Problem Set 2

Zimo Shu

Problem 1 - Modified Random walk

a. Implement the random walk in three versions.

Based on the rule, a single walk can only be: +1, -1, +10, -3.

Probabilities:

$P(+1) = 0.5 \cdot 0.95 = 0.475$, $P(-1) = 0.5 \cdot 0.8 = 0.40$, $P(+10) = 0.5 \cdot 0.05 = 0.025$, $P(-3) = 0.5 \cdot 0.2 = 0.10$.

(At first I was so not sure about the definition here so I reflected on the problem again. "If the +10 is chosen, it's not +1 then +10, it is just +1." –> If +1 is chosen, it could either be +1 or +10.)

```r
steps <- c(10, 1, -1, -3)
probabilities <- c(0.025, 0.475, 0.40, 0.10)
```

Version 1: using a loop.

```r
random_walk1 <- function(n) {
  start_pos <- 0 # start position at 0
  for (i in 1:n) {
    start_pos <- start_pos + sample(steps, 1, prob = probabilities, replace = TRUE)
  }
  start_pos
}
```

Version 2: using built-in R vectorized functions. (Using no loops.) (Hint: Does the order of steps matter?)

```r
random_walk2 <- function(n) {
  sum(sample(steps, n, replace = TRUE, prob = probabilities))
}
```

Version 3: Implement the random walk using one of the "apply" functions.

```r
random_walk3 <- function(n) {
  sum(vapply(seq_len(n), function(i) sample(steps, 1, prob = probabilities, replace = TRUE),
}
```

```r
# Demonstrate that all versions work by running the following

random_walk1(10)
```

```
[1] 9
```

```r
random_walk2(10)
```

```
[1] -8
```

```r
random_walk3(10)
```

```
[1] 6
```

```r
random_walk1(1000)
```

```
[1] 21
```

```r
random_walk2(1000)
```

```
[1] 21
```

```r
random_walk3(1000)
```

```
[1] -78
```

b. Demonstrate that the three versions can give the same result. Show this for both n=10 and n=1000. (You will need to add a way to control the randomization.)

```r
# set the same seed to control
# for n = 10
set.seed(506)
random_walk1(10)
```

```
[1] -8
```

```r
set.seed(506)
random_walk2(10)
```

```
[1] -8
```

```r
set.seed(506)
random_walk3(10)
```

```
[1] -8
```

```r
# for n = 10000
set.seed(321)
random_walk1(10000)
```

```
[1] 202
```

```r
set.seed(321)
random_walk2(10000)
```

```
[1] 202
```

```r
set.seed(321)
random_walk3(10000)
```

```
[1] 202
```

c. Use the microbenchmark package to clearly demonstrate the speed of the implementations. Compare performance with a low input (1,000) and a large input (100,000). Discuss the results.

```r
library(microbenchmark)
```

```
Warning: package 'microbenchmark' was built under R version 4.3.3
```

```r
perform_func <- function(n, times = 50) {
  microbenchmark(
    loop  = random_walk1(n),
    apply = random_walk3(n),
    vect  = random_walk2(n),
    times = times
  )
}

bench_low <- perform_func(1000, 100)  # low input
bench_large <- perform_func(100000, 100)  # large input
bench_low
```

```
Unit: microseconds
  expr      min       lq       mean   median        uq       max neval cld
  loop 4542.593 5301.276 5991.33397 5618.202 5931.4195 16765.147   100 a
 apply 5473.722 5904.203 6594.10655 6119.053 6737.1790 11595.122   100  b
  vect   25.815   27.414   30.15635   29.771   31.6105    54.782   100   c
```

```r
bench_large
```

```
Unit: milliseconds
  expr        min         lq       mean   median         uq        max neval
  loop 587.460210 628.980043 656.977458 650.0085 672.589971 816.275280   100
 apply 615.900724 681.994970 711.825918 707.0139 729.341648 881.515425   100
  vect   1.926122   2.233964   2.331749   2.3176   2.384133   2.860121   100
 cld
 a
  b
   c
```

Based on the outputs, in both low and large input cases, we could see that vectorized function (version 2) typically has the least running time, and fastest speed. Apply is the slowest. Loop is slightly faster than apply at these sizes.

