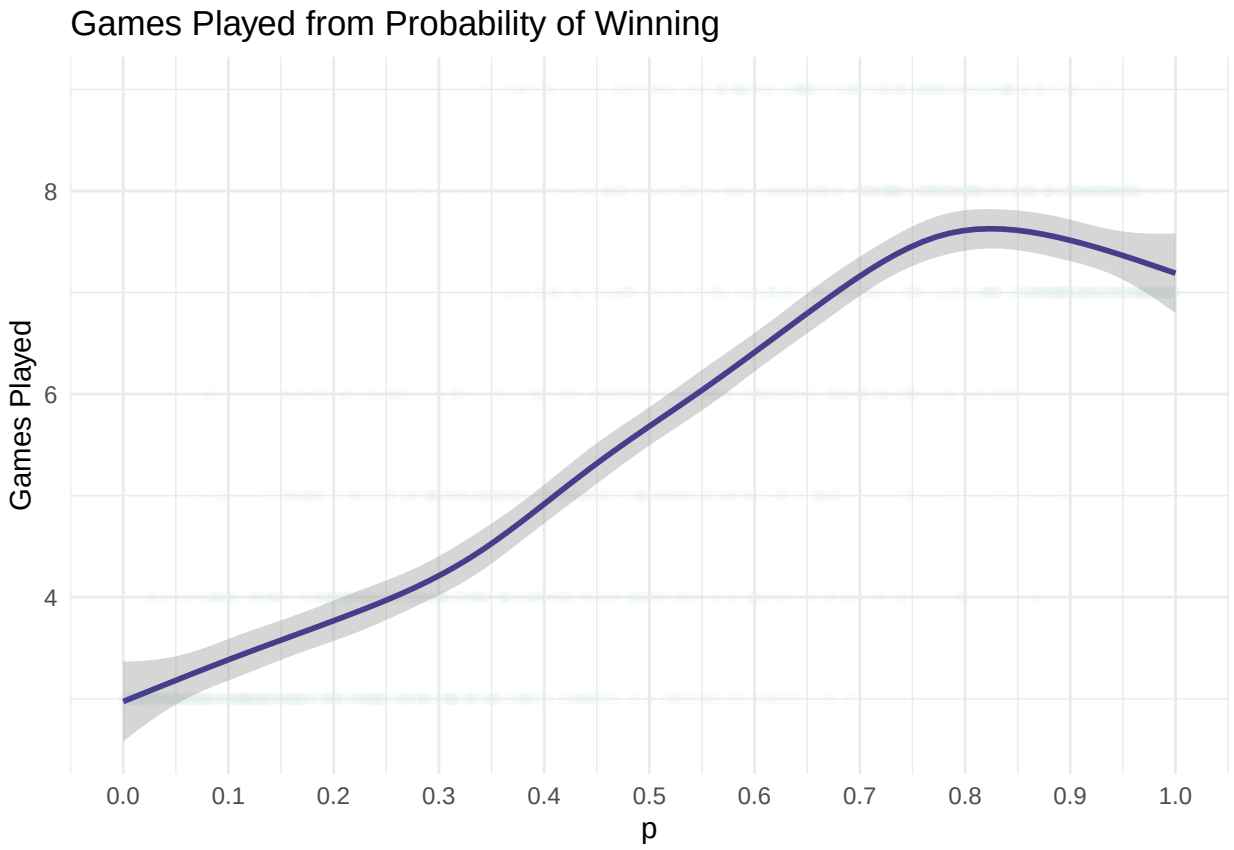
```
# Count number of games the team needs to play to win or loose the match.
# The team needs to accrue 7 wins or 3 losses (no draws).
# For reproducibility
set.seed(333)
```

```r
# Input:
#  w - wins to finish the match,
#  l - losses to finish the match.
# Output:
#  game_n - number of games played for the match to be finished.
tournament <- function(prob) {
  # Pre-allocate memory
  games <- NA
  # Assign counter
  w <- 0
  l <- 0
  # Conditions
  wins <- 7
  loses <- 3
  while(w < wins && l < loses) {
    result <- rbinom(1, 1, prob)
    if(result == 1) {
      w <- w + 1
    }
    else {
      l <- l + 1
    }
  }
  games <- w + l
  return(games)
}

# Set p values
p <- seq(0, 1, 0.001)
# Number of times to repeat the function
n_rep <- 10000
# Create function p
p_function <- function(n_rep) {
  # Store results
  result_list <- NULL
  # Create p function with p values on x axis, total games played on y axis
  result_list <- foreach(i = 1:length(p), .combine = "c") %dopar% {
    tournament(p[i])
  }
  return(result_list)
}
```

