HW03 - Solutions

Stat 133, Spring 2018, Prof. Sanchez

Due date: Sun Apr-01 (before midnight)

General Self-Grading Instructions

- Please read this document (answer key).
- You will have to enter your score and comments in the *Assignments Comments* section of the correspoing assignment on bCourses.
- Enter your own scores and comments for (every part) of every problem in the homework on a simple coarse scale:
  - **0%** = Didn't attempt or very very wrong,
  - **20%** = Got started and made some progress, but went off in the wrong direction or with no clear direction,
  - **50%** = Right direction and got half-way there,
  - **80%** = Mostly right but a minor thing missing or wrong,
  - **100%** = 100% correct.
- Also, enter your total score (out of an overall score of 110 points).
- If there is a cascading error, use a single deduction (don't double or triple or multiple penalize).
- Note: You must justify every self-grade score with a comment. If you are really confused about how to grade a particular problem, you should post on Piazza. This is not supposed to be a stressful process.
- We will accept late self-grades up to a week after the original homework deadline for half credit on the associated homework assignment.
- If you don't enter a proper grade by this deadline, you are giving yourself a zero on that assignment.
- Merely doing the homework is not enough, you must do the homework; turn it in on time; read the solutions; do the self-grade; and turn it in on time. Unless all of these steps are done, you will get a zero for that assignment.

1) File Structure (10 pts)

After completing this assignment, the file structure of your project should look like this. Although it was not required, you may have folders for `report/` and `data/` (this is okay).

```
hw03/
  README.md
  # report files
  hw03-first-last.Rmd
  hw03-first-last.md
  # data files
  LAC.csv
  experience-counts.txt
  gsw-height-weight.csv
  nba2017-roster.csv
  position-names.txt
  team-names.txt
  top10-salaries.csv
  code/
    binomial-functions.R
  images/
    ... # image files
```

2) Pipelines and Redirection (40 pts)

Use the `nba2017-roster.csv` data to perform the following tasks. All the commands have to be bash-shell commands (not R commands).

The solutions in the=is answer key show one way to perform the required tasks. As usual with coding, it is possible to achieve the same operations in different ways. As long as you have the right output, give full credit of 5 pts for each part.

2.1) Write a pipeline to obtain the unique team names, and redirect the output to a text file `team-names.txt`. Use `head` to display the first five lines of the created file (output shown belown).

```
# file with list of unique team names
cut -f 2 -d "," nba2017-roster.csv | tail +2 | sort -u > team-names.txt
head -n 5 team-names.txt
```

```
## "ATL"
## "BOS"
## "BRK"
## "CHI"
```

```
## "CHO"
```

2.2) Write a pipeline to obtain the unique positions, and redirect the output to a text file `position-names.txt`. Use `head` to display the first five lines of the created file (output shown below).

```
# file with list of unique positions
cut -f 3 -d "," nba2017-roster.csv | tail +2 | sort -u > position-names.txt
head -n 5 position-names.txt
```

```
## "C"
## "PF"
## "PG"
## "SF"
## "SG"
```

2.3) Write a pipeline to obtain the counts (i.e. frequencies) of the different `experience` values, displayed from largest to smallest (i.e. descending order). Redirect the output to a text file `experience-counts.txt`. Use `head` to display the first five lines of the created file (output shown below). The first column corresponds to count, the second column corresponds to experience.

```
# command broken down so that code fits in this page width
# frequencies of experience (descending order)
cut -f 7 -d "," nba2017-roster.csv | tail +2 | sort -n
| uniq -c | sort -n -r > experience-counts.txt

head -n 10 experience-counts.txt
```

```
# frequencies of experience (descending order)
cut -f 7 -d "," nba2017-roster.csv | tail +2 | sort -n | uniq -c | sort -n -r > experien
head -n 10 experience-counts.txt
```

```
##   80 0
##   52 1
##   46 2
##   36 3
##   35 4
##   30 8
##   29 5
##   27 6
##   26 7
##   18 9
```