We could also see the mean and median time of large inputs are way smaller than the mean and median time of low inputs, which is true for all three versions.

d. What is the probability that the random walk ends at 0 if the number of steps is 10? 100? 1000? Defend your answers with evidence based upon a Monte Carlo simulation.

```
end_zero <- function(n, R = 1e6, seed = 123) {
  set.seed(seed)
  # Multinomial(n, p)
  X <- rmultinom(R, size = n, prob = probabilities)
  # final positions
  Final <- drop(steps %*% X)
  phat <- mean(Final == 0)     # Monte Carlo estimate
  se   <- sqrt(phat * (1 - phat) / R)
  ci   <- phat + qnorm(c(.025, .975)) * se
  list(p = phat, se = se, ci = ci)
}

res10 <- end_zero(10, R = 1e6, seed = 1)
res100 <- end_zero(100, R = 1e6, seed = 1)
res1000 <- end_zero(1000, R = 1e6, seed = 1)

res10
```

```
$p
[1] 0.132197

$se
[1] 0.0003387048

$ci
[1] 0.1315332 0.1328608
```

```
res100
```

```
$p
[1] 0.019751

$se
[1] 0.0001391434

$ci
[1] 0.01947828 0.02002372
```

```r
res1000
```

```
$p
[1] 0.0058

$se
[1] 7.593655e-05

$ci
[1] 0.005651167 0.005948833
```

We could tell that the probability of ending at 0 decreases with n. When n = 10, the probability is approximately 0.132. When n = 100, the probability is approximately 0.0197. When n = 10, the probability is approximately 0.0058.

Problem 2 - Mean of Mixture of Distributions

The number of cars passing an intersection is a classic example of a Poisson distribution. At a particular intersection, Poisson is an appropriate distribution most of the time, but during rush hours (hours of 8am and 5pm) the distribution is really normally distributed with a much higher mean.

Using a Monte Carlo simulation, estimate the average number of cars that pass an intersection per day under the following assumptions:

From midnight until 7 AM, the distribution of cars per hour is Poisson with mean 1. From 9am to 4pm, the distribution of cars per hour is Poisson with mean 8. From 6pm to 11pm, the distribution of cars per hour is Poisson with mean 12. During rush hours (8am and 5pm), the distribution of cars per hour is Normal with mean 60 and variance 12 Accomplish this without using any loops.

```r
hour <- 0:23
rush_hours <- c(8, 17)                # 8am and 5pm
normal_hours <- numeric(24)
normal_hours[hour %in% 0:7]   <- 1          # midnight until 7am
normal_hours[hour %in% 9:16]  <- 8          # 9am to 4pm
normal_hours[hour %in% 18:23] <- 12         # 6pm to 11pm

# Monte Carlo
avg_cars <- function(R = 2e5, seed = 1) {
  set.seed(seed)
  pois_hours <- setdiff(hour, rush_hours)
```

```r
  # Poisson
  X_pois <- matrix(rpois(length(pois_hours) * R, normal_hours[pois_hours + 1]), nrow = lengt
  # Rush hours
  X_norm <- matrix(rnorm(2 * R, mean = 60, sd = sqrt(12)), nrow = 2)

  # Daily totals
  daily_totals <- colSums(X_pois) + colSums(X_norm)
  # Monte Carlo estimate
  estimation <- mean(daily_totals)
  se  <- sd(daily_totals) / sqrt(R)
  ci  <- estimation + qnorm(c(.025, .975)) * se
  list(estimate = estimation, se = se, ci95 = ci)
}
# I got an error in using colsums: colSums(pmax(0, round(rbind(X_pois, X_norm)))) : 'x' must
avg_cars()
```

```
$estimate
[1] 263.9933

$se
[1] 0.02894918

$ci95
[1] 263.9366 264.0501
```

Hence, the average number of cars is approximately 264 (263.99).

Problem 3 - Linear Regression

```r
youtube <- read.csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/da
```

