Simulation Study for MT5763

Nico Herrig

2022-10-15

Github Repo

```
library(tidyverse)
library(parallel)
library(scales)
library(bibtex)
```

```
# Setting seed for parallel computing:
RNGkind("L'Ecuyer-CMRG")
set.seed(0911)
```

Important note at the beginning: As there seems to be no consent in the literature about the number of Monte Carlo simulations required for obtaining sufficient results, I followed the recommendation of Liu (2017) who aims for numbers between 100k and 500k for a proper use of the law of large numbers theorem. As this "accuracy" comes at the cost of longer computation time, some code chunks might take several seconds for computation.

Problem 1

Description: Consider the following independent random variables:
X ~ N($\mu = 4$, $\sigma^2 = 10$)
Y ~ U(a = 2, b = 8).
Compute $Pr(X > Y)$
Use bootstrapping to derive the sampling distribution for your estimate of $Pr(X > Y)$
Show how the sample variance of this sampling distribution changes as a function of the number of Monte Carlo simulations.

For the underlying problem, a sample containing 100,000 random deviates from X ~ N($\mu = 4$, $\sigma^2 = 10$) and Y ~ U(a = 2, b = 8) is used.

```
RNGkind("L'Ecuyer-CMRG")
set.seed(0911)


probability_calculator <- function(n) {

  X <- rnorm(n, mean = 4, sd = sqrt(10)) #generating deviates
  Y <- runif(n, min = 2, max = 8)
```

```
  # Calculating Pr(X>Y)
  Pr_hat <- sum(X > Y) / n #calculating Pr(X>Y)

  output <- list(X = X,
                 Y = Y,
                 Pr_hat = Pr_hat)

  return(output)
}


results <- probability_calculator(100000) #running 100k simulations


print(results[3])
```

```
## $Pr_hat
## [1] 0.39229
```

Calculating $\widehat{Pr}(X > Y)$ from the initial sample without any further methods, we derive a value of 0.39229 .

To derive the distribution of $\widehat{Pr}(X > Y)$, we use a non-parametric bootstrap. One benefit of this technique is that it does not rely on the assumption of normally distributed data. The following chunk of code defines a function for such bootstrap.

```
# Function for bootstrap procedure, using parallel computing technique for
# speeding up the computation.
bootstrap_multicore_problem1 <- function(n_straps, vec1 = X, vec2 = Y) {
  # bootstrapping input vec1 and vec2 over n_straps iterations
prob_vector <- unlist(mclapply(1 : n_straps, function(n = n, vec1 = X, vec2 = Y)
  {
  # sampling both vectors X & Y
  resX <- vec1[sample(1 : length(vec1), length(vec1), replace = TRUE)]
  resY <- vec2[sample(1 : length(vec2), length(vec2), replace = TRUE)]

  Prob <- sum(resX > resY) / length(resX) # calculating Pr(X>Y) of each
  return(Prob)                            # bootstrap sample
}, mc.cores = 6, mc.set.seed = TRUE))
return(prob_vector)

}
```

The bootstrap algorithm above re-samples the vectors X and Y *with replacement*, calculates the resulting $\widehat{Pr}(X > Y)$, and repeats this procedure n times. The algorithm generates a vector with n probabilities. Using the bootstrap algorithm, the distribution of $\widehat{Pr}(X > Y)$ can now be evaluated.

```
RNGkind("L'Ecuyer-CMRG")
set.seed(0911)

X <- unlist(results[1]) #transforming the output of the probability_calculator
Y <- unlist(results[2]) #function into vectors
```

```r
#NOTE: as I use 3000 bootstraps, computation can take some time.
prob_vec <- bootstrap_multicore_problem1(n_straps = 3000)
```

