Homework 4 Arrays and Lists

Gavin Gunawardena

6/25/21

STAT 600

Dr. Claussen

Instructions

Reuse

For many of these exercises, you may be able to reuse functions written in prior homework. Include those functions here. You may find that you will need to modify your functions to work correctly for these exercises.

```
#Convert SAS functions to R and test them
norm.pdf <- function(x, mu=0, sigma=1) {
  pi2 <- pi*2
  var_1 <- sigma^2
  part1 <- (1/((sigma)*(sqrt(pi2))))
  part2 <- (exp(((-1*(x-mu)^2/(2*var_1)))))
  return(round(part1 * part2, 4))

}


pois.pmf <- function(x, lambda) {
    var_fac <- factorial(x)
    return((exp(-1*lambda)*lambda^x)/var_fac);
    }


    print(pois.pmf(20,1))

## [1] 1.5121013503e-19

    print(dpois(20,1))

## [1] 1.5121013503e-19
```

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)
Year <- c(1936, 1946, 1951, 1963, 1975, 1997, 2006)
```

Exercise 1

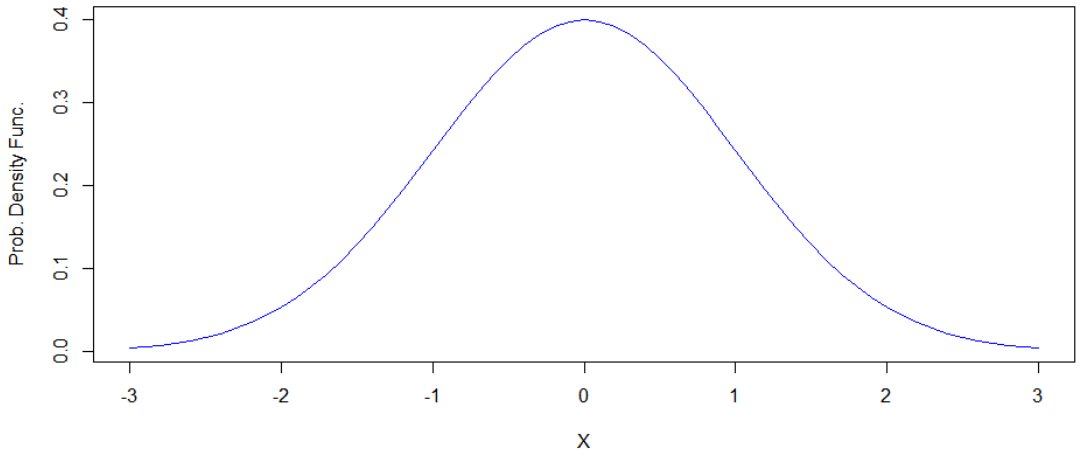
In this exercise, we will test your `norm.pdf` function with a range of inputs.

Do not print the vectors you create for this exercise in the final typeset submission We will check the results by examining the plots, and printing the vectors themselves will unnecessarily clutter your report. If you get stuck, use the built normal functions to create your plots.

Part a.

Generate a sequence of values from $-3, \ldots, 3$ incremented by 0.1; let this be `x_1`. Calculate the PDF of each value of `x_1` using the `norm.pdf` function from Homework 3, letting `mu=0` and `sd=1`. Plot the PDF curve ($norm.pdf$ is the dependent variable, x is independent) as a line graph.

```
#Set variables and do calculations
x_1 <- seq(-3.0,3,0.1)
x_1_pdf <- norm.pdf(x_1)