a.

```r
# Take a look at the table at first
head(youtube)
```

```
  year     brand
1 2018    Toyota
2 2020 Bud Light
```

```
3 2006 Bud Light
4 2018   Hynudai
5 2003 Bud Light
6 2020    Toyota
                                                superbowl_ads_dot_com_url
1                             https://superbowl-ads.com/good-odds-toyota/
2 https://superbowl-ads.com/2020-bud-light-seltzer-inside-posts-brain/
3                 https://superbowl-ads.com/2006-bud-light-bear-attack/
4  https://superbowl-ads.com/hope-detector-nfl-super-bowl-lii-hyundai/
5                 https://superbowl-ads.com/2003-bud-light-hermit-crab/
6 https://superbowl-ads.com/2020-toyota-go-places-with-cobie-smulders/
                                  youtube_url funny show_product_quickly
1 https://www.youtube.com/watch?v=zeBZvwYQ-hA FALSE                FALSE
2 https://www.youtube.com/watch?v=nbbp0VW7z8w  TRUE                 TRUE
3 https://www.youtube.com/watch?v=yk0MQD5YgV8  TRUE                FALSE
4 https://www.youtube.com/watch?v=lNPccrGk77A FALSE                 TRUE
5 https://www.youtube.com/watch?v=ovQYgnXHooY  TRUE                 TRUE
6 https://www.youtube.com/watch?v=f34Ji70u3nk  TRUE                 TRUE
  patriotic celebrity danger animals use_sex          id          kind
1     FALSE     FALSE  FALSE   FALSE   FALSE zeBZvwYQ-hA youtube#video
2     FALSE      TRUE   TRUE   FALSE   FALSE nbbp0VW7z8w youtube#video
3     FALSE     FALSE   TRUE    TRUE   FALSE yk0MQD5YgV8 youtube#video
4     FALSE     FALSE  FALSE   FALSE   FALSE lNPccrGk77A youtube#video
5     FALSE     FALSE   TRUE    TRUE    TRUE ovQYgnXHooY youtube#video
6     FALSE      TRUE   TRUE    TRUE   FALSE f34Ji70u3nk youtube#video
                         etag view_count like_count dislike_count
1 rn-ggKNly38Cl0C3CNjNnUH9xUw     173929       1233            38
2 1roDoK-SYqSpqYwKbYrMH0jEJQ4      47752        485            14
3 0HiDfHTB3kilXfN8W0VTH0nwUIg     142310        129            15
4 G9Dhby9Xe1UpnfcIrHmcnZYRCFI        198          2             0
5 acsuYsFFQ_gHCC060Wus0tTgLjM      13741         20             3
6 _UAuS0SlCytmE2NGd6u5pfaYhnA      23636        115            11
  favorite_count comment_count         published_at
1              0            NA 2018-02-03T11:29:14Z
2              0            14 2020-01-31T21:04:13Z
3              0             9 2006-02-06T10:02:36Z
4              0             0 2018-03-09T15:40:18Z
5              0             2 2006-07-18T04:53:42Z
6              0            13 2020-02-02T21:21:27Z
                                                    title
1            Toyota Super Bowl Commercial 2018 Good Odds
2  Bud Light: Post Malone #PostyStore Inside Post's Brain
3             Super Bowl 2006: Bud Light "Save Yourself"
```

```
4                           Hyundai / Hope Detector (2018)
5                                         bud light pick up
6 Toyota Super Bowl Commercial 2020 Cobie Smulders Heroes

1
2 Bud Light, Post Malone "#PostyStore Inside Post's Brain"\n\nGarrick Sheldon (Copywriting, 2
3
4
5
6
                                          thumbnail     channel_title
1 https://i.ytimg.com/vi/zeBZvwYQ-hA/sddefault.jpg Funny Commercials
2 https://i.ytimg.com/vi/nbbp0VW7z8w/sddefault.jpg   VCU Brandcenter
3                                              <NA>      John Keehler
4                                              <NA>         IATSE 490
5                                              <NA>          jassymei
6 https://i.ytimg.com/vi/f34Ji70u3nk/sddefault.jpg Funny Commercials
  category_id
1           1
2          27
3          17
4          22
5          24
6           1
```

```r
# Remove any column that might uniquely identify a commercial
drop_cols <- c(
  "brand",
  "superbowl_ads_dot_com_url", "youtube_url", "thumbnail",
  "channel_title",
  "published_at",
  "id", "kind", "etag",
  "title", "description"
)

youtube_cleaned <- youtube[ , !(names(youtube) %in% drop_cols)]

# report dimensions
dim(youtube_cleaned)
```

```
[1] 247  14
```

The dimensional now is as above.