2.4) Use output redirection commands to create a CSV file `LAC.csv` containing data for the `LAC` team (Los Angeles Clippers). Your CSV file should include column names. Use `cat` to display the content of the created file (output shown below).

```
# LAC file with header (i.e. column names)
head -n 1 nba2017-roster.csv > LAC.csv
grep "LAC" nba2017-roster.csv >> LAC.csv
cat LAC.csv
```

```
## "player","team","position","height","weight","age","experience","salary"
## "Alan Anderson","LAC","SF",78,220,34,7,1315448
## "Austin Rivers","LAC","SG",76,200,24,4,1.1e+07
## "Blake Griffin","LAC","PF",82,251,27,6,20140838
## "Brandon Bass","LAC","PF",80,250,31,11,1551659
## "Brice Johnson","LAC","PF",82,230,22,0,1273920
## "Chris Paul","LAC","PG",72,175,31,11,22868828
## "DeAndre Jordan","LAC","C",83,265,28,8,21165675
## "Diamond Stone","LAC","C",83,255,19,0,543471
## "J.J. Redick","LAC","SG",76,190,32,10,7377500
## "Jamal Crawford","LAC","SG",77,200,36,16,13253012
## "Luc Mbah a Moute","LAC","SF",80,230,30,8,2203000
## "Marreese Speights","LAC","C",82,255,29,8,1403611
## "Paul Pierce","LAC","SF",79,235,39,18,3500000
## "Raymond Felton","LAC","PG",73,205,32,11,1551659
## "Wesley Johnson","LAC","SF",79,215,29,6,5628000
```

2.5) Write a pipeline to display the age frequencies of `LAL` players. The first column corresponds to count, the second column corresponds to age.

```
# ages of LAL players
grep "LAL" nba2017-roster.csv | cut -f 6 -d "," | sort | uniq -c
```

```
##    2 19
##    1 20
##    2 22
##    3 24
##    2 25
##    2 30
##    2 31
##    1 37
```

2.6) Write a pipeline to find the number of players in `CLE` (Cleveland) team; the output should be just the number of players.

```
# how many players in CLE
grep "CLE" nba2017-roster.csv | wc -l
```

```
##       15
```

2.7) Write pipelines to create a CSV file `gsw-height-weight.csv` that contains the `player`, `height` and `weight` of `GSW` players. Your CSV file should include column names. Use `cat` to display the file contents:

```
# file: height and weight of GSW
cut -f 1,4-5 -d "," nba2017-roster.csv | head -n 1 > gsw-height-weight.csv
grep "GSW" nba2017-roster.csv | cut -f 1,4-5 -d "," >> gsw-height-weight.csv
cat gsw-height-weight.csv
```

```
## "player","height","weight"
## "Andre Iguodala",78,215
## "Damian Jones",84,245
## "David West",81,250
## "Draymond Green",79,230
## "Ian Clark",75,175
## "James Michael McAdoo",81,230
## "JaVale McGee",84,270
## "Kevin Durant",81,240
## "Kevon Looney",81,220
## "Klay Thompson",79,215
## "Matt Barnes",79,226
## "Patrick McCaw",79,185
## "Shaun Livingston",79,192
## "Stephen Curry",75,190
## "Zaza Pachulia",83,270
```

2.8) Write pipelines to create a file `top10-salaries.csv` containing the top10 player salaries, arranged by `salary` from largest to smallest. Your CSV file should include column names. Use `cat` to display the file contents:

```
# file: top10 player salaries
cut -f 1,8 -d "," nba2017-roster.csv | head -n 1 > top10-salaries.csv
cut -f 1,8 -d "," nba2017-roster.csv | tail +2 | sort -t "," -k 2 -n -r | head >> top10-
cat top10-salaries.csv
```

```
## "player","salary"
## "LeBron James",30963450
## "Russell Westbrook",26540100
## "Mike Conley",26540100
```

```
## "Kevin Durant",26540100
## "James Harden",26540100
## "DeMar DeRozan",26540100
## "Al Horford",26540100
## "Carmelo Anthony",24559380
## "Damian Lillard",24328425
## "Dwyane Wade",23200000
```

---

3) Binomial Probability Functions (50 pts)

You will have to write code implementing the functions listed below. Write the functions in an `R` script `binomial-functions.R`, and save it inside the `code/` folder. The script file should have a header with fields like title, and description. In addition, all the functions must have Roxygen comments.

Give 5 pts for each function correctly implemented

- `is_integer()`
- `is_positive()`
- `is_nonnegative()`
- `is_positive_integer()`
- `is_nonneg_integer()`
- `is_probability()`
- `bin_factorial()`
- `bin_combinations()`
- `bin_probability()`
- `bin_distribution()`

We don't expect that your functions have the exact same code of the solutions. However, your functions should return similar outputs.

Also, please take time to review the code of the solutions, and learn how to write code in a very compact when working with logical comparisons and conditionals.

R script `binomial-functions.R`

You will have to write code implementing the functions listed below. Write the functions in an `R` script `binomial-functions.R`, and save it inside the `code/` folder. The script file should have a header with fields like title, and description. In addition, all the functions must have Roxygen comments. For more information, refer to lab07.

