Homework 6

James Young

7/5/2019

There are six exercises below. You are required to provide solutions for at least four of the five. You are required to solve at least one exercise in R, and at least one in SAS. You are required to provide five solutions, each solution will be worth 10 points. Thus, you may choose to provide both R and SAS solutions for a single exercise, or you may solve five of the sixth problems, mixing the languages as you wish.

## Reuse

For many of these exercises, you may be able to reuse functions written in prior homework. Define those functions here. I’m also including data vectors that can be used in some exercises.

CaloriesPerServingMean <- c(268.1, 271.1, 280.9, 294.7, 285.6, 288.6, 384.4)  
CaloriesPerServingSD <- c(124.8, 124.2, 116.2, 117.7, 118.3, 122.0, 168.3)

# Exercise 1 Please Grade

Write a general Cohen function to be more generally useful, accepting a wider range of arguments. For convenience, name this general.d.

The new function should accept two parameters, m, s

In your function, check for these condititions:

* If m is of length 1 and s is length 1, then simply divide m/s - that is, proceed with the calculations as if m = %Diff and s = CV.
* If m is of length 2, then calculate the difference and proceed with the calculations.
* If m is of length greater than 2, find the difference between the min and max of m and proceed with the calculations.
* If s is of length greater than 1 calculate pooled sd as

general.d <-function(m,s){  
m.len = length(m)  
s.len = length(s)  
spooled = (((sum(s\*\*2))/2)^(1/2))  
 if (m.len > 2){print((abs(min(m)-max(m)))/spooled)  
 }  
 else if (m.len <- 2) { print((max(m)-min(m))/s)  
 } else if (m.len <- 1) {  
 print(m / s)  
 } else {print(m/s)}  
}

Test your function with the three cases:

min(CaloriesPerServingMean)-max(CaloriesPerServingMean)

## [1] -116.3

#### 1936 Effect Size

Use just the mean from 1936 and the associated standard deviation

general.d(268.1, 124.8)

#### 1936 versus 2006

This should duplicate results from prior exercises.

general.d(c(268.1,384.4), c(124.8,168.3))  
#first calc is diff second calc is cohen.d

#### All of Calories per Serving.

Use the vectors from the Reuse chunk.

general.d(CaloriesPerServingMean,CaloriesPerServingSD)

Are these three scenarios sufficient to test every path through general.d?

Yes.

# Exercise 3 Please Grade

Calculate a cumulative probability value from the normal pdf, using the Newton-Cotes formula

where are a sequence of evenly spaced numbers from , with , is the number of in the sequence and step size .

We will calculate this integral by calculating successive approximations of = norm.pdf over series of with increasingly smaller step sizes.

## Part a.

Calculate by summing over $L(\bf{X\_0})$, where is a series from incremented by . Multiply this sum by for an approximate

Think of this as the sum of a series of rectangles, each wide and a height given by the normal pdf.

x.seq = data.frame(seq(from=(-2), to=(2), by = 0.1))  
x.sum = sum(abs(x.seq))  
x.sum = x.sum \* 0.1  
x.sum

## [1] 4.2

## Part b.

Create a second series by setting . Compute from this series as in part a. Let You now have the are of twice as many rectangles as part a, but each is half as wide.

x.seq = data.frame(seq(from=(-2), to=(2), by = 0.05))  
x.sum = sum(abs(x.seq))  
x.sum = x.sum \* 0.05  
x.sum

## [1] 4.1

## Part c.

Compute . If , your sequence of iterations has converged on a solution for . Finish with Part d. Otherwise, increment , let . Create the next series and compute the next .

Hint: code this first as a for loop of a small number of until you know your code will converge toward a solution.

i = 1  
h.3 = 0.1  
for(i in 1:20){  
 h.3 = h.3/2  
 x.seq = data.frame(seq(from=(-2), to=(2), by = h.3))  
   
 x.sum3 = sum(x.seq)  
 x.by.h3 = x.sum3 \* h.3  
   
   
 i=i+1  
   
}

# Part d

Report , and .

print(i)

## [1] 21

n = nrow(x.seq)  
print(n)

## [1] 41943041

h = h.3  
print(h)

## [1] 9.536743e-08

To check your results, compare your final to

L\_i = x.by.h3  
  
L\_i - pnorm(-2, lower.tail = TRUE)-pnorm(-2, lower.tail = TRUE)

## [1] -0.04550026

Is your estimate within of this value? No, my estimate is not within 0.0001 of this value.

You might find it useful to produce staircase plots for the first 2-4 iterations (plot vs on one graph). You might also find it interesting to plot or versus or . You can create vectors to hold the intermediate steps - 10 iterations should be enough. How many iterations might it take to get within 0.000001 of the expected value from R?

# Exercise 4 Please Grade

## Part a.

Write a function to compute mean, standard deviation, skewness and kurtosis from a single vector of numeric values. You can use the built-in mean function, but must use one (and only one) for loop to compute the rest. Be sure to include a check for missing values. Note that computationally efficient implementations of moments calculations take advantage of , etc.