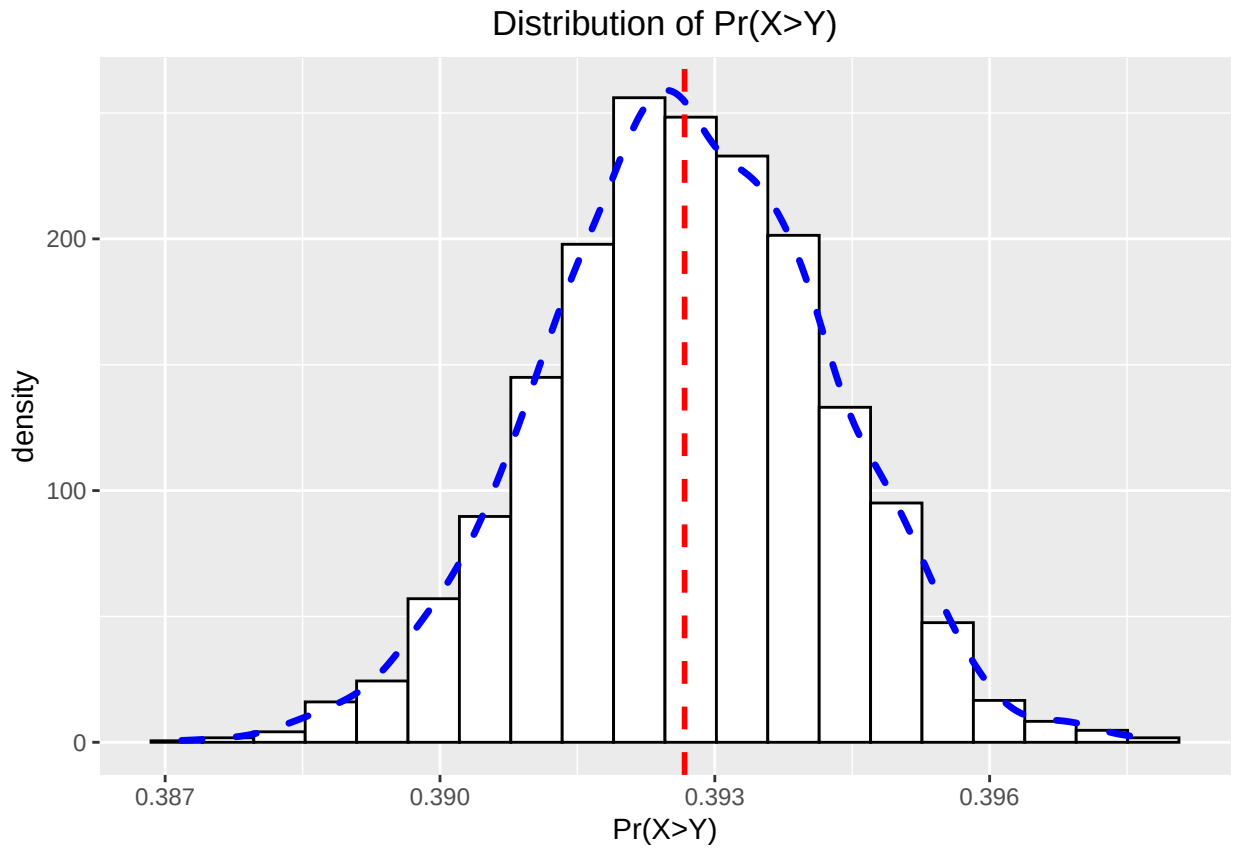
1. Point estimates (Quantile):

```r
quantile(prob_vec, c(0.025, 0.5, 0.975))
```

```
##      2.5%       50%     97.5%
## 0.3896100 0.3926700 0.3956203
```

2. Visualization (histogram):

```r
df_probabilities <- as.data.frame(prob_vec)

df_probabilities %>%
ggplot(aes(x = prob_vec)) +
  geom_histogram(aes (y = ..density..),
                 bins = 20,
                 colour = 1,
                 fill = "white")+
  geom_density(lwd = 1.2,
               linetype = 2,
               colour = "blue")+
  xlab("Pr(X>Y)")+
  ggtitle("Distribution of Pr(X>Y)")+
  theme(plot.title = element_text(hjust= 0.5))+
  geom_vline(xintercept = median(prob_vec), colour = "red", lty='dashed', lwd=1)
```



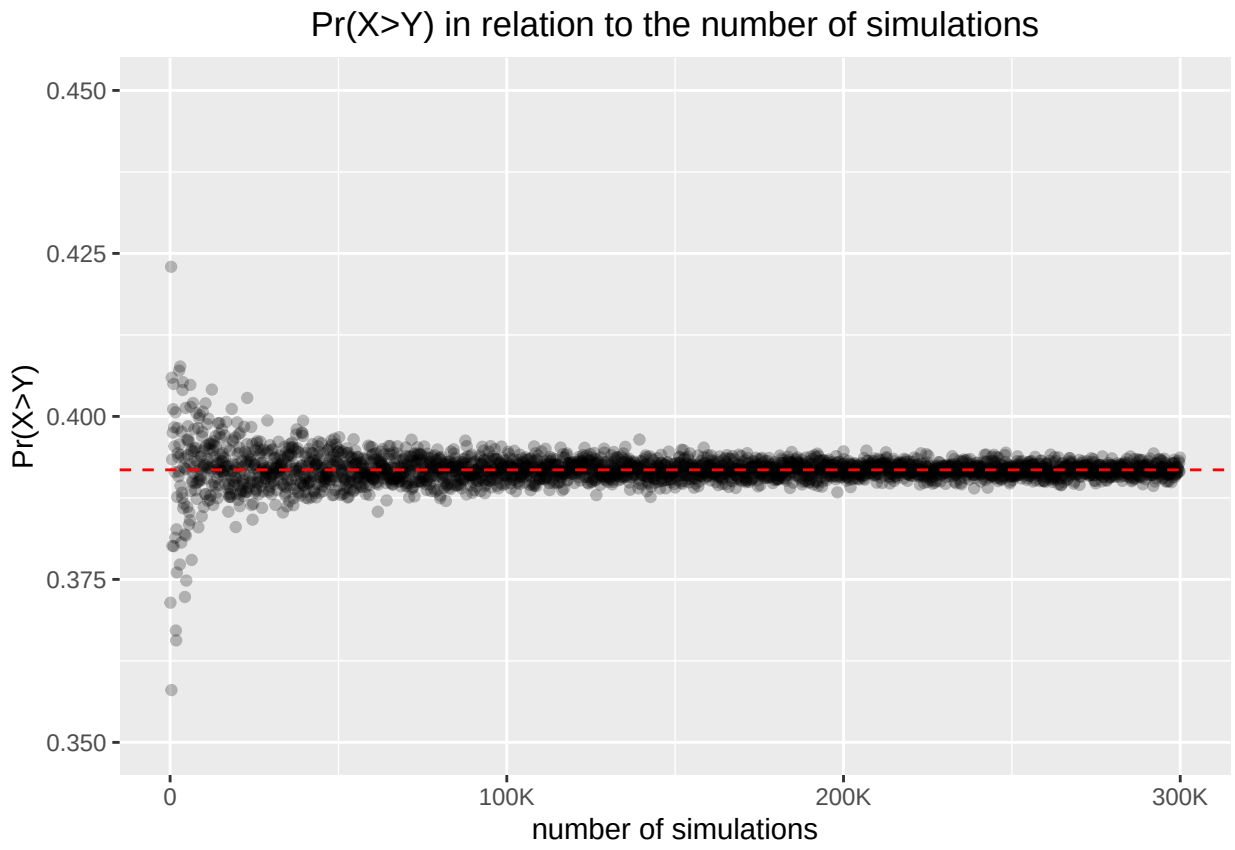
From the observed data we can infer that, after bootstrapping the original sample, the data is normally distributed around the central value (median) of 0.39267