Function `is_integer()`

Write a function `is_integer()` that tests if a numeric value can be considered to be an integer number (e.g. `2L` or `2`). This function should return `TRUE` if the input can be an integer, `FALSE` otherwise. *Hint:* the modulo operator `%%` is your friend (see `?'%%'` for more info). Assume that the input is always a single number.

```
#' @title Is Integer
#' @description tests whether a number is an integer
#' @param x numeric value
#' @return TRUE if input is valid, FALSE otherwise
is_integer <- function(x) {
  (x %% 1) == 0
}

# TRUE's
is_integer(-1)
```

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

```
is_integer(0)
```

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

```
is_integer(2L)
```

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

```
is_integer(2)
```

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

```
# FALSE's
is_integer(2.1)
```

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

```
is_integer(pi)
```

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

```
is_integer(0.01)
```

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

Function `is_positive()`

Write a function `is_positive()` that tests if a numeric value is a positive number. This function should return `TRUE` if the input is positive, `FALSE` otherwise. Assume that the input

is always a single number.

```
#' @title Is Positive
#' @description tests whether a number is positive
#' @param x numeric value
#' @return TRUE if input is positive, FALSE otherwise
is_positive <- function(x) {
  x > 0
}

# TRUE's
is_positive(0.01)
```

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

```
is_positive(2)
```

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

```
# FALSE's
is_positive(-2)
```

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

```
is_positive(0)
```

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

Function `is_nonnegative()`

Write a function `is_nonnegative()` that tests if a numeric value is a non-negative number. This function should return `TRUE` if the input is non-negative, `FALSE` otherwise. Assume that the input is always a single number.

```
#' @title Is Non-Negative
#' @description tests whether a number is non-negative
#' @param x numeric value
#' @return TRUE if input is non-negative, FALSE otherwise
is_nonnegative <- function(x) {
  x >= 0
}

# TRUE's
is_nonnegative(0)
```

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

```r
is_nonnegative(2)
```

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

```r
# FALSE's
is_nonnegative(-0.00001)
```

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

```r
is_nonnegative(-2)
```

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

Function `is_positive_integer()`

Use `is_positive()` and `is_integer()` to write a function `is_positive_integer()` that tests if a numeric value can be considered to be a positive integer. This function should return `TRUE` if the input is positive integer, `FALSE` otherwise. Assume that the input is always a single number.

```r
#' @title Is Positive Integer
#' @description tests whether a number is a positive integer
#' @param x numeric value
#' @return TRUE if input is positive integer, FALSE otherwise
is_positive_integer <- function(x) {
  (is_positive(x) & is_integer(x))
}

# TRUE
is_positive_integer(2)
```

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

```r
is_positive_integer(2L)
```

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

```r
# FALSE
is_positive_integer(0)
```

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

```r
is_positive_integer(-2)
```

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

Function `is_nonneg_integer()`

Use `is_nonnegative()` and `is_integer()` to write a function `is_nonneg_integer()` that tests if a numeric value can be considered to be a non-negative integer. This function should return `TRUE` if the input is non-negative integer, `FALSE` otherwise. Assume that the input is always a single number.

```
#' @title Is Non-negative Integer
#' @description tests whether a number is a non-negative integer
#' @param x numeric value
#' @return TRUE if input is non-negative integer, FALSE otherwise
is_nonneg_integer <- function(x) {
  (is_nonnegative(x) & is_integer(x))
}

# TRUE's
is_nonneg_integer(0)
```

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

```
is_nonneg_integer(1)
```

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

```
# FALSE
is_nonneg_integer(-1)
```

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

```
is_nonneg_integer(-2.5)
```

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

Function `is_probability()`

Write a function `is_probability()` that tests if a given number p is a valid probability value: $0 \leq p \leq 1$. This function should return `TRUE` if the input is a valid probability, `FALSE` otherwise. Assume that the input is always a single number.

```
#' @title Is probability
#' @description checks whether a number is a valid probability value
#' @param x numeric value
#' @return TRUE if input is valid, otherwise FALSE
is_probability <- function(x) {
  (is_nonnegative(x) & x <= 1)
}
```

```r
is_probability(0)
```

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

```r
is_probability(0.5)
```

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

```r
is_probability(1)
```

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

```r
is_probability(-1)
```

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

```r
is_probability(1.1)
```