#plot data
plot(x_1,x_1_pdf,type='l',ylab='Prob. Density Func.',xlab='X',col='blue');
```



Part b.

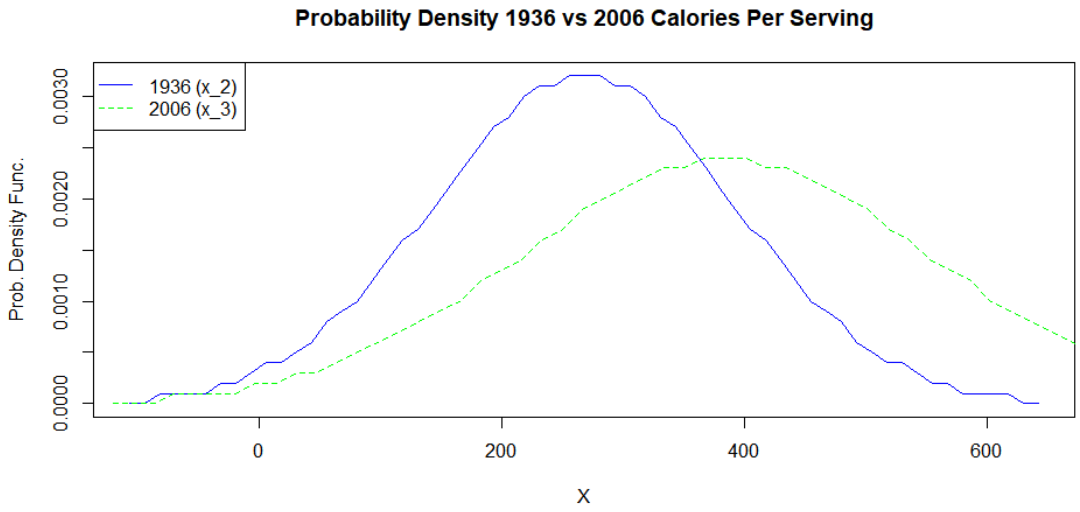
Let m_{1936} be the mean Calories per Serving from 1936, and let m_{2006} be the mean Calories per Serving, 2006. Let s_{1936} and s_{2006} be the corresponding standard deviations.

Create two sequences and name these x_2 and x_3. Define x_2 to be a range of values $[m_{1936} - 3 \times s_{1936}, \ldots, m_{1936} + 3 \times s_{1936}]$ and define x_3 to be $[m_{2006} - 3 \times s_{2006}, \ldots, m_{2006} + 3 \times s_{2006}]$. x_2 and x_3 should be the same length as x_1.

Calculate the corresponding pdf for these sequences, using $\{\mu = m_{1936}, \sigma = s_{1936}\}$ with x_2 and use $\{\mu = m_{2006}, \sigma = s_{2006}\}$ with x_3.

As with part a, plot the pdf curve for both sequences, but include both in the same graph. Use two different colors or line types for each curve. You may need to use `min` and `max` to find `xlim` values or `ylim` to fit both curves on the same plot. The first curve in this graph should appear identical to the curve in part a; the second curve will be similar but will differ in location and spread.

```
#Set variables and do calculations
m_1936 <- CaloriesPerServingMean[1]
m_2006 <- CaloriesPerServingMean[7]
s_1936 <- CaloriesPerServingSD[1]
s_2006 <- CaloriesPerServingSD[7]
x_2 <- seq(m_1936-3*s_1936,m_1936+3*s_1936,s_1936*.1)
x_3 <- seq(m_2006-3*s_2006,m_2006+3*s_2006,s_2006*.1)
 # print(x_2)
 # print(x_3)
x_2_pdf <- norm.pdf(x_2,m_1936,s_1936)
x_3_pdf <- norm.pdf(x_3,m_2006,s_2006)
# print(x_2_pdf)
# print(x_3_pdf)

#plot
x <- seq(-5,5,.1)
plot(x_2,x_2_pdf,type='l',main='Probability Density 1936 vs 2006 Calories Per
Serving' ,ylab='Prob. Density Func.',xlab='X',col='blue',ylim=c(min(x_2_pdf),
max(x_2_pdf)),xlim=c(min(x_2),max(x_2)));
lines(x_3,x_3_pdf,col='green',lty=2);
legend('topleft', legend = c("1936 (x_2)", "2006 (x_3)"),
       pch = c(NA,NA), lty = c(1, 2),
       col = c('blue','green'))
```



If you choose to solve this with SAS, I've included code in the SAS template to create the graphs, since combining plots in IML is not as easy as in R.

If you wish, you may reproduce the curves using `dnorm` to compare with your function.

Exercise 2

Suppose we wish to determine the relationship between per Calories per Serving and Year. We can determine this by solving a system of linear equations, of the form

$$\begin{aligned} 268.1 &= \beta_1 + \beta_2 \times 1936 \\ 271.1 &= \beta_1 + \beta_2 \times 1946 \\ \vdots &= \vdots \\ 384.4 &= \beta_1 + \beta_2 \times 2006 \end{aligned}$$

We write this in matrix notation as

$$\begin{pmatrix} 268.1 \\ 271.1 \\ \vdots \\ 384.4 \end{pmatrix} = \begin{pmatrix} 1 & 1936 \\ 1 & 1946 \\ \vdots & \vdots \\ 1 & 2006 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}^t$$

We can also write this as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}$$

and find a solution by computing $\widehat{\boldsymbol{\beta}} = \mathbf{X}^{-1}\mathbf{y}$.

However, an exact solution for the inverse, $\mathbf{X}^{-1}$ require square matrices, so commonly we use the *normal* equations,

$$\mathbf{X}^t\mathbf{y} = \mathbf{X}^t\mathbf{X}\boldsymbol{\beta}$$

(where $\mathbf{X}^t$ is the transpose of $\mathbf{X}$). We then find

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{y}$$

Answer

Define appropriate X and y matrices (y can be a vector in R) in the chunk below.

Multiply the transpose of X by X, then use `solve` (R) or `inv` (IML) to find the inverse $(\mathbf{X}^t\mathbf{X})^{-1}$. Multiply this by the product of transpose X and y to find `hat.beta`.

Print your `hat.beta`.