Lastly, it is of interest how the sample variance of the sampling distribution changes with dependence on the simulations run. This part of the experiment is based on the law of large numbers theorem. As an observer, we can assume that with increasing number of simulations (increasing number of random deviates used), our simulated $\widehat{Pr}(X > Y)$ approximates the expected value of $Pr(X > Y)$ (Dekking et al. (2005)).
To test this assumption, we analyse the behavior of $\widehat{Pr}(X > Y)$ for n simulations, where n is a vector from 5 to 300,000 by steps of 100 simulations.

```
RNGkind("L'Ecuyer-CMRG")
set.seed(0911)
# vector for the number of simulations/deviates, as a sequence from 5 to 300k
# by steps of 100.
n_deviates <- seq(5, 300000, by = 100)

# Generating n deviates, corresponding to the vector defined above, and
# calculating Pr(X>Y) from the used deviates.
prob <- unlist(mclapply(n_deviates, function(i) {

    x <- rnorm(i, mean = 4, sd = sqrt(10))
    y <- runif(i, min = 2, max = 8)

    prob <- sum(x > y) / length(x)
    deviates <- i
```

```
    return(prob)

}, mc.cores = 6, mc.set.seed = TRUE)
)
```

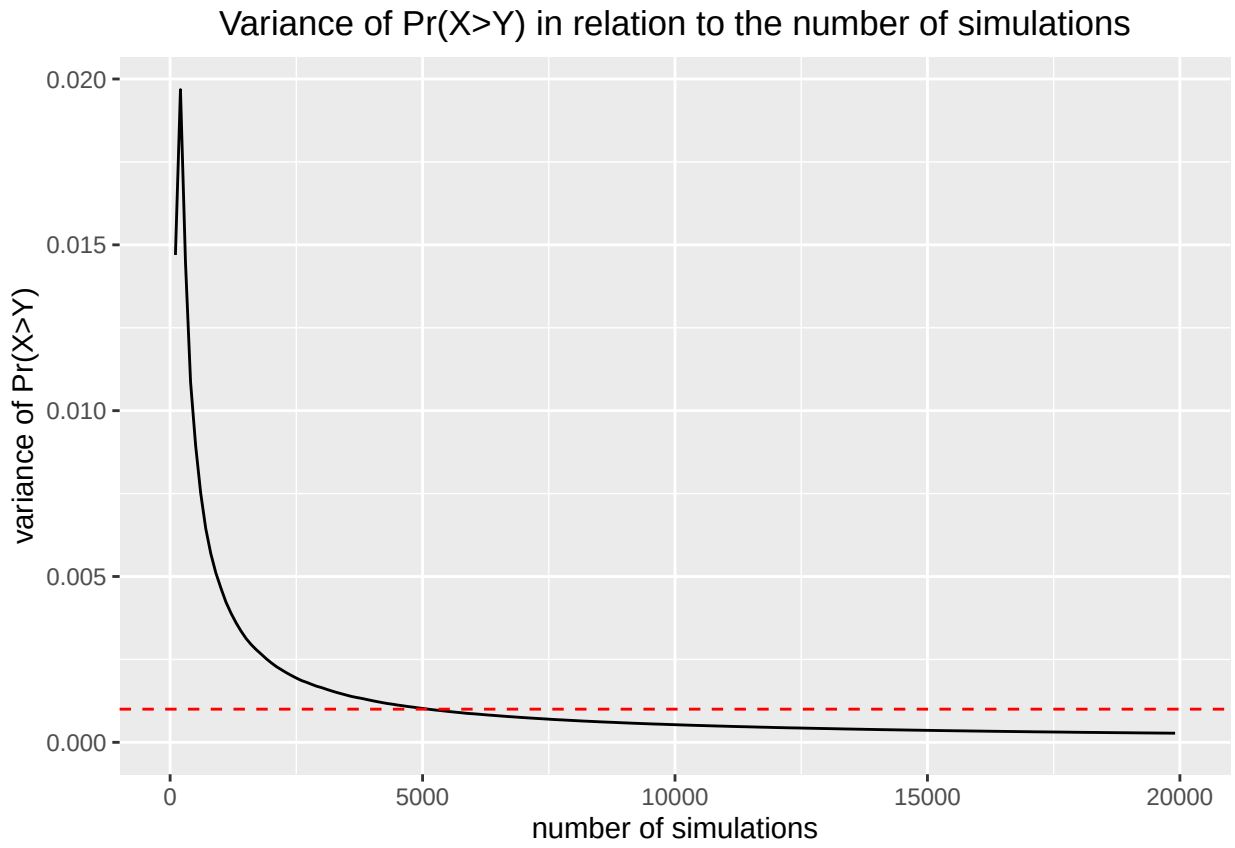
```
results_1.3 <- data.frame(prob, n_deviates)


results_1.3 %>%
  ggplot(aes(x = n_deviates, y = prob)) +
  geom_point(alpha = 1/4)+
  geom_hline(yintercept = median(prob), colour = "red", lty='dashed', lwd=0.5)+
  xlab("number of simulations")+
  ggtitle("Pr(X>Y) in relation to the number of simulations")+
  scale_y_continuous(name="Pr(X>Y)", limits=c(0.35, 0.45),
                     labels = label_number_si())+
  scale_x_continuous(name="number of simulations",
                     labels = label_number_si())+
  theme(plot.title = element_text(hjust= 0.5))
```



The plot above shows that with an increasing number of simulations used, the observed $\widehat{Pr}(X > Y)$ converges to its expected value (Britannica (2020)).