b.

```
vars <- c("view_count","like_count","dislike_count","favorite_count","comment_count")
# Examine the distribution
summ <- sapply(youtube[vars], \(x) c(
  n      = sum(!is.na(x)),
  min    = min(x, na.rm=TRUE),
  q25    = unname(quantile(x, .25, na.rm=TRUE)),
  median = median(x, na.rm=TRUE),
  mean   = mean(x, na.rm=TRUE),
  q75    = unname(quantile(x, .75, na.rm=TRUE)),
  max    = max(x, na.rm=TRUE),
  sd     = sd(x, na.rm=TRUE),
  p_zeros= mean(x == 0, na.rm=TRUE)
))
round(t(summ), 3)
```

```
                 n min  q25 median        mean       q75       max           sd
view_count     231  10 6431  41379 1407556.459 170015.50 176373378 11971111.010
like_count     225   0   19    130    4146.031    527.00    275362    23920.403
dislike_count  225   0    1      7     833.538     24.00     92990     6948.522
favorite_count 231   0    0      0       0.000      0.00         0        0.000
comment_count  222   0    1     10     188.640     50.75      9190      986.457
               p_zeros
view_count       0.000
like_count       0.040
dislike_count    0.209
favorite_count   1.000
comment_count    0.185
```

From the outputs, we could see:

view_count: (ii) right-skewed, should be transformed

like_count: right-skewed, outliers, should be transformed

dislike_count: skewed, outliers, should be transformed

favorite_count: sd is zero (variance is zero), **not appropriate**

comment_count: right-skewed, should be transformed.

```r
# Transformation
youtube$view_count_log     <- log1p(youtube$view_count)
youtube$like_count_log     <- log1p(youtube$like_count)
youtube$dislike_count_log  <- log1p(youtube$dislike_count)
youtube$comment_count_log  <- log1p(youtube$comment_count)
```

c. For each variable in part b. that are appropriate, fit a linear regression model predicting them based upon each of the seven binary flags for characteristics of the ads, such as whether it is funny. Control for year as a continuous covariate.

Discuss the results. Identify the direction of any statistically significant results.

```r
# Seven binary flags
flags <- c("funny","show_product_quickly","patriotic",
                "celebrity","danger","animals","use_sex")
flags <- flags[flags %in% names(youtube)]
```

```r
# Fit models and print outputs
fit_models <- function(y) {
  formulas <- as.formula(paste(y, "~ year +", paste(flags, collapse = " + ")))
  dat <- youtube[, c(y, "year", flags)]
  dat <- dat[complete.cases(dat), ]
  mod <- lm(formulas, data = dat)
  cat("\n============================\n")
  cat("Outcome:", y, "\n")
  cat("============================\n")
  print(summary(mod)$coefficients)
  invisible(mod)
}

mods <- lapply(c("view_count_log","like_count_log","dislike_count_log","comment_count_log"),
```

```
============================
Outcome: view_count_log
============================
                           Estimate  Std. Error    t value  Pr(>|t|)
(Intercept)             -31.55015804 71.00537527 -0.4443348 0.6572334
year                      0.02053399  0.03530599  0.5816007 0.5614258
funnyTRUE                 0.56492445  0.46702107  1.2096338 0.2277060
show_product_quicklyTRUE  0.21088918  0.40530215  0.5203258 0.6033550
patrioticTRUE             0.50699051  0.53811043  0.9421681 0.3471307
```

```
celebrityTRUE             0.03547862  0.42228051  0.0840167 0.9331189
dangerTRUE                0.63131085  0.41812403  1.5098650 0.1324998
animalsTRUE              -0.31001838  0.39347937 -0.7878898 0.4316016
use_sexTRUE              -0.38670726  0.44781782 -0.8635370 0.3887743

============================
Outcome: like_count_log
============================
                              Estimate  Std. Error    t value   Pr(>|t|)
(Intercept)              -150.51356902 63.45723424 -2.3718898 0.01857564
year                        0.07684959  0.03155325  2.4355517 0.01567989
funnyTRUE                   0.47476026  0.41816138  1.1353518 0.25748620
show_product_quicklyTRUE    0.20017362  0.36390920  0.5500647 0.58284333
patrioticTRUE               0.80688663  0.49790889  1.6205507 0.10657295
celebrityTRUE               0.41283478  0.38212296  1.0803716 0.28118156
dangerTRUE                  0.63894992  0.37350269  1.7106970 0.08857278
animalsTRUE                -0.05943769  0.35298000 -0.1683883 0.86643540
use_sexTRUE                -0.42951949  0.40063789 -1.0720890 0.28487641

============================
Outcome: dislike_count_log
============================
                              Estimate  Std. Error    t value     Pr(>|t|)
(Intercept)              -183.06812777 53.34768465 -3.4316040 0.0007189054
year                        0.09207348  0.02652642  3.4710107 0.0006260350
funnyTRUE                   0.25949148  0.35154292  0.7381502 0.4612243850
show_product_quicklyTRUE    0.27511424  0.30593381  0.8992607 0.3695152223
patrioticTRUE               0.81407206  0.41858564  1.9448160 0.0530952460
celebrityTRUE              -0.20214335  0.32124588 -0.6292481 0.5298516125
dangerTRUE                  0.22184469  0.31399893  0.7065142 0.4806298632
animalsTRUE                -0.21191815  0.29674576 -0.7141405 0.4759113338
use_sexTRUE                -0.32980132  0.33681115 -0.9791877 0.3285826970

============================
Outcome: comment_count_log
============================
                             Estimate  Std. Error    t value   Pr(>|t|)
(Intercept)              -99.09835037 52.92351239 -1.8724825 0.06251009
year                       0.05033845  0.02631887  1.9126376 0.05713584
funnyTRUE                  0.21954414  0.34527939  0.6358449 0.52556010
show_product_quicklyTRUE   0.40938620  0.30229342  1.3542676 0.17708596
patrioticTRUE              0.66698069  0.39901933  1.6715498 0.09608156
celebrityTRUE              0.29767273  0.31541291  0.9437557 0.34636374
```

```
dangerTRUE                   0.17783936  0.31068999  0.5724013 0.56765378
animalsTRUE                 -0.26802431  0.29346559 -0.9133074 0.36211349
use_sexTRUE                 -0.39323227  0.33162665 -1.1857680 0.23703504
```