```
X_m <- Year
y_m <- CaloriesPerServingMean
x <- matrix(cbind(1,X_m),nrow=7,ncol=2)
y <- matrix(cbind(y_m),nrow=7,ncol=1)
transpose_times_x <- t(x) %*% x

  hat.beta <- solve(transpose_times_x)%*%t(x)%*%y
  hat.y <- x%*%hat.beta
  # print(hat.y)
  print(hat.beta)
```

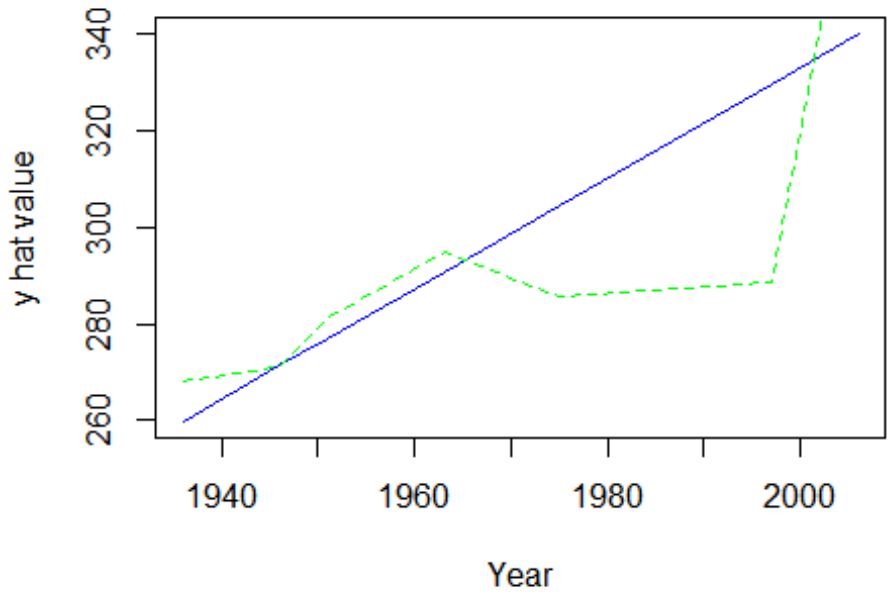
```
##                    [,1]
## [1,] -1965.37260726008
## [2,]     1.14933993399
```

To check your work, calculate the values predicted by your statistical model. Compute `hat.y` by multiplying X and `hat.beta`,

$$\hat{y} = \mathbf{X}\hat{\beta}$$

Plot y vs the independent variable (the second column of X) as points, and `hat.y` vs independent variable as a line, preferably a different colors. The `hat.y` values should fall a straight line that interpolates y values.

```
#Plot Check
plot(x[,2],hat.y,type='l',ylab='y hat value',xlab='Year',col='blue',xlim=c(mi
n(x[,2]),max(x[,2])),ylim=c(min(hat.y),max(hat.y)))
lines(x[,2],y,col='green',lty=2)
```



You can also compare your result to the R function (set `eval=TRUE`).

```
#,eval=False}
summary(lm(CaloriesPerServingMean~Year))