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

Function `bin_factorial()`

Use a `for` loop to write a function `bin_factorial()` that calculates the factorial of a non-negative integer n.

```r
#' @title Factorial
#' @description Computes factorial of a positive integer
#' @param x numeric input
#' @return factorial
bin_factorial <- function(x) {
  if (x == 0) {
    return(1)
  } else {
    product <- 1
    for (i in 1:x) {
      product <- product * i
    }
    return(product)
  }
}

bin_factorial(0)
```

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

```r
bin_factorial(1)
```

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

```r
bin_factorial(2)
```

```
## [1] 2
```

```r
bin_factorial(3)
```

```
## [1] 6
```

```r
bin_factorial(4)
```

```
## [1] 24
```

```r
# valid
bin_factorial(5)
```

```
## [1] 120
```

```r
bin_factorial(0)
```

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

Function `bin_combinations()`

Use `bin_factorial()` to write a function `bin_combinations()` that calculates the number of combinations in which k successes can occur in n trials. Your function should have arguments `n` and `k`.

```r
#' @title Combinations
#' @description Compute number of combinations
#' @param n number of trials
#' @param k number of successes
#' @return number of combinations
bin_combinations <- function(n, k) {
  numerator <- bin_factorial(n)
  denominator <- bin_factorial(k) * bin_factorial(n-k)
  return(numerator / denominator)
}

bin_combinations(n = 5, k = 2)
```

```
## [1] 10
```

```r
bin_combinations(10, 3)
```

```
## [1] 120
```

```r
bin_combinations(4, 4)
```

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

Function `bin_probability()`

Use your functions `is_nonneg_integer()`, `is_probability()`, and `bin_combinations()` to create a `bin_probability()` function. Your function should have arguments `trials`, `success`, and `prob`.

Use `is_nonneg_integer()` to check that `trials` and `success` are valid non-integer numbers. If any of `trials` or `success` is invalid, then `bin_probability()` should raise an error (triggered by `stop()`). Likewise, use `is_probability()` to test that `prob` is a valid probability value. If `prob` is invalid, then `bin_probability()` should `stop()` execution with an error.

```
#' @title Binomial Probability
#' @description Computes the binomial probability
#' @param trials number of trials
#' @param success number of successes
#' @param prob probability of success
#' @return probability value
bin_probability <- function(trials, success, prob) {
  # check inputs
  if (!is_positive_integer(trials)) {
    stop('invalid trials value')
  }
  if (!is_nonneg_integer(success)) {
    stop('invalid success value')
  }
  if (!is_probability(prob)) {
    stop('invalid probability value')
  }
  n <- trials
  k <- success
  p <- prob
  bin_combinations(n, k) * p^k * ((1-p)^(n-k))
}

bin_probability(5, 2, 0.5)
```

```
## [1] 0.3125
```

```
bin_probability(5, 4, 0.8)
```

```
## [1] 0.4096
```

```
# probability of getting 2 successes in 5 trials
# (assuming prob of success = 0.5)
bin_probability(trials = 5, success = 2, prob = 0.5)
```

```
## [1] 0.3125
```

Function `bin_distribution()`

Use `bin_probability()` to create a `bin_distribution()` function. Your function should have arguments `trials`, and `prob`. This function should return a data frame with the probability distribution:

```
#' @title Binomial Distribution
#' @description Computes binomial distribution
#' @param trials number of trials
#' @param prob probability
#' @return data frame with the probability distribution
bin_distribution <- function(trials, prob) {
  n <- trials
  p <- prob
  probs <- rep(0, n+1)
  for (k in 0:n) {
    probs[k+1] <- bin_probability(n, k, p)
  }
  distrib <- data.frame(
    success = 0:n,
    probability = probs
  )
  return(distrib)
}

bin_distribution(10, 0.3)
```

```
##    success  probability
## 1        0 0.0282475249
## 2        1 0.1210608210
## 3        2 0.2334744405
## 4        3 0.2668279320
## 5        4 0.2001209490
## 6        5 0.1029193452
## 7        6 0.0367569090
## 8        7 0.0090016920
## 9        8 0.0014467005
## 10       9 0.0001377810
## 11      10 0.0000059049
```

```
# binomial probability distribution
bin_distribution(trials = 5, prob = 0.5)
```

```
##   success probability
## 1       0     0.03125
## 2       1     0.15625
```

```
## 3       2     0.31250
## 4       3     0.31250
## 5       4     0.15625
## 6       5     0.03125
```

`Rmd` file (10 pts)

In addition to including the bash-shell commands, your Rmd file should source your `binomial-functions.R` script: use `source()` to import your `R` script in your `Rmd` file. **Use a relative path!** (don't set absolute directories).