It can be assumed that for our simulation, $|\widehat{Pr}(X > Y) - Pr(X > Y)| \to 0$ as $n \to \infty$.

```r
variance <- c() # empty storage vector

for (i in 1 : (length(prob) - 1)) {
  variance[i] <- var(prob[1 : (i + 1)]) #variance of interval [t, t+i]
}

results_1.3.2 <- data.frame(variance, n_deviates[2:3000])

results_1.3.2 %>%
  ggplot(aes(x = n_deviates.2.3000., y = variance))+
  geom_line()+
  scale_y_continuous(name="variance of Pr(X>Y)", labels = label_number_si())+
  ggtitle("Variance of Pr(X>Y) in relation to the number of simulations")+
  theme(plot.title = element_text(hjust= 0.5))+
  xlim(c(0, 20000))+
  xlab("number of simulations")+
  geom_hline(yintercept = 0.001, color = "red", lty='dashed', lwd=0.5)
```



A look at the total sample variance shows that the variance approaches zero arbitrarily closely the more simulations are run, which underlines the assumption made above. The horizontal line, indicating a variance of *0.001*, is passed when the number of simulations exceed 5000.

Problem 2

Description: Consider the following football tournament format: a team keeps playing until they accrue 7 wins or 3 losses (whichever comes first - no draws allowed). Assume a fixed win rate $P \in [0, 1]$ across all

rounds (they are paired at random).
Plot how the total number of matches played (i.e. wins + losses) varies as a function of p.
Comment on the observed win rate relative to the assumed win rate p (i.e. if a team obtains 2 wins - 3 losses, the maximum likelihood point estimate for their win rate is 40%). Specifically, focus on the effect driven by the format of this tournament.

First, an algorithm simulating the above described tournament is needed. The code below shows a function for simulating the process.

```
# function for simulating the tournament
tournament_sim <- function(p_win) {
  p_loss <- 1 - p_win #loss rate
  outcome <- c("win", "loss")
  results_storage <- c() #empty vector for storing match results

# as you never play more than 9 games
# (6 wins + 3 losses (=9) or 2 losses and 7 wins (=9))
for (i in 1 : 9) {

  # simulating matches with sampling from c("win", "loss") with given
  # probabilities and replacement.
  # Adding the result to storage vector.
  results_storage <- c(sample(outcome, size = 1,
                              replace = TRUE,
                              prob = c(p_win, p_loss)), results_storage)

  # Conditions for winning or loosing the tournament
    if (length(results_storage[results_storage == "win"]) == 7) {
      break
    }

    if (length(results_storage[results_storage == "loss"]) == 3) {
      break
   }
  }

  true_winrate <- (sum(str_count(results_storage, pattern = "win")) /
                     length(results_storage))

  #binding output together as a list
  output <- list(played_matches = length(results_storage),
                 overview = table(results_storage),
                 true_winrate = true_winrate)

  return(output)
}
```

To solve the problem, a fixed expected win rate p has to be defined. The following assumes a win rate per round of 75%, so $p = 0.75$.

```r
# declaring p
p <- 0.75

RNGkind("L'Ecuyer-CMRG")
set.seed(0911)

#simulations one single tournament
results <- tournament_sim(p_win = p)
```

Playing one tournament, the observed results are as following:

1. Overall results:

```r
results$overview
```

```
## results_storage
## loss  win
##    1    7
```

2. Number of matched played:

```r
results$played_matches
```

```
## [1] 8
```

3. True win rate

```r
results$true_winrate
```

```
## [1] 0.875
```

It now should be taken into consideration how the *number of total matches* changes if we alter p. As the conditions for ending a tournament are either 7 wins or 3 losses, the number of played matches can be defined as $x \in \mathbb{N}[3, 9]$, where x is the number of matches played. Win rates from $p = 0.1$ to $p = 0.9$ are taken into consideration, with steps of 0.1.

```r
# Generating a vector containing the win rates of interest
win_rates <- seq(from = 0.1, to = 0.90, by = 0.1)
```

The algorithm described below takes a vector of expected win rates (*rates*) and the number of simulations per win rate n as input. It then simulates n tournaments per win rate and stores the number of played matches per tournament in a corresponding storing matrix.

```
tournament_stats_ngames <- function(n, rates = win_rates) {

  # generating a matrix for storing the results
  store <- matrix(nrow = length(rates), ncol=n)

  # iterations i are corresponding the the number of rates the algorithm
  # shall take into consideration
  for (i in 1 : length(rates)) {
    prob <- rates[i]

    # simulating n tournaments per win rate and storing the number of matches
    # played into the matrix "store"
    store[i,] <- unlist(mclapply(1 : n, function(i){
    as.integer(tournament_sim(p_win = prob)[1])
  }, mc.cores = 6, mc.set.seed = TRUE))
}
  return(store)
}
```