##
## Call:
## lm(formula = CaloriesPerServingMean ~ Year)
##
## Residuals:
##           1           2           3           4           5
6
##   8.35049505  -0.14290429   3.91039604   3.91831683 -18.97376238 -41.25924
092
##           7
##  44.19669967
##
## Coefficients:
##                    Estimate      Std. Error  t value Pr(>|t|)
## (Intercept) -1965.372607261   875.878613771 -2.24389 0.074849 .
## Year            1.149339934     0.445090776  2.58226 0.049297 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28.6917147 on 5 degrees of freedom
## Multiple R-squared:  0.571479922,    Adjusted R-squared:  0.485775906
## F-statistic: 6.66806471 on 1 and 5 DF,  p-value: 0.0492969117
```

Alternative methods

You can also compute $\hat{\beta}$ by passing both $\mathbf{X}^t\mathbf{X}$ and $\mathbf{X}^t\mathbf{y}$ as arguments to `solve`.

Alternatively, you can install the `MASS` library and use `ginv` to compute a generalized inverse $\mathbf{X}^{-1}$. Use this to compute $\widehat{\boldsymbol{\beta}} = \mathbf{X}^{-}\mathbf{y}$ in the chunk below:

```
library(MASS)
print(hat.beta <- ginv(X) %*% y)
```

Exercise 3

Given a vector of mean estimates $x = x_1, x_2, \ldots, x_k$, a vector of standard deviations $s = s_1, s_2, \ldots, s_k$ and a vector of sample sizes $n = n_1, n_2, \ldots, n_k$, we can calculate a one-way analysis of variance by

$$MSB = \frac{n_1(x_1 - \bar{x})^2 + n_2(x_2 - \bar{x})^2 + \cdots + n_k(x_k - \bar{x})^2}{k-1} = \frac{\sum_i n_i\,(x_i - \bar{x})^2}{k-1}$$

and

$$MSW = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2 + \cdots (n_k - 1)s_k^2}{N-k} = \frac{\sum_i (n_i - 1)s_i^2}{N-k}$$

where $\bar{x}$ is the weighted mean of x_i, where $\bar{x} = \frac{\sum_i n_i * x_i}{N}$ and $N = \sum_i n_i$. The test statistic is $F = \frac{MSB}{MSW}$ which is distributed as $F_{\alpha,k-1,N-k}$

Part a

Calculate MSW and MSB for Calories per Serving from Wansink Table 1. You can use the variables `CaloriesPerServingMean` and `CaloriesPerServingSD` defined below. Let $n_1 = n_2 \ldots = n_k = 18$

Use array functions and arithmetic for your calculations, you should not need iteration (for loops). Do not hard code values for N and k, calculate these from the `CaloriesPerServingMean` or `CaloriesPerServingSD`.

Print both MSB and MSW.

```
msb.solve <- function(x,sample_size) {
  #x can be either standard deviation or mean
  x_weighted_avg <- weighted.mean(x, rep(1,7))
  #print(x_weighted_avg)
  iterations <- rep(sample_size,length(x))
  #print(iterations)
    pt_1 <- sum(iterations%*%(x-x_weighted_avg)^2)
  pt_2 <- length(x) - 1
  return(pt_1/pt_2)
}
```

```
MSW.solve <- function(x, sample_size) {
  #x must be standard deviation
  iterations <- rep(sample_size,length(x))
  pt_1 <- sum((iterations-1)*(x^2))
  pt_2 <- sum(iterations)-length(x)
  return(pt_1/pt_2)
}

#Print Results
print('Mean Square Between Groups:')
```

```
## [1] "Mean Square Between Groups:"
```

```
print(msb.solve(CaloriesPerServingMean,18))
```

```
## [1] 28815.96
```

```
print('Mean Square Within Groups:')
```

```
## [1] "Mean Square Within Groups:"
```

```
print(MSW.solve(CaloriesPerServingSD,18))
```

```
## [1] 16508.5985714
```

Part b

Calculate an F-ratio and a p for this F, using the F distribution with $k-1$ and $N-k$ degrees of freedom. Use $\alpha = 0.05$. Compare these values to the corresponding values reported in Wansink Table 1.

```
#Set variables and do calculations
df1 <- length(CaloriesPerServingMean)-1
df2 <- sum(rep(18,length(CaloriesPerServingMean)))-length(CaloriesPerServingM
ean)
MSB <- msb.solve(CaloriesPerServingMean,18)
MSW <- MSW.solve(CaloriesPerServingSD,18)
f_stat <- MSB / MSW
p_val <- pf(f_stat,df1,df2,lower.tail=FALSE)