In your `Rmd` file report, write code to carry out the following computations

- Assume that the "successful" event is getting a "six" when rolling a die. Consider rolling a fair die 10 times. Use `bin_probability()` to find the probability of getting exactly 3 sixes.

```
bin_probability(trials = 10, success = 3, prob = 1/6)
```

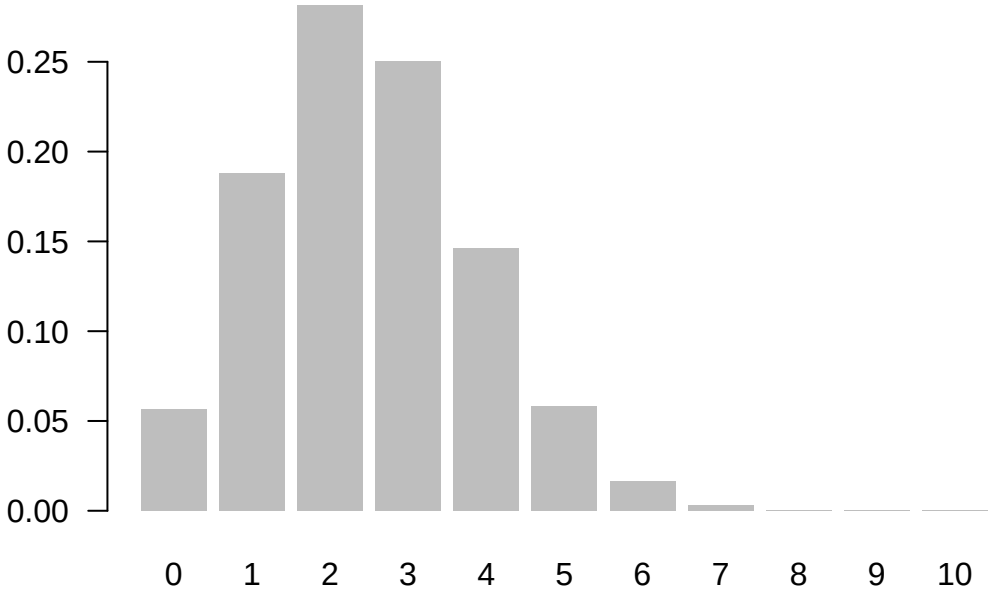
```
## [1] 0.1550454
```

- Use `bin_distribution()` to obtain the distribution of the number of "sixes" when rolling a loaded die 10 times, in which the number "six" has probability of 0.25. Make a plot of this distribution.

```
distrib1 <- bin_distribution(trials = 10, prob = 0.25)
distrib1
```

```
##    success  probability
## 1        0 5.631351e-02
## 2        1 1.877117e-01
## 3        2 2.815676e-01
## 4        3 2.502823e-01
## 5        4 1.459980e-01
## 6        5 5.839920e-02
## 7        6 1.622200e-02
## 8        7 3.089905e-03
## 9        8 3.862381e-04
## 10       9 2.861023e-05
## 11      10 9.536743e-07
```

```
barplot(distrib1$probability, border = NA, las = 1,
        names.arg = distrib1$success)
```



- Use `bin_probability()`, and a `for` loop, to obtain the probability of getting more than 3 heads in 5 tosses with a biased coin of 35% chance of heads.

```
heads <- c(4, 5)

# greater than 3 heads
gt3heads <- 0
for (i in heads) {
  gt3heads <- gt3heads + bin_probability(trials = 10, success = i, prob = 0.35)
}
gt3heads
```

```
## [1] 0.3912389
```

- Use `bin_distribution()` to obtain the probability distribution of the number of heads when tossing a loaded coin 15 times, with 35% chance of heads. Make a plot of this distribution.

```
distrib2 <- bin_distribution(trials = 15, prob = 0.35)
distrib2
```

```
##    success  probability
## 1        0 1.562069e-03
## 2        1 1.261672e-02
## 3        2 4.755531e-02
## 4        3 1.109624e-01
## 5        4 1.792469e-01
## 6        5 2.123387e-01
## 7        6 1.905604e-01
## 8        7 1.319264e-01
```

```
## 9         8 7.103729e-02
## 10        9 2.975066e-02
## 11       10 9.611752e-03
## 12       11 2.352527e-03
## 13       12 4.222484e-04
## 14       13 5.246873e-05
## 15       14 4.036056e-06
## 16       15 1.448841e-07
```

```
barplot(distrib2$probability, border = NA, las = 1,
        names.arg = distrib2$success, cex.names = 0.7)
```