GU4206-GR5206 Final Exam Key [100 pts]

Student (UNI)

May 4th, 2018

The STAT Spring 2018 GU4206-GR5206 Final Exam is open notes, open book(s), open computer and online resources are allowed. Students are **not** allowed to communicate with any other people regarding the final with the exception of the instructor (Gabriel Young) and TA (Fan Gao). This includes emailing fellow students, using WeChat and other similar forms of communication. If there is any suspicious of students cheating, further investigation will take place. If students do not follow the guidelines, they will receive a zero on the exam and potentially face more severe consequences. The exam will be posted on Canvas at 9:00AM. Students are required to submit both the .pdf and .Rmd files on Canvas (or .html if you must) by 12:00PM. Late exams will not be accepted. If for some reason you are unable to upload the completed exam on Canvas by 12:00PM, them immediately email the markdown file to the course TA.

Part 1: Simulation [40 pts]

In this section, we consider a **mixture** of two normal distributions. Here we assume that our random variable is governed by the probability density $f(x)$, defined by

$$\begin{aligned} f(x) &= f(x; \mu_1, \sigma_1, \mu_2, \sigma_2, \delta) \\ &= \delta f_1(x; \mu_1, \sigma_1) + (1 - \delta) f_2(x; \mu_2, \sigma_2) \\ &= \delta \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp - \frac{1}{2\sigma_1^2}(x - \mu_1)^2 + (1 - \delta) \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp - \frac{1}{2\sigma_2^2}(x - \mu_2)^2, \end{aligned}$$

where $-\infty < x < \infty$ and the parameter space is defined by $-\infty < \mu_1, \mu_2 < \infty$, $\sigma_1, \sigma_2 > 0$, and $0 \leq \delta \leq 1$. The **mixture parameter** δ governs how much mass gets placed on the first distribution $f(x; \mu_1, \sigma_1)$ and the complement of δ governs how much mass gets placed on the other distribution $f_2(x; \mu_2, \sigma_2)$.

In our setting, suppose that we are simulating $n = 10,000$ heights from the population of both males and females. Assume that males are distributed normal with mean $\mu_1 = 70$[in] and standard deviation $\sigma_1 = 3$[in] and females are distributed normal with mean $\mu_2 = 64$[in] and standard deviation $\sigma_2 = 2.5$[in]. Also assume that each distribution contributes equal mass, i.e., set the mixture parameter to $\delta = .5$. The distribution of males is governed by

$$f_1(x; \mu_1, \sigma_1) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp - \frac{1}{2\sigma_1^2}(x - \mu_1)^2, \quad -\infty < x < \infty,$$

and the distribution of females is governed by

$$f_2(x; \mu_2, \sigma_2) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp - \frac{1}{2\sigma_2^2}(x - \mu_2)^2, \quad -\infty < x < \infty.$$

The goal is to **simulate** from the **mixture distribution**

$$\delta f_1(x; \mu_1, \sigma_1) + (1 - \delta) f_2(x; \mu_2, \sigma_2),$$

where $\mu_1 = 70, \sigma_1 = 3, \mu_2 = 64, \sigma_2 = 2.5, \delta = 2.5$ using the accept-reject algorithm.