Plot how the total number of matches played (i.e. wins + losses) varies as a function of p. As expected, when probability of winning increases, the number of games played increases too. That is particular to this tournament, since to win a game, there have to be 7 wins. It's a lot quicker to lose, since number of loses to end a match is 3. The curve slowly rises at the start because there are more simply more possible outcomes to lose than there are to win.

```r
# Store results of number of games played to plot
games_played <- p_function(n_rep)

# Function p plot
ggplot() +
  geom_point(aes(x = p, y = games_played), colour = "azure2", alpha = 0.1) +
```

```
  geom_smooth(aes(x = p, y = games_played), colour = "darkslateblue") +
  scale_x_continuous(breaks = seq(0, 1, 0.1)) +
  xlab("p") +
  ylab("Games Played") +
  ggtitle("Games Played from Probability of Winning") +
  theme_minimal()
```

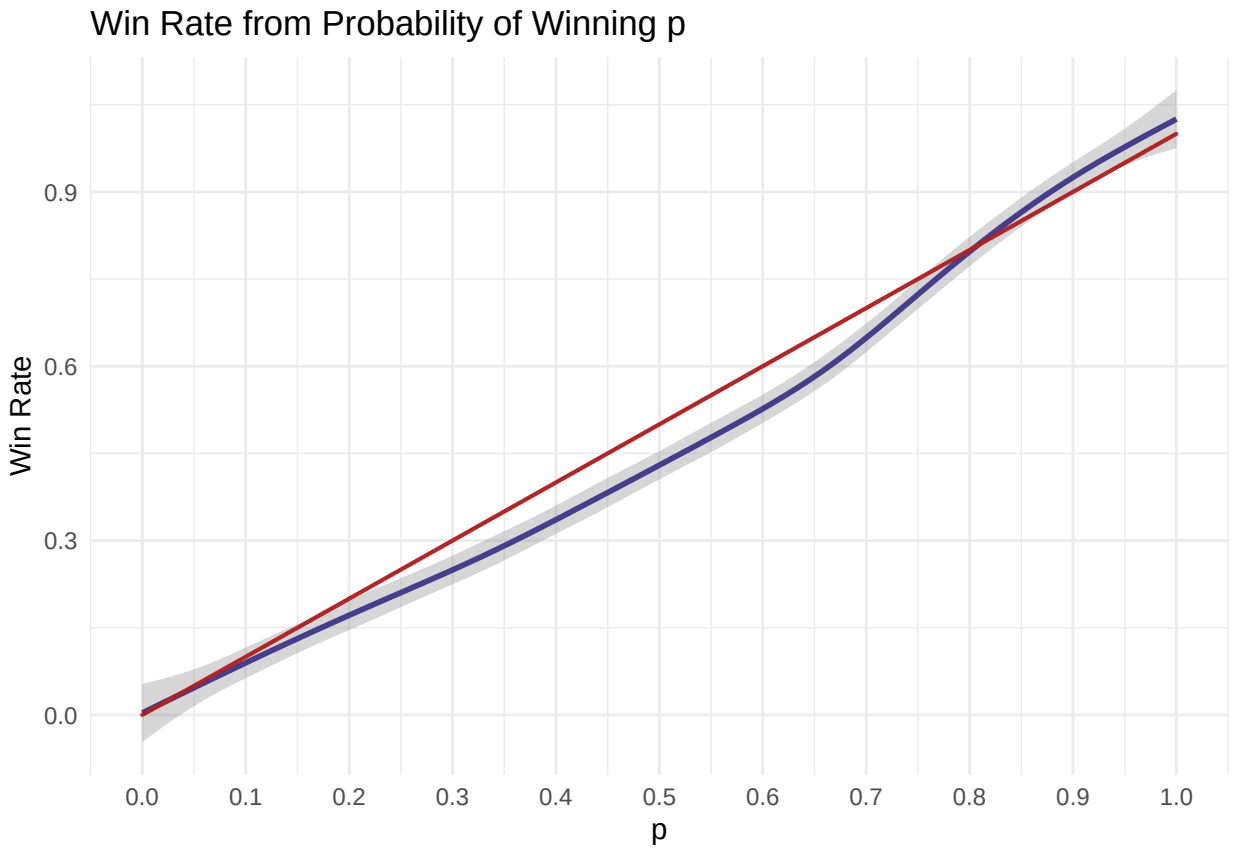


Compute the observed win rate using win_rate() function, using probabilities p[0, 1]. The assumed win rates are directly related to the p probabilities. Plot to compare the results, see the graph below. Ran a t-test to see if there is difference between the theoretical values and the observed (simulated) values. Large p-value does not indicate difference. See Table 1. The effect driven by this format is slightly too far from the assumption values, which means it is a relatively unfair as a tournament structure. It dips in win rate quite a bit from p 0.3 to p 0.7. Decreasing the count of games that need to be won might help to even the odds out.

```
# Compute expected net winning
#expectedNetWin <- mean(netWin)

# Win rate = total wins/all games
win_rate <- function(prob) {
  # Pre-allocate memory
  games <- NA
  win_rate_res <- NA