See <https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm> for formula for skewness and kurtosis. This reference gives several definitions for both skewness and kurtosis, you only need to implement one formula for each. Note that for computing skewness and kurtosis, standard deviation is computed using as a divisor, not .

Your function should return a list with Mean, SD, Skewness and Kurtosis. If you use IML, you will need to implement this as a subroutie and use call by reference; include these variables in parameter list.

data = rnorm(1000, mean=10, sd=1)  
  
summarydata= function(data){  
 data.mean = mean(data, na.rm = TRUE)  
 data = data[!is.na(data)]  
 data.len = length(data)  
   
 data.sd.nu = 0  
 data.sk.nu = 0  
 data.kur.nu = 0  
   
   
 for(i in 1:data.len){  
 data.sd.nu = ((data[[i]]-data.mean)\*\*2)+data.sd.nu  
 data.sk.nu = ((data[[i]]-data.mean)\*\*3)+data.sk.nu  
 data.kur.nu = ((data[[i]]-data.mean)\*\*4)+data.kur.nu  
 }  
   
 data.sd = sqrt(data.sd.nu/(data.len-1))  
 data.sk = sqrt(data.sk.nu/data.len)/(data.sd\*\*3)  
 data.kur = sqrt(data.kur.nu/data.len)/(data.sd\*\*4)  
 print(data.kur)   
 print(data.mean)  
 print(data.sd)  
 print(data.sk)  
   
}  
  
summarydata(data)

## [1] 1.883885  
## [1] 10.03969  
## [1] 0.9525783  
## [1] 0.06207013

## Part b.

Test your function by computing moments for Price from pumpkins.csv, for ELO from elo.csv or the combine observations from SiRstvt. If find that ELO shows both skewness and kurtosis, Price is kurtotic but not skewed, while SiRstvt are approximately normal.

setwd("C://data/")  
pumpkins <- read.csv("pumpkins.csv")  
x = summarydata(pumpkins$Price)

## Warning in sqrt(data.sk.nu/data.len): NaNs produced

## [1] 0.0007424466  
## [1] 178.0667  
## [1] 41.02391  
## [1] NaN

x

## [1] NaN

If you wish, compare your function results with the skewness and kurtosis in the moments package. This package also implements test of significance for skewness and kurtosis.

library(moments)  
kurtosis(pumpkins$Price)

# Exercise 6 Please Grade

Write an improved Poisson pmf function, call this function smart.pois, using the same parameters x and lamba as before, but check for the following conditions. 1. If is negative, return a missing value (NA, .). 2. If is non-integer, truncate then proceed. 3. If is too large for the factorial function, return the smallest possible numeric value for your machine. What is too large? You could test the return value of factorial against Inf.

You can reuse previously tested code writing this function as a wrapper for a previously written pois.pmf and call that function only after testing the for specified conditions.

Test this function by repeating the plots from Homework 4, Ex 4. How is the function different than dpois?

**Warning** You may not be able to call this new function exactly as in the last exercise (Hint - what are the rules for conditions in if statements?). Instead, you might need to create a matrix or data table and use apply functions, or write a loop for visit each element in a vector of x.

seq =seq(-5,5,by=0.1)  
  
pois.pmf = function(x){  
 for(i in 1:x)  
 if (i>170) {i = 170  
 } else if ( i < 170){i=i  
 }  
 else if (i<0){print(NaN)  
 } else {  
 i = trunc(i)}  
 ( 0.36787944117\*(-1)\*\*(x))/(factorial(x))  
  
  
}

pois.pmf(seq)

## Warning in 1:x: numerical expression has 101 elements: only the first used

## Warning in gamma(x + 1): NaNs produced

## [1] NaN NaN NaN NaN NaN  
## [6] NaN NaN NaN NaN NaN  
## [11] NaN NaN NaN NaN NaN  
## [16] NaN NaN NaN NaN NaN  
## [21] NaN NaN NaN NaN NaN  
## [26] NaN NaN NaN NaN NaN  
## [31] NaN NaN NaN NaN NaN  
## [36] NaN NaN NaN NaN NaN  
## [41] NaN NaN NaN NaN NaN  
## [46] NaN NaN NaN NaN NaN  
## [51] 0.367879441 NaN NaN NaN NaN  
## [56] NaN NaN NaN NaN NaN  
## [61] -0.367879441 NaN NaN NaN NaN  
## [66] NaN NaN NaN NaN NaN  
## [71] 0.183939721 NaN NaN NaN NaN  
## [76] NaN NaN NaN NaN NaN  
## [81] -0.061313240 NaN NaN NaN NaN  
## [86] NaN NaN NaN NaN NaN  
## [91] 0.015328310 NaN NaN NaN NaN  
## [96] NaN NaN NaN NaN NaN  
## [101] -0.003065662