As a next step, 10,000 matches *for each p* are simulated and stored in a matrix (*table_n_matches*).

```
RNGkind("L'Ecuyer-CMRG")
set.seed(0911)
# simulating 10,000 tournaments per fixed win rate (variable win_rates [0.1, 0.9] is
# set as default)
table_n_matches <- tournament_stats_ngames(n = 10000)
```

The matrix containing the number of matches per tournament is put into a non-parametric bootstrap algorithm. Using a bootstrap algorithm allows us to make inferences by treating the produced sample as if it is the population and calculating an empirical estimate from the statistic's sampling distribution by using Monte-Carlo sampling techniques (Abney (2002)). The algorithm, using a matrix and the number of bootstraps *n* (*per winning rate*) as an input, generates the mean of the strapped sample in a first step (i.e., *n* means per fixed winning rate). In a second step, the algorithm then calculates the mean of the calculated means in step 1.
.

```
# Generating a bootstrap function with number of straps and matrix of results
# as Input. Generates a Data.frame as Output, containing the average value of
# the sample's medians and the corresponding win rate.

bootstrap_problem_two <- function(n_straps, store){

    # generating a store matrix with 9 rows (one per win rate) and one column
    # per bootstrap, storing the median of the bootstrap sample
    avg_storer <- matrix(nrow = 9, ncol = n_straps)

    for (j in 1 : 9){ # 9 iterations (as we use 9 win rates)

      #bootstrap and calculate the mean of each bootstrap sample
      avg_storer[j,] <- unlist(mclapply(1 : n_straps, function(i){
      mean(store[j, sample(1 : dim(store)[2], size = dim(store)[2],
                           replace = TRUE)])
      }, mc.cores = 6, mc.set.seed = TRUE))
```

```
    }

    # generating a vector to store the median of each row
    substore <- rep(NA, 9)

    # calculating the median of each row of avg_storer
    for (i in 1 : 9) {
    substore[i] <- mean(avg_storer[i,])
    }

    # binding together the win rate and its corresponding average matches per
    # tournament
    output <- data.frame(Prob = seq(from = 0.1, to = 0.90, by = 0.1),
                         average_value = substore)

    return(output)
}
```

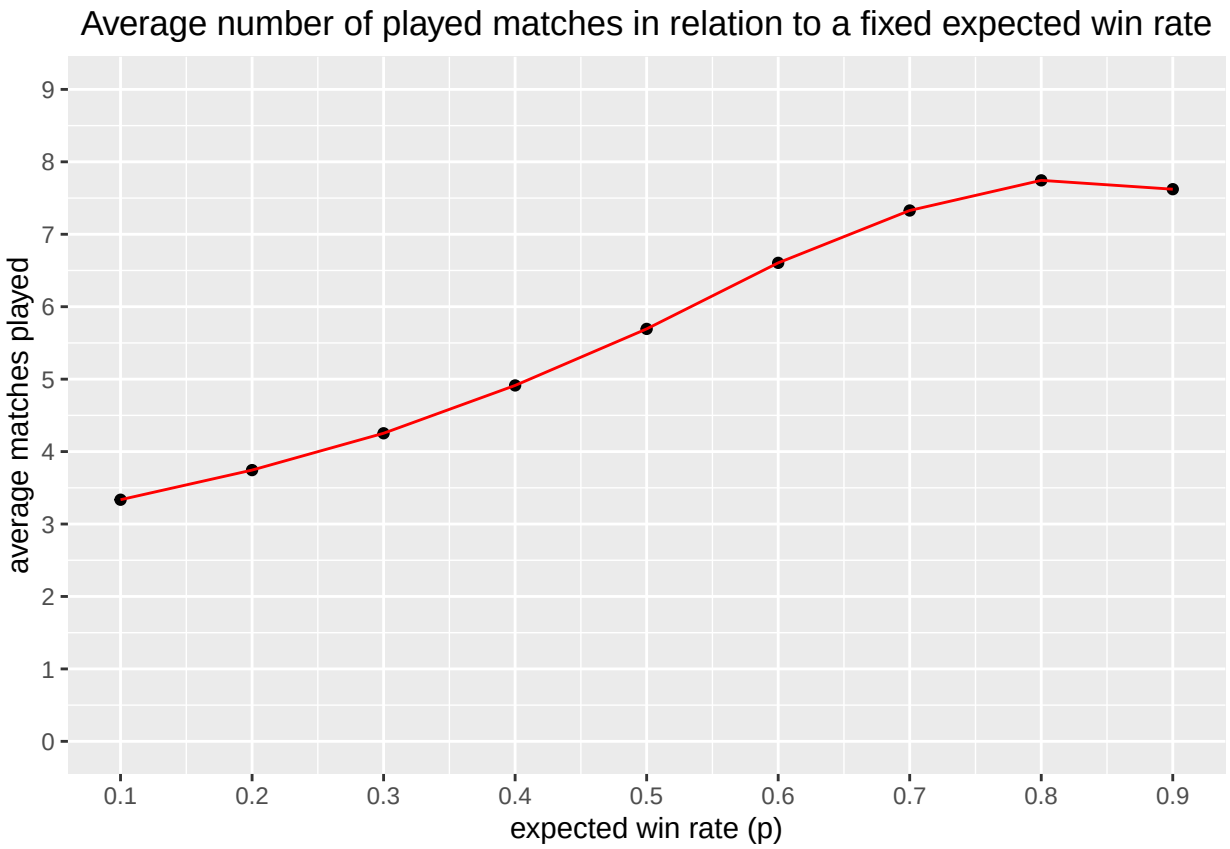
Using this algorithm on our matrix from above (*table_n_matches*) with $n = 1000$ bootstraps, we derive the following results:

```
RNGkind("L'Ecuyer-CMRG")
set.seed(0911)
# 1000 bootstraps
results_2.1 <- bootstrap_problem_two(1000, table_n_matches)

# calculating delta values
results_2.1$delta <- c(0, sapply(1:8, function(i) {
  results_2.1$average_value[i+1] - results_2.1$average_value[i]
}))

results_2.1 <- results_2.1 %>%
  rename(p = 1,
         matches_played = 2)

print(results_2.1)
```

```
##     p matches_played      delta
## 1 0.1       3.335595  0.0000000
## 2 0.2       3.744607  0.4090119
## 3 0.3       4.252899  0.5082922
## 4 0.4       4.913228  0.6603285
## 5 0.5       5.693612  0.7803842
## 6 0.6       6.605376  0.9117637
## 7 0.7       7.328171  0.7227947
## 8 0.8       7.744267  0.4160963
## 9 0.9       7.622093 -0.1221743
```

```
# plotting the results
results_2.1 %>%
  ggplot(aes(x = p, y = matches_played))+
  geom_point()+
```