#Print values:
print('F Statistic')
```

```
## [1] "F Statistic"
```

```
print(round(f_stat,4))
```

```
## [1] 1.7455
```

```
print('P Value')
```

```
## [1] "P Value"
```

```
print(round(p_val,4))
```

```
## [1] 0.1163
```

Compared to the results shown in Wansink Table 1 for average calories per serving, these above obtained results are vastly different, with the P value I obtained being farther from 0. Also, the F value I obtained is less than the F critical value which, with an alpha level of .05, is 2.1750(going by an F chart), while the F value obtained by Wansink was 436.9 which is higher than the critical value, although from doing some research, it seems it's best not to solely use the F value but instead use it with the results of a P value or another test for checking statistical significance.

You can also check results by entering appropriate values in an online calculator like http://statpages.info/anova1sm.html .

Exercise 4

In this, we compare the normal and Poisson distributions, using the functions you've written previously. This is also a way to test your normal and Poisson functions over a range of arguments.

Do not print the vectors you create for this exercise in the final typeset submission We will check the results by examining the plots, and printing the vectors themselves will unnecessarily clutter your report. If you get stuck, use the built functions to create your plots. However, the final submission must call your functions.

Part a

Create a sequence of x_a from $(-5 \ldots 5)$, incremented by 0.1. Calculate the normal PDF for each x, assuming $\mu = 0$ and $\sigma = 1$. Also calculate Poisson PMF of each x given a `lambda=1`.

Plot both sets of probabilities against `x_a` as lines, using a different color for each curve. Make sure that both curves fit in the plot; you may need to determine minimum and maximum values and set these as graphic parameters (see `ylim`).

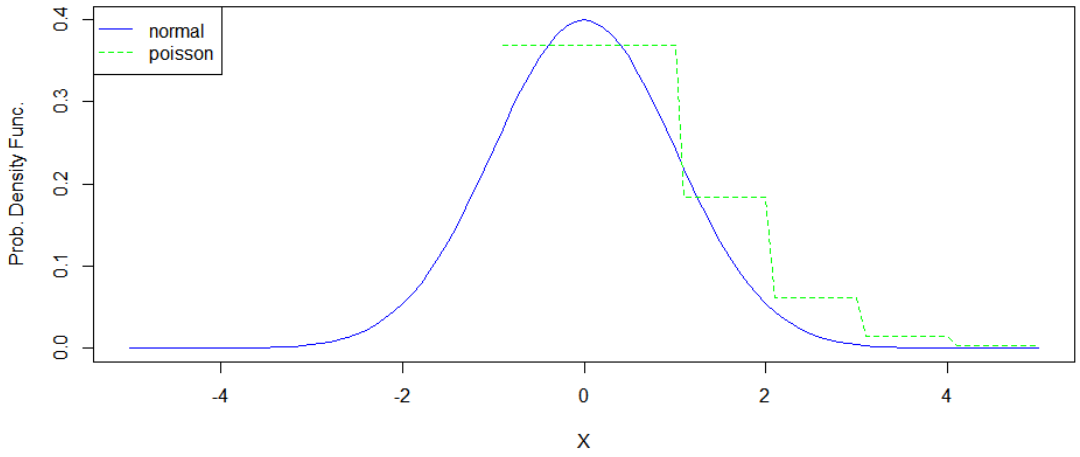
Warning: if you do this in SAS, you may have to adjust the lower bound of x.

```
#Set variables and do calculations
x_q4a <- seq(-5.0,5,0.1)
x_q4a_norm <- norm.pdf(x_q4a)
 x_q4a_pois <- pois.pmf(ceiling(x_q4a),1)
```

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

```
#Alternative for testing
 #x_q4a_pois <- pois.pmf(x_q4a,1)
 # x_q4a_pois <- dpois(ceiling(x_q4a),1)
```

