CMDA-3654

Homework 2

Michael La Vance

Due as a .pdf upload

Problem 1: (40 pts) Learning to write functions in R.

Part 1: Recall the Pythagorean theorem:

$$a^2 + b^2 = c^2$$

- Write a function called `pythagorean()` that, given the lengths of two sides of the triangle, calculates the length of the third side.
- This function should be flexible - that is, the function works if I give it values for `a` and `b`, or `b` and `c`, or `a` and `c`.
- If the user only provides the length of one side, the function should throw an error with `stop()` (read about the `stop()` function). Likewise, if the user provides the lengths of all three sides, the function should throw an error.
- If the user provides any values other than numeric values, the function should throw an error.
- Negative values don't make sense, so you should also throw an error.

```
pythagorean <- function( a = NA, b = NA , c = NA ){
  #temp
  #which(c(a,b,c) != 0) != 2 | a < 0 | b < 0 | c < 0
  temp <- c(a,b,c)
  if (sum(is.na(temp)) != 1 | min(temp, na.rm = TRUE) < 0){
    stop()
  }
  final <- 0
  if ( is.na(c)){
    c <- ( (a**2 + b**2)**.5)
  } else if (is.na(a)){
    a <- ((c**2 - b**2)**.5)
  } else if (is.na(b)){
    b <- ((c**2-a**2)**.5)
  }
  return(list("a" = a, "b" = b, "c" = c))

}
```

Part 2: Now write a new function called `pythagoreans_revenge()` that can take in the following data frame (or similar data frames):

```
prob4df <- data.frame(
  a = c(2.5,  4.3,   NA, NA, -2, 4),
  b = c(8.7,   NA,  5.9, NA, 5,  4),
  c = c( NA, 12.4,  7.7,  5, NA, 4)
)

prob4df
```

```
     a   b    c
1  2.5 8.7   NA
2  4.3  NA 12.4
3   NA 5.9  7.7
4   NA  NA  5.0
5 -2.0 5.0   NA
6  4.0 4.0  4.0
```

- This time we want to fill in the missing values with the appropriate value that satisfies the Pythagorean theorem.
- This time don't have the function stop when there is a problem. Instead create a new character array or factor to add onto the data frame that has the following categories:
  - `"okay"` - it took the two values and computed the missing value without any problem;
  - `"has negative"` - one or more of the values was negative;
  - `"too many missing"` - two values were missing, hence we cannot find them;

- "incorrect math" - all the values were provided, but they don't satisfy the theorem.

```
pythagoreans_revenge <- function(a = c(NA,NA,NA), b = c(NA,NA,NA), c = c(NA,NA,NA)){
  #assuming all data frames are the same size and all vectors are provided
  reasons <- 1:length(a)
  for (i in 1:length(a)){
    #welcome to if else hell
    if (min(c(a[i],b[i],c[i]), na.rm = TRUE) < 0){
      reasons[i] = "has negative"
    } else {
      temp <- sum(is.na(a[i]),is.na(b[i]),is.na(c[i]))
      if( temp == 0 ){
        if (a[i]**2 + b[i]**2 == c[i]**2){
          #correct
          reasons[i] = "okay"
        } else {
          #incorrect
          reasons[i] = "incorrect math"
        }
      }
      else if (temp == 1){
        answers <- pythagorean(a[i],b[i],c[i])
        #okay
        a[i] <-answers$a
        b[i] <-answers$b
        c[i] <-answers$c
        reasons[i] = "okay"
      } else if (temp > 1){
        #too many missing values
        reasons[i] = "too many missing"
      }
    }

  }
  #return statement
  return(list("a" = a, "b" = b, "c" = c, "reasons" = reasons))
}
```

Problem 2: (40 pts) Riemann Sums with R

In this problem, we will write our own function to perform numerical integration via Riemann sums - a refresher of this concept can be found here.