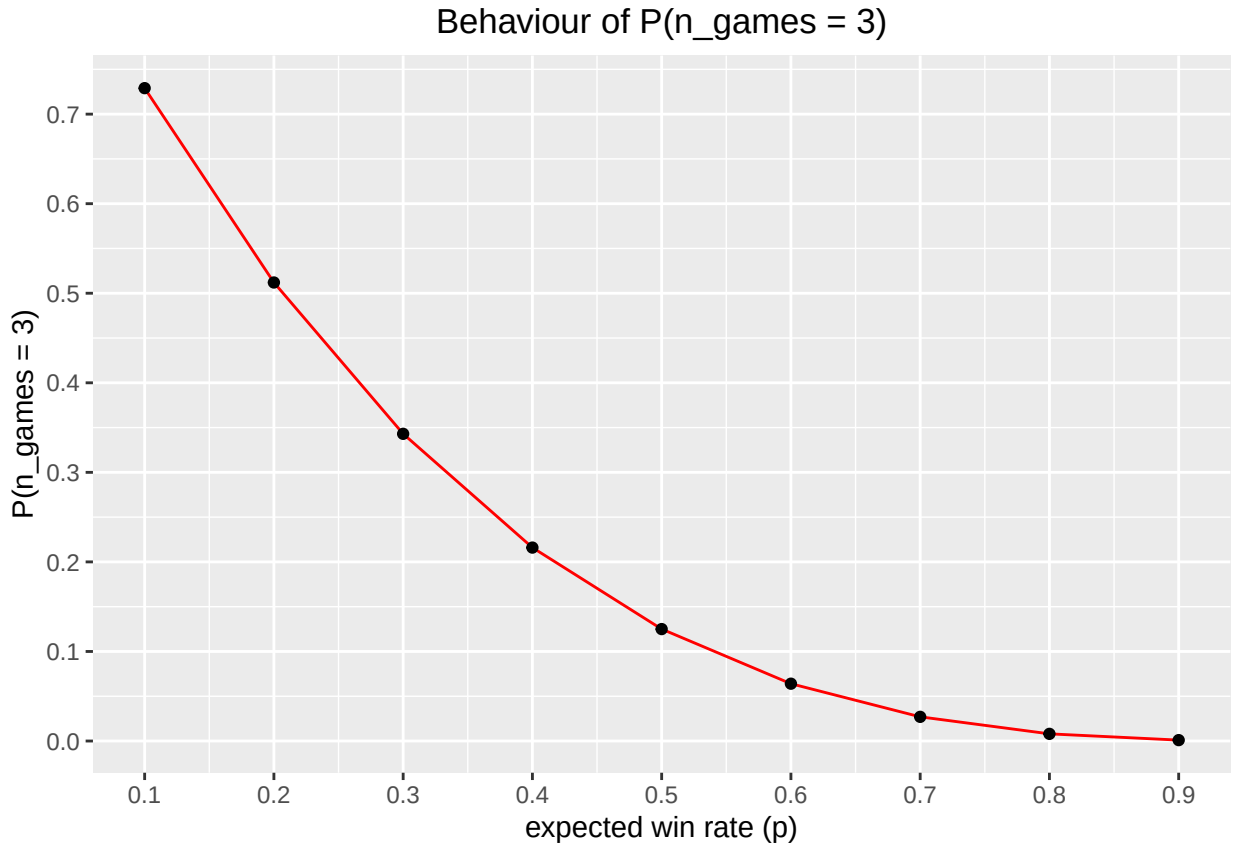
```
  geom_line(color = "red")+
  scale_x_continuous(name="expected win rate (p)", limits=c(0.1, 0.9),
                     n.breaks = 9)+
  scale_y_continuous(name="average matches played", limits=c(0, 9),
                     n.breaks = 9)+
  ggtitle("Average number of played matches in relation to a fixed expected win rate")+
  theme(plot.title = element_text(hjust= 0.5))
```



