Assignment 4

Hu Tianao

2022-10-18

1. Probability theory

1.1 Q1

```
P <- function(event) {
  probabilities <- c(a = 0.5, b = 0.1, c = 0.4)

  if (length(event) == 1 && event %in% names(probabilities)) {
    return(probabilities[event])
  } else if (length(event) > 1 && all(event %in% names(probabilities))) {
    return(sum(probabilities[event]))
  } else {
    return(0)
  }
}

event_a <- "a"
event_b <- "b"
event_c <- "c"
event_ab <- c("a", "b")
event_bc <- c("b", "c")
event_ac <- c("a", "c")
event_abc <- c("a", "b", "c")
print(P(event_a))
```

```
##   a
## 0.5
```

```
print(P(event_b))
```

```
##   b
## 0.1
```

```
print(P(event_c))
```

```
##   c
## 0.4
```

```
print(P(event_ab))
```

```
## [1] 0.6
```

```
print(P(event_bc))
```

```
## [1] 0.5
```

```r
print(P(event_ac))
```

```
## [1] 0.9
```

```r
print(P(event_abc))
```

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

1.1 Q2

```r
Sample <- c(0, 1)
E <- list(
  empty_set = integer(0),   #
  event_0 = 0,   # {0}
  event_1 = 1,   # {1}
  event_01 = c(0, 1)   # {0, 1}
)

P <- function(event, q) {
  if (length(event) == 0) {
    return(0)
  } else if (identical(event, E$event_0)) {
    return(1 - q)
  } else if (identical(event, E$event_1)) {
    return(q)
  } else if (identical(event, E$event_01)) {
    return(1)
  } else {
    return(0)
  }
}

q <- 0.4   #      q
total_probability <- 0
#rule 1
for (event in E) {
  probability <- P(event, q)
  if (probability > 0 || probability == 0) {
    print('yes')
  }
}
```

```
## [1] "yes"
## [1] "yes"
## [1] "yes"
## [1] "yes"
```

```r
#rule 2
if (P(E$event_01, q) == 1) {
  print('yes')
}
```

```
## [1] "yes"
```

```r
#rule 3
for (i in 1:length(E)) {
```

```
  event_combinations <- combn(E, i)
  for (j in 1:ncol(event_combinations)) {
    events <- event_combinations[, j]
    probability_union <- P(unlist(events), q)
    probability_sum <- sum(sapply(events, function(event) P(event, q)))
    if (probability_union == probability_sum) {
      print("yes")
    }
  }
}
```

```
## [1] "yes"
## [1] "yes"
## [1] "yes"
## [1] "yes"
## [1] "yes"
## [1] "yes"
## [1] "yes"
## [1] "yes"
## [1] "yes"
```

2. Finite probability spaces

2.1 (Q1)

```
choose(8,3)
```

```
## [1] 56
```

$$f\left(k\right) = \binom{22}{z} 0.3^{z} \left(0.7\right)^{22-z} (\#eq: binom) \tag{1}$$

….

2.1 (Q2)

```
prob_red_spheres <- function(z) {
  p_red <- 0.3
  p_not_red <- 0.7
  probability <- choose(22, z) * (p_red^z) * (p_not_red^(22 - z))
  return(probability)
}

result <- prob_red_spheres(10)
print(result)
```

```
## [1] 0.05285129
```

2.1 (Q3)