- Consider the function $f(x) = 2e^{-2x}$ (notice, this is the probability density function for an exponential distribution with $\lambda = 2$, but that's not important here). Use midpoint Riemann sums to approximate $P(0 \leq x \leq 1) = \int_0^1 f(x)dx$
  - For a specified number of segments n, your function should compute the approximate probability and compare with the exact answer by displaying the exact answer, and showing the error (you can determine the exact answer however you like, it can be hardcoded).
  - It should have an option called: `make_plot = FALSE` where `FALSE` is it's default value. When the option is `TRUE`, then your function should plot $f(x) = 2e^{-2x}$ on the specified interval `[0, 1]`, as well as the rectangles used to approximate the area. **Hint: look into using the `rect()` and `lines()` functions.**
  - Finally, it should return the run-time (how long it took to do everything) in seconds.

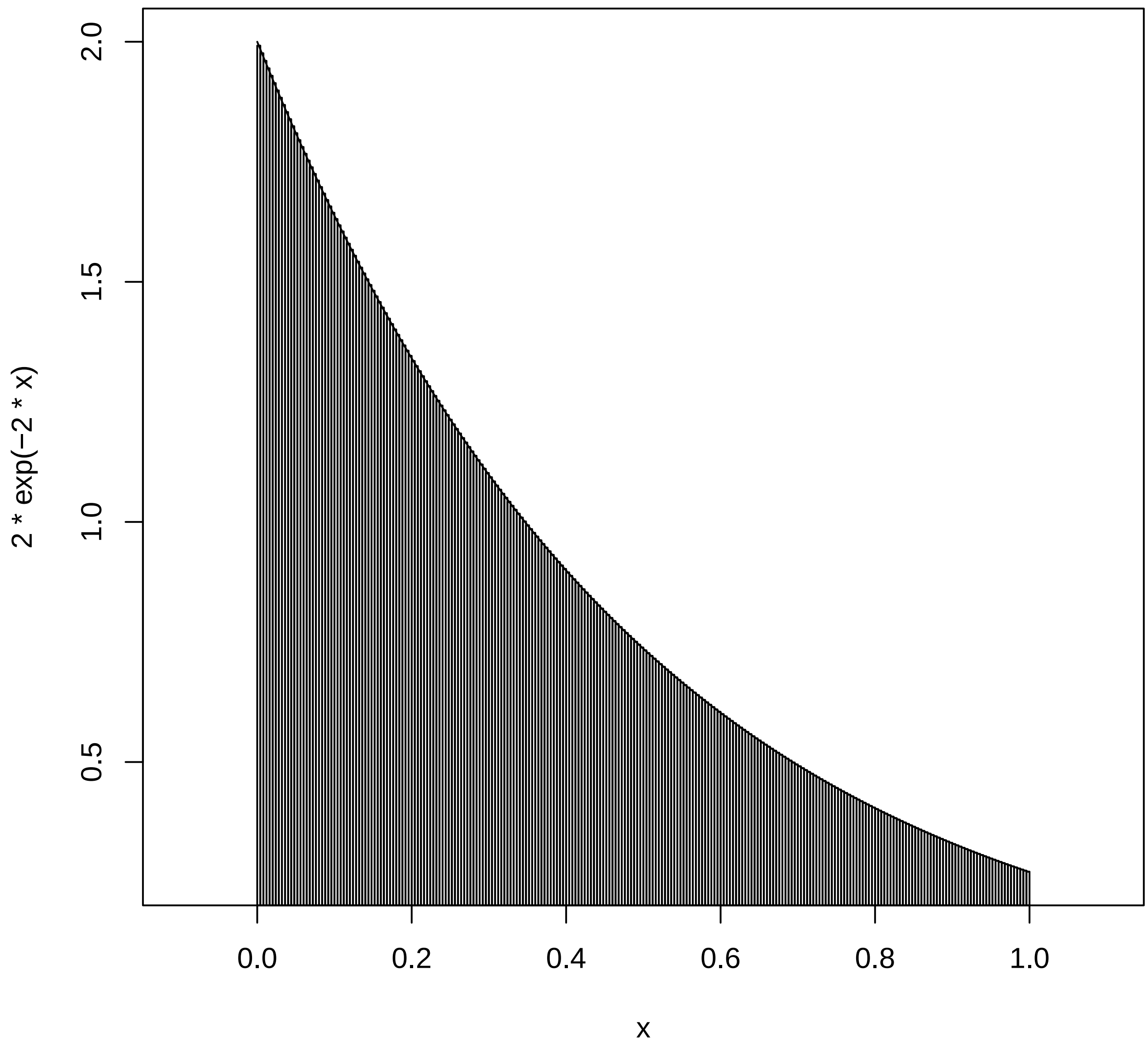
a. For `n = 250`, what is your approximate answer and how does it compare to the exact answer? How long did your function take to run? Go ahead provide plot (`make_plot = TRUE`) for this situation.