Analyzing the average number of matches played within a tournament as a function of p, we can observe an increase in the number of matches played from $p = 0.1$ until its peak at $p=0.8$ and afterwards decreasing. The reason lies in the format of the tournament itself. Due to the conditions of 7 wins or 3 losses, the tournament implies specific characteristics in terms of observable probabilities. In example, the probability for just playing the minimum amount of games, implying a team loses all of its first three games, can be expressed as $p(n_{games} = 3) = (1 - p)^3$, while winning *at least* one game can be expressed as $1 - p(n_{games} = 3)$. Assuming $p = 0.1$, $p(n_{games} = 3)$ is $(1 - 0.1)^3 = 0.729$, implying that in 73 out of 100 tournaments observed, no more than 3 games are played if $p = 10\%$. Thinking of $p(n_{games} = 3)$ as a *bottleneck*, the probability of losing the first three games decreases exponentially with increasing p.

```
p_loseall <- c() # generating empty vector

p_loseall <- sapply(win_rates, function(i){ # calculating Pr(n_games = 3)
  (1 - i)^3
  })

df_loseall <- data.frame(win_rates, p_loseall)
```

```r
# plotting
df_loseall %>%
  ggplot(aes(x = win_rates, y = p_loseall))+
  geom_line(color = "red")+
  geom_point()+
  scale_x_continuous(name="expected win rate (p)", limits=c(0.1, 0.9),
                     n.breaks = 9)+
  scale_y_continuous(name="P(n_games = 3)",
                     n.breaks = 9)+
  ggtitle("Behaviour of P(n_games = 3)")+
  theme(plot.title = element_text(hjust= 0.5))
```



This explains the growing Δ values observed, which are peaking at $p = 0.6$.
Another interesting factor is the peak of average played matches at $p=0.8$ and a negative Δ at 0.9. Thinking about the problem as a maximization problem of matches played per tournament, we want to know for which win rate p the probability of playing 9 matches in a tournament is the highest.We are looking at two functions: One for 7 wins and 2 losses ($f_1(p)$) and the other for 6 wins and 3 losses ($f_2(p)$):
$f_1(p) = \binom{9}{7}p^7(1-p)^{9-7}$
$f_2(p) = \binom{9}{2}(1-p)^2(1-(1-p))^{9-2}$

```r
# Using simple grid optimization
p <- seq(0.1, 0.95, 0.001) #vector with fixed win rates

y <- unlist(lapply(p, function(i){ # f(p)_1
  (choose(9,7) * i^7 * (1 - i)^2)
```

```
}))

index_max1 <- which.max(y) # index for maximum of f(p)_1
maximum_func1 <- p[index_max1]

z <- unlist(lapply(p, function(i){ # f(p)_2
  (choose(9,3) * (1-i)^3 * (1 - (1-i))^6)
}))

index_max2 <- which.max(z) # index for maximum of f(p)_2
maximum_func2 <- p[index_max2]

cat("maximum of function 1:", maximum_func1, "\n")
```

```
## maximum of function 1: 0.778
```

```
cat("maximum of function 2:", maximum_func2, "\n")
```

```
## maximum of function 2: 0.667
```

Using a simple grid search algorithm for obtaining the maximum of both $f_1(p)$ and $f_2(p)$, the functions peek at a p of 0.778 and 0.667, respectively. In conclusion, the likelihood of playing the maximum amount of matches in a tournament decreases with a $p > 0.78$. Therefore, we can observe a negative Δ at $p = 0.9$ and can generally assume negative Δ-values for $p > 0.9$.

The last part of the problem asks for the true win rates (observed win rates) of a team in comparison to the underlying expected win rate of the simulation. To solve this problem with computation, an appropriate algorithm is needed.
The algorithm below is an altered version of the *tournament_stats_ngames* algorithm. The only difference is that it now uses the third element of the output list of the original underlying algorithm *tournament_sim*, which is the proportion of matches won in a tournament.

```
tournament_stats_truewinrate <- function(n, rates = win_rates) {
  store <- matrix(nrow = length(rates), ncol=n)

for (i in 1 : length(rates)) {
  prob <- rates[i]

store[i,] <- unlist(mclapply(1:n, function(i){
  as.numeric(tournament_sim(p_win = prob)[3]) # output of tournament_sim is
  }, mc.cores = 6, mc.set.seed = TRUE)) # now true_winrate, as numeric
}
  return(store)
}
```