  # Assign counter
  w <- 0
  l <- 0
  # Conditions
  wins <- 7
```

```
  loses <- 3
  while(w < wins && l < loses) {
    result <- rbinom(1, 1, prob)
    if(result == 1) {
      w <- w + 1
    }
    else {
      l <- l + 1
    }
  }
  # Total number of games played
  games <- w + l
  # Win rate
  win_rate_res <- w / games
  return(win_rate_res)
}

# Set p values
p <- seq(0, 1, 0.001)
# Number of times to repeat the function
n_times <- 10000
# Create function p
win_rate_obs <- function(n_times) {
  # Store results
  result_list <- NULL
  # Create p function with p values on x axis, total games played on y axis
  result_list <- foreach(i = 1:length(p), .combine = "c") %dopar% {
    win_rate(p[i])
  }
  return(result_list)
}

# Store results of number of games played to plot
win_rate_obs_res <- win_rate_obs(n_times)

# Assumed win rate
ab <- p

# Function p plot
ggplot() +
  geom_smooth(aes(x = p, y = win_rate_obs_res), colour = "darkslateblue") +
  geom_point(aes(x = p, y = ab), size = 0.1, colour = "firebrick", alpha = 0.7) +
  scale_x_continuous(breaks = seq(0, 1, 0.1)) +
  xlab("p") +
  ylab("Win Rate") +
  ggtitle("Win Rate from Probability of Winning p") +
  theme_minimal()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```




```
# Run a two-tailed t-test for assumed vs observed win rate
#  H0 - no difference
#  H1 - there is a difference
test_win_rates <- t.test(x = ab, y = win_rate_obs_res)

# Return result in a table
kable(test_win_rates$p.value, caption = "p-value form T-Test")
```

Table 2: p-value form T-Test

x
0.0303661

Bibliography

1. Matthew J. Sigal & R. Philip Chalmers (2016) Play It Again: Teaching Statistics With Monte Carlo Simulation, Journal of Statistics Education, 24:3, 136-156, DOI: 10.1080/10691898.2016.1246953