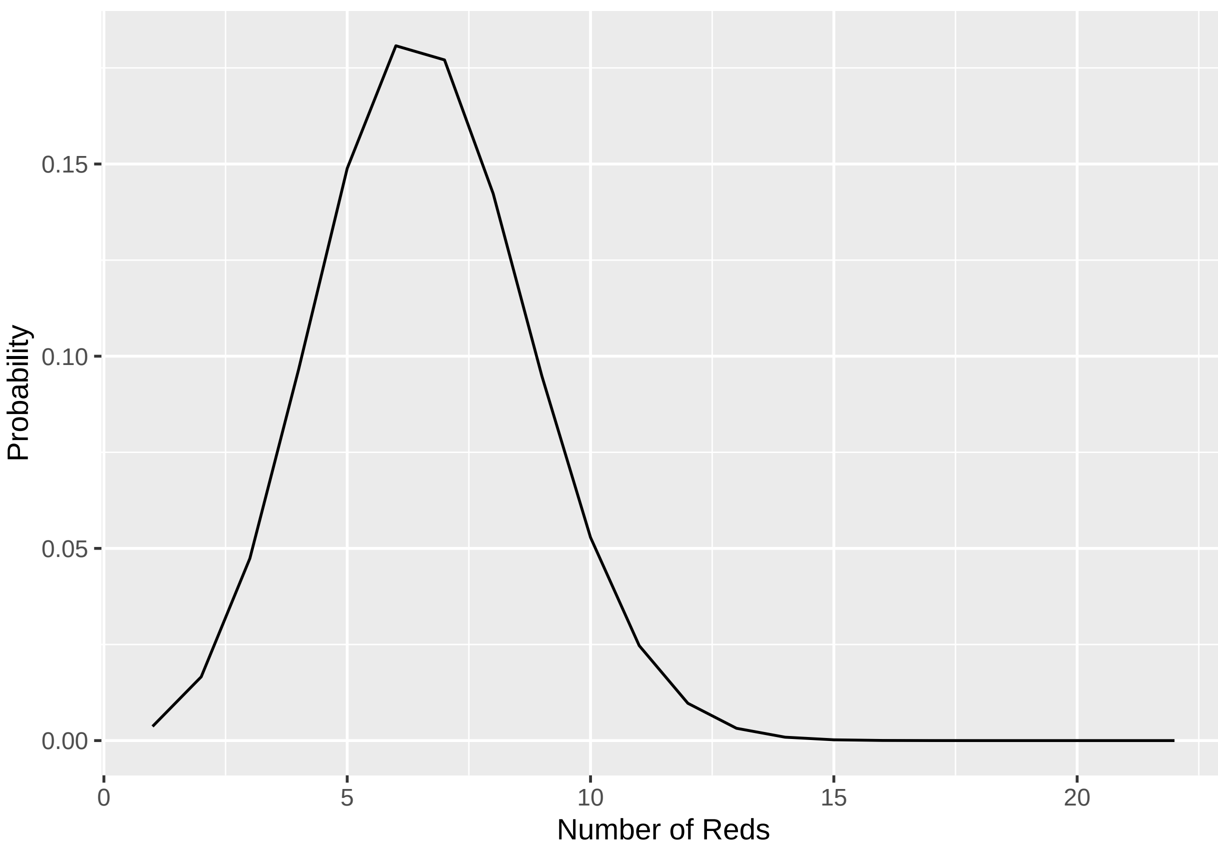
```
prob_by_num_reds <- data.frame(num_reds = numeric(0), prob = numeric(0))

for (z in 1:22) {
  probability <- prob_red_spheres(z)
```

```r
  prob_by_num_reds <- rbind(prob_by_num_reds, data.frame(num_reds = z, prob = probability))
}

head(prob_by_num_reds, 3)
```

```
##   num_reds        prob
## 1        1 0.003686403
## 2        2 0.016588812
## 3        3 0.047396606
```

2.1 (Q4)

```r
library(ggplot2)

ggplot(data = prob_by_num_reds, aes(x = num_reds, y = prob)) +
  geom_line() +
  labs(x = "Number of Reds", y = "Probability")
```



2.1 (Q4)

```r
sample(10, 22, replace=TRUE)
```

```
##  [1] 10  7  5  3  9  2  1 10 10  3  4  9  1  9  6  5  6  9  7  9 10  4
```

2.1 (Q5)

```r
## case 1: Setting the random seed just once
set.seed(0)
```

```r
for(i in 1:5){
print(sample(100,5,replace=FALSE))
# The result may well differ every time
}
```

```
## [1] 14 68 39  1 34
## [1] 87 43 14 82 59
## [1] 51 97 85 21 54
## [1] 74  7 73 79 85
## [1]  37  89 100  34  99
```

```r
## case 2: Resetting the random seed every time
set.seed(1)
print(sample(100,5,replace=FALSE))
```

```
## [1] 68 39  1 34 87
```

```r
set.seed(1)
print(sample(100,5,replace=FALSE))
```

```
## [1] 68 39  1 34 87
```

```r
set.seed(1)
print(sample(100,5,replace=FALSE))
```

```
## [1] 68 39  1 34 87
```

```r
# The result should not change
## case 3: reproducing case 1 if we set a random seed at the beginning.
set.seed(0)
for(i in 1:5){
print(sample(100,5,replace=FALSE))
} # The result will be 5 samples exactly the same as in case 1 (why?).
```

```
## [1] 14 68 39  1 34
## [1] 87 43 14 82 59
## [1] 51 97 85 21 54
## [1] 74  7 73 79 85
## [1]  37  89 100  34  99
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(purrr)
num_trials<-1000 # set the number of trials
set.seed(0) # set the random seed
sampling_with_replacement_simulation<-data.frame(trial=1:num_trials) %>%
mutate(sample_balls = map(.x=trial, ~sample(10,22, replace = TRUE)))
# generate collection of num_trials simulations
```