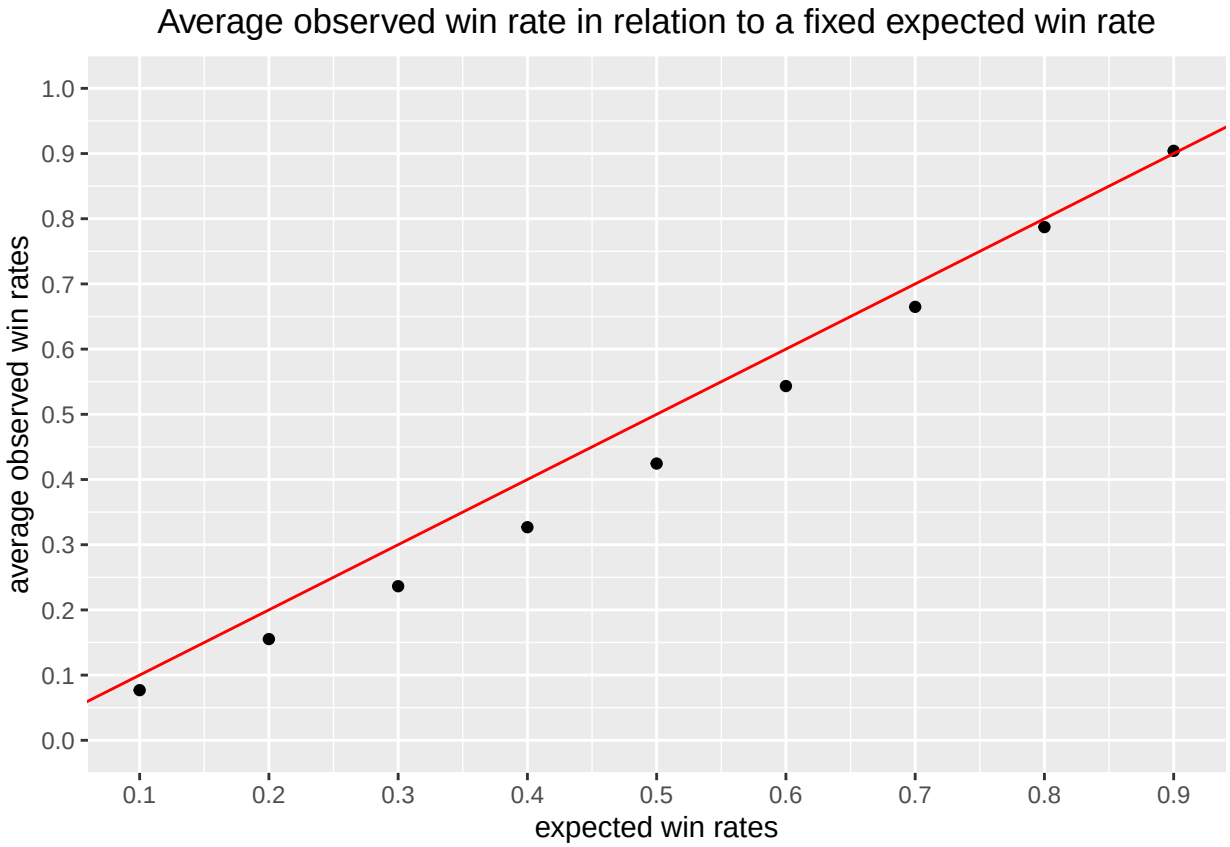
We again simulate 10,000 tournaments and use the bootstrap algorithm (1000 bootstrap-samples per expected win rate) for deriving reliable estimates.

```
RNGkind("L'Ecuyer-CMRG")
set.seed(0911)
# simulating 10k tournaments
true_w_rates <- tournament_stats_truewinrate(10000)
```

```r
# generating 1000 bootstraps
df_true_rates <- bootstrap_problem_two(n_straps = 1000, store = true_w_rates)
# calculating difference between observed vs. expected
df_true_rates$diff <- df_true_rates$average_value - df_true_rates$Prob


df_true_rates <- df_true_rates%>% #renaming
  rename(p = 1,
         win_rate_observed = 2,
         difference = 3)


print(df_true_rates)
```

```
##     p win_rate_observed   difference
## 1 0.1        0.07691215 -0.023087852
## 2 0.2        0.15524741 -0.044752586
## 3 0.3        0.23634399 -0.063656006
## 4 0.4        0.32686315 -0.073136854
## 5 0.5        0.42448927 -0.075510734
## 6 0.6        0.54334802 -0.056651978
## 7 0.7        0.66482399 -0.035176006
## 8 0.8        0.78717065 -0.012829350
## 9 0.9        0.90413220  0.004132198
```

```r
# plotting
df_true_rates %>%
  ggplot(aes(x = p, y = win_rate_observed))+
  geom_point()+
  scale_x_continuous(name="expected win rates", limits=c(0.1, 0.9),
                     n.breaks = 9)+
  scale_y_continuous(name="average observed win rates", limits=c(0.0, 1.0),
                     n.breaks = 11)+
  ggtitle("Average observed win rate in relation to a fixed expected win rate")+
  theme(plot.title = element_text(hjust= 0.5))+
  geom_abline(slope = 1, colour = "red")
```



The graphic above shows the average (*mean*) observed win rate per match in relation to its underlying fixed win rate. Although simulating 10,000 tournaments, we can observe a certain difference between the expected win rate per match and the actual observed counterpart.
This discrepancy is again due to the format of the tournament, i.e. the condition of 7 wins or 3 losses ending the tournament. As $p(n_{games} = 3) = 0.729$ for $p = 0.1$, we can observe a 0% win rate at 73 out of 100 tournaments with such p, implying a difference between the expected and the observed win rate of -0.1 each time.
The greatest difference between the expected and the average observed win rate can be seen around $p = 0.5$. If we apply the logic from the example above, the probability of no wins at all is $(1-0.5)^3 = 0.125$, implying a difference of -0.5 in 13 out of 100 tournament compared to the expected win rate, which then results in a the observable discrepance between the expected and the average observed win rate.

References:

Abney, Steven. 2002. "Bootstrapping." In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 360–67.

Britannica, The Editors of Encyclopaedia. 2020. ""Convergence"." *Encyclopedia Britannica*.

Dekking, Frederik Michel, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, and Ludolf Erwin Meester. 2005. *A Modern Introduction to Probability and Statistics: Understanding Why and How*. Vol. 488. Springer.

Liu, Marco. 2017. "Optimal Number of Trials for Monte Carlo Simulation." *VRC–Valuation Research Report*.