```
#plot
plot(x_q4a,x_q4a_norm,type='l' ,ylab='Prob. Density Func.',xlab='X',col='blue
',ylim=c(min(x_q4a_norm),max(x_q4a_norm)),xlim=c(min(x_q4a),max(x_q4a)));
lines(x_q4a,x_q4a_pois,col='green',lty=2);
legend('topleft', legend = c("normal", "poisson"),
       pch = c(NA,NA), lty = c(1, 2),
       col = c('blue','green'))
```



Does this graph tell you if your Normal PDF function behaves properly? Does your Poisson handle negative or non-integer values as expected? You might need to call a rounding function on the parameters inside your function.

Yes, I believe it does since, at least for positive values, it follows the normal distribution. No, my Poisson function does not handle negative or non-integer values as expected, especially compared to the built-in Poisson function. From what I understand from doing some research, this is due to the factorial function in R not working with negative numbers and thus I'm getting some null results. From testing it, rounding functions do not seem to have an effect on the plot.

Part b

Create a sequence of $x_b = \lfloor \mu - 5 \times \sigma \rfloor, \ldots, \lceil \mu + 5 \times \sigma \rceil$ using mean and standard deviation for Servings per Recipe from 1936.

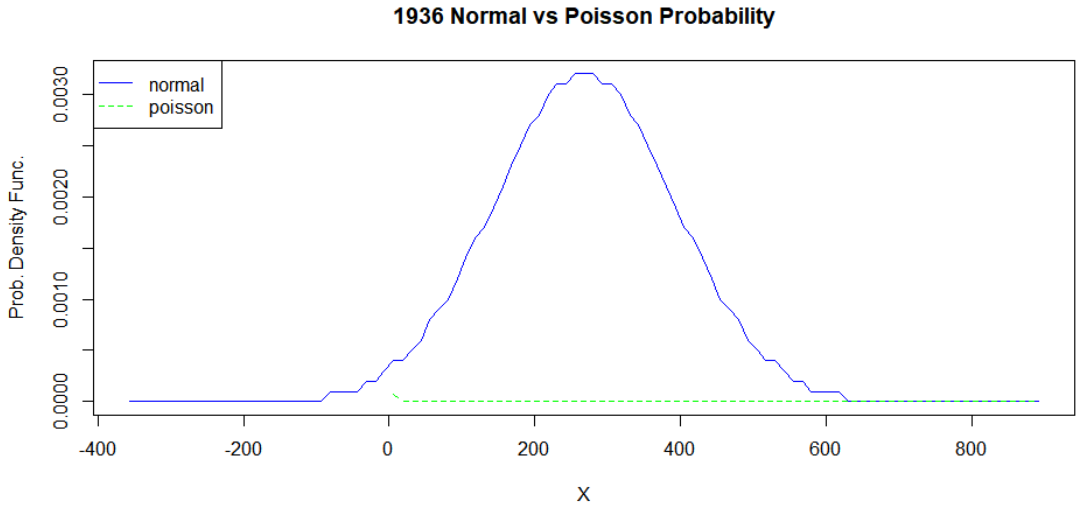
Calculate the normal and Poisson probability for each x_b as in part a, again using mean and standard deviation from servings per recipe, 1936. The length of this vector should be the same length as the x_a vector as in part a (± 1), so you will need to calculate an interval based on the range `x_b` and the number of elements in `x_a`

Show the the length of both x vectors are similar by calling `length` for each.

Repeat the plot from part a with this sequence.

If you choose to solve this with SAS, I've included code in the SAS template to create the graphs, since combining plots in IML is not as easy as in R.

```
m_1936 <- CaloriesPerServingMean[1]
s_1936 <- CaloriesPerServingSD[1]
x_q4b <- seq(m_1936-5*s_1936,m_1936+5*s_1936,s_1936*.1)
#print(x_q4b)

#normal probability
x_q4b_norm <- norm.pdf(x_q4b,m_1936,s_1936)

#tested alternative
# x_q4b_norm <- dnorm(x_q4b,m_1936,s_1936)


#Poisson probability
x_q4b_pois <- pois.pmf(ceiling(x_q4b),1)

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

#Tested alternatives
# x_q4b_pois <- dpois(x_q4b,1)
# x_q4b_pois <- dpois(ceiling(x_q4b),1)


#lengths
print('question 4a length')

## [1] "question 4a length"

(length(x_q4a))

## [1] 101

print('question 4b length')

## [1] "question 4b length"

print(length(x_q4b))

## [1] 101

#plot
plot(x_q4b,x_q4b_norm,type='l',main='1936 Normal vs Poisson Probability' ,yla
b='Prob. Density Func.',xlab='X',col='blue',ylim=c(min(x_q4b_norm),max(x_q4b_
norm)),xlim=c(min(x_q4b),max(x_q4b)));
lines(x_q4b,x_q4b_pois,col='green',lty=2);
legend('topleft', legend = c("normal", "poisson"),
       pch = c(NA,NA), lty = c(1, 2),
       col = c('blue','green'))
```



To check you work, duplicate the plots by calling built in normal and Poisson functions. Does the built in Poisson function handle negative x differently than your function?

In this case I'm getting much different results than in 4a. The poisson distribution isn't following the normal distribution via the built in poisson distribution function nor with my version. These functions are getting very opposite results when not rounded, with my function getting results that include infinite values and the built-in function getting results that include 0 values. When rounded, these 2 functions are getting very similar results that don't follow the normal distribution. Also, when rounded, my poisson distribution function seems to stop at negative values compared to the built in function which seems to still work on negative values.