From the outputs, we could see that for view_count_log, all $p > 0.13$, so no predictors are significant. For like_count_log, year is positive and significant. All flags are not significant. For dislike_count_log, year is positive and significant. Flag patriotic is very close to 0.05, and the direction is positive. All others are not significant. For comment_count_log, no predictors are really significant. Flag year is very close to 0.05 though.

Hence, after log-transforming, there is no sufficient evidence that shows that the seven flags (funny, show product quickly, patriotic, celebrity, danger, animals, use sex) are statistically associated with views, likes, dislikes, or comments at the 0.05 level. (We could see for like_count_log and dislike_count_log, the variable year is positive and significant, which indicates that year is significantly positive for likes and dislikes.)

d. Consider only the outcome of view counts. Calculate $\hat{\beta}$ manually (without using lm) by first creating a proper design matrix, then using matrix algebra to estimate β. Confirm that you get the same result as lm did in part c.

```r
fml_viewcount <- as.formula(
  paste("view_count_log ~ year +", paste(flags, collapse = " + "))
)

vars_new <- c("view_count_log","year", flags)
dat <- youtube[complete.cases(youtube[, vars_new]), vars_new]

# design matrix X and response y
X <- model.matrix(fml_viewcount, data = dat)
y <- dat$view_count_log

# Use normal equations
XtX <- t(X) %*% X
Xty <- t(X) %*% y
beta_hat <- solve(XtX, Xty)

# Use lm to get the value
fit <- lm(fml_viewcount, data = dat)
coef_lm <- coefficients(fit)

print(beta_hat)
```

```
                                [,1]
(Intercept)              -31.55015804
year                       0.02053399
funnyTRUE                  0.56492445
show_product_quicklyTRUE   0.21088918
patrioticTRUE              0.50699051
celebrityTRUE              0.03547862
dangerTRUE                 0.63131085
animalsTRUE               -0.31001838
use_sexTRUE               -0.38670726
```

```r
print(coef_lm)
```

```
             (Intercept)                     year                funnyTRUE
            -31.55015804               0.02053399               0.56492445
show_product_quicklyTRUE            patrioticTRUE            celebrityTRUE
              0.21088918               0.50699051               0.03547862
              dangerTRUE              animalsTRUE              use_sexTRUE
              0.63131085              -0.31001838              -0.38670726
```

```r
all.equal(as.vector(beta_hat), as.vector(coef_lm))
```

```
[1] TRUE
```

Hence, I got the same result as lm did. The calculated result would be:
$\hat{\beta} \approx (-31.55, 0.02, 0.56, 0.21, 0.51, 0.04, 0.63, -0.31, -0.38)$.