b. For `n = 500`, what is your approximate answer and how does it compare to the exact answer? How long did your function take to run?

c. Use the function you just wrote to estimate the probability for `n <- c( seq(10, 90, by = 10), seq(100, 1000, by = 100))`, make sure you save all of the answers.

d. Create a scatterplot with connecting lines for the estimate of the probability versus `n`. Use the function `abline()` to overlay a red line that shows the exact answer.

```
eq <- function(x){
  return(2*exp(-2*x))
}

riemann <- function(first = 0, ending = 1, n = 250, make_plot = FALSE){
  #hard coding exact solution
  start.time = Sys.time()
  solution <- 0.8646647167633873
  #code
  displacement <- (ending - first)/(n)
  numbers <- seq(first, ending, displacement)
  off <- displacement/2
  numbers <- head(numbers,-1)

  #only saving the values if make plot is true
  if(make_plot){
    fin <- 0
    #making initial graph
    curve(2*exp(-2*x), 0, 1, xlim = c(-.1,1.1))
    #for loop using rect to add to plot as calculated
    fin <- 0
    for (i in numbers){
      temp <- eq(i + off)
      fin <- fin + displacement*temp
      rect(i,0,i+2*off,temp)

    }
  } else {
    fin <- 0
    for ( i in numbers){
      fin <- fin + displacement*eq(i+off)
    }

  }

  #return statement
  end.time = Sys.time()
  ttotal <-end.time - start.time
  return(list("Approximation" = fin , "Difference" = solution - fin , "time" = ttotal))
}

riemann(0,1,250, TRUE)
```