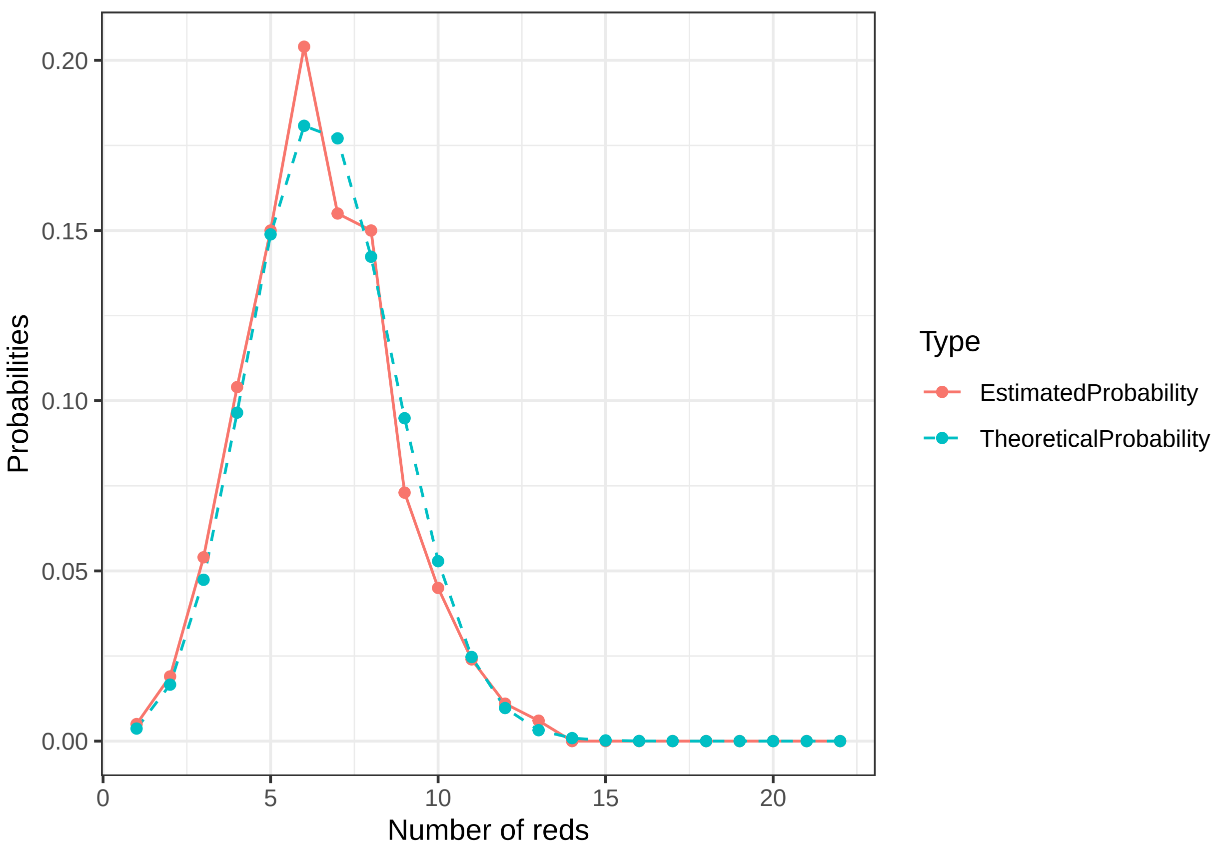
```r
# in the above code we have used "~" which defines an anonymous function

#   num_reds
sampling_with_replacement_simulation <- sampling_with_replacement_simulation %>%
  mutate(num_reds = map_dbl(sample_balls, ~sum(.x <= 3)))
```

2.1 (Q6)

```r
num_reds_in_simulation<-sampling_with_replacement_simulation %>%
pull(num_reds)
# we extract a vector corresponding to the number of reds in each trial
prob_by_num_reds<-prob_by_num_reds %>%
mutate(predicted_prob=map_dbl(.x=num_reds,~sum(num_reds_in_simulation==.x))/num_trials)
```

2.1 (Q7)

```r
library(tidyr)
prob_by_num_reds %>%
rename(TheoreticalProbability=prob, EstimatedProbability=predicted_prob) %>%
pivot_longer(cols=c("EstimatedProbability","TheoreticalProbability"),
names_to="Type",values_to="count") %>%
ggplot(aes(num_reds,count)) +
geom_line(aes(linetype=Type, color=Type)) + geom_point(aes(color=Type)) +
scale_linetype_manual(values = c("solid", "dashed"))+
theme_bw() + xlab("Number of reds") + ylab("Probabilities")
```



2.2 (Q1)

```r
# Load the necessary libraries
library(dplyr)
library(purrr)

# Set the random seed for reproducibility
set.seed(0)

# Specify the number of trials (simulations) and sample size
num_trials <- 10000  # You can increase this for more accuracy
sample_size <- 10

# Create a data frame to store the simulation results
simulation_results <- data.frame(trial = 1:num_trials) %>%
  mutate(
    # Generate samples of size 10 without replacement
    sample_result = map(.x = trial, ~sample(1:100, sample_size, replace = FALSE)),

    # Calculate the counts of red, green, and blue spheres in each sample
    red_count = map_dbl(sample_result, ~sum(.x <= 50)),
    green_count = map_dbl(sample_result, ~sum(.x > 50 & .x <= 80)),
    blue_count = map_dbl(sample_result, ~sum(.x > 80)),

    # Compute the minimum count among red, green, and blue
    min_count = pmin(red_count, green_count, blue_count),

    # Check if one or more colors are missing (minimum count is zero)
    missing_color = (min_count == 0)
  )

# Calculate the proportion of samples with missing colors
probability_missing_colors <- mean(simulation_results$missing_color)

# Print the probability
cat("Probability that one or more colors are missing:", probability_missing_colors, "\n")
```

```
## Probability that one or more colors are missing: 0.1145
```