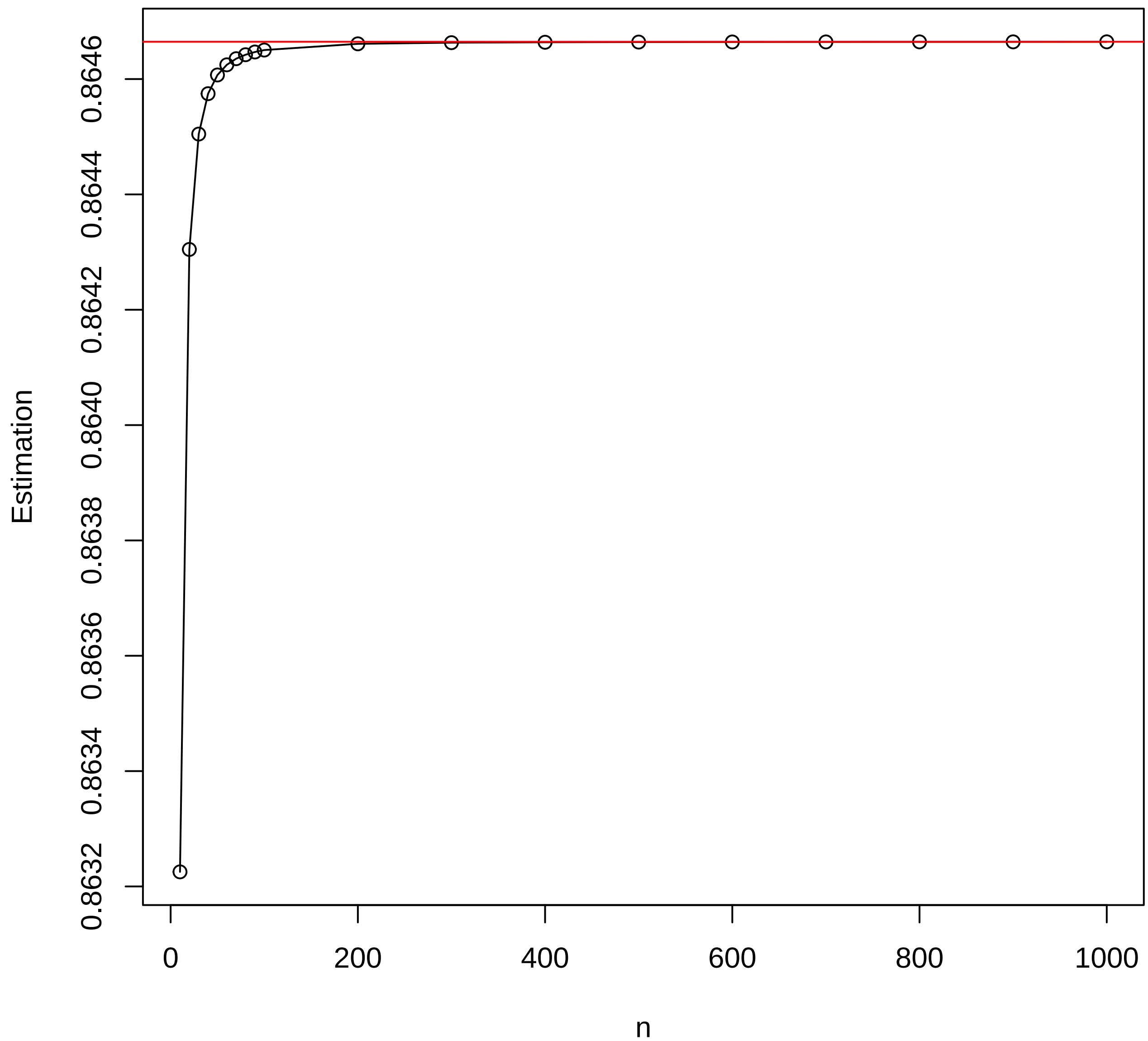

```
$Approximation
[1] 0.8646624

$Difference
[1] 2.305768e-06

$time
Time difference of 0.02293921 secs
```

```
riemann(0,1,500)
```

```
$Approximation
[1] 0.8646641

$Difference
[1] 5.764429e-07

$time
```

```
Time difference of 0.0009980202 secs
```

```
n <- c( seq(10, 90, by = 10), seq(100, 1000, by = 100))
answers <- 1:length(n)
for(i in answers){
  answers[i] <- riemann(0,1,n[i])$Approximation
}

#uhhhhhh
plot(n,answers, ylab = "Estimation")
lines(n,answers)
abline(0.8646647167633873,0,col = "Red")
```



Problem 3: (20 pts) More function writing with R

A common concept in machine learning is one of cross-validation. Don't worry about what it is or why it's important just yet, this topic will be covered in more detail in the future. The basic idea of the setup is to take a dataset with n observations, and randomly partition it into k equally sized subsets. Here, you will implement your own function to split a dataset into k folds.

- Write a function called `k.fold.partition` that takes in any data set (but demonstrate by passing in `mtcars`), shuffles the rows, and splits it into $k = 4$ folds, and returns a list (with components named `fold1`, `fold2`, etc) of the folds. Essentially, return an `R` object containing these smaller dataframes.

```r
#function
k.fold.partition <- function(df){
  #randomizing
  df1 <- df[sample(nrow(df)),]
  #I know this is over coded but it works
  sizes <- 1:4
  for( i in sizes){
    if (i < nrow(df)%%4){
      sizes[i] = floor(nrow(df)/4)+1
    } else {
      sizes[i] = floor(nrow(df)/4)
    }
  }
  #could have made a function specifically for this but I didn't think about it until I finished writing it ou
  return(list("fold1" = df1[1:sizes[1],],"fold2" = df1[sizes[1]:sum(sizes[1:2]),], "fold3" = df1[sum(sizes[1:2

}

#demonstration
mtcars
```

```
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

```
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
temp <- k.fold.partition(mtcars)

temp$fold1
```

```
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Merc 240D        24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Camaro Z28       13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Mazda RX4        21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Lotus Europa     30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Merc 280         19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
AMC Javelin      15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Ferrari Dino     19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
```

```r
temp$fold2
```

```
                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Porsche 914-2      26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Dodge Challenger   15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
Toyota Corolla     33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Honda Civic        30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Volvo 142E         21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
```

```r
temp$fold3
```

```
                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Fiat 128           32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
```

```r
temp$fold4
```

```
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
```