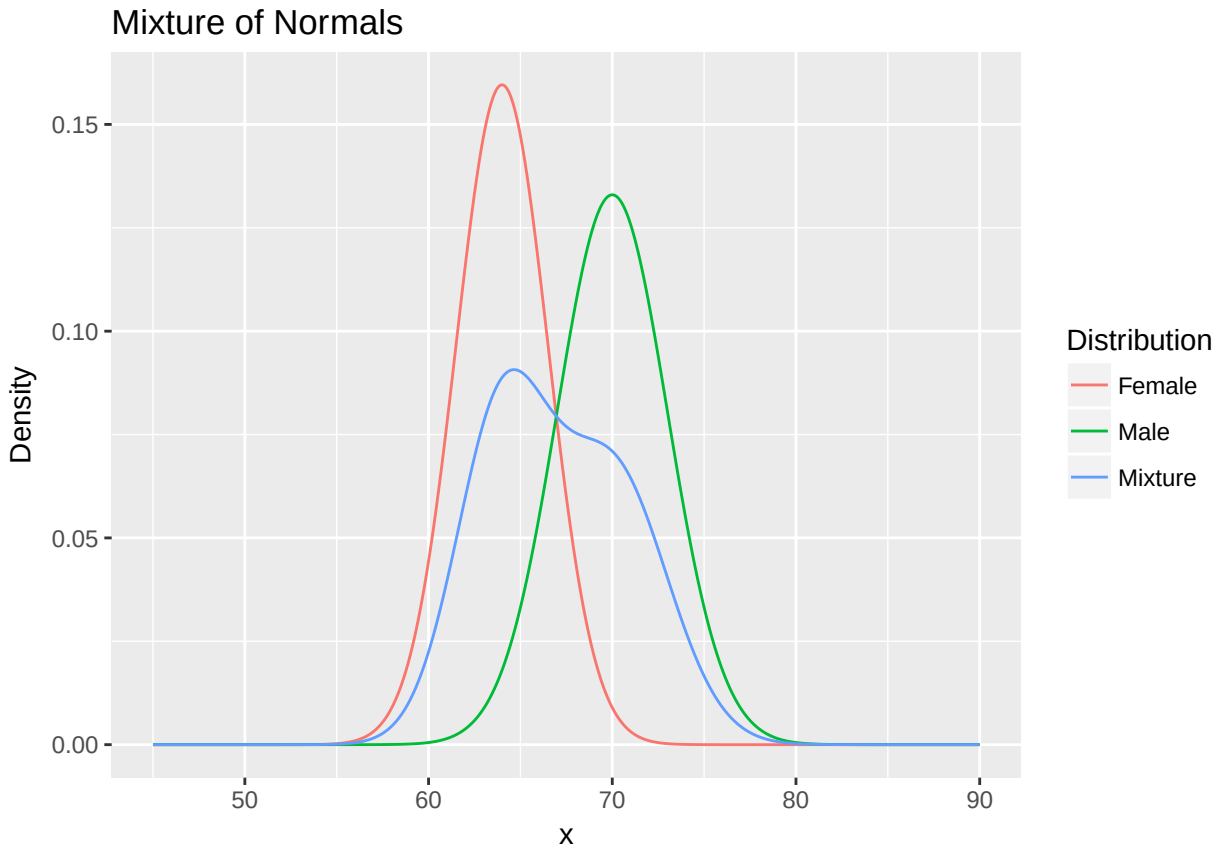
Perform the following tasks:

1) [10 pts] Using **ggplot**, graph $f_1(x;\mu_1,\sigma_1)$, $f_2(x;\mu_2,\sigma_2)$ and the mixture $f(x)$ all on the same plot. Make sure the plot includes a legend and is labeled appropriately.

```
x <- seq(45,90,by=.05)
n.x <- length(x)
f_1 <- dnorm(x,mean=70,sd=3)
f_2 <- dnorm(x,mean=64,sd=2.5)
f <- function(x) {
  return(.5*dnorm(x,mean=70,sd=3) + .5*dnorm(x,mean=64,sd=2.5))
  }

plot_df <-  data.frame(x=c(x,x,x),
                       Density=c(f_1,f_2,f(x)),
                       Distribution=c(rep("Male",n.x),rep("Female",n.x),rep("Mixture",n.x))
                       )

library(ggplot2)
ggplot(data = plot_df) +
   geom_line(mapping = aes(x = x, y = Density,color=Distribution))+
   labs(title = "Mixture of Normals")
```



2) [25 pts] Use the **accept-reject** algorithm to simulate from this mixture distribution. To receive full credit:

   2.i Clearly identify an **easy to simulate** distribution $g(x)$. I recommend picking a normal distribution or a Cauchy distribution for $g(x)$.

2.ii Identify a **suitable** value of *alpha* such that your envelope function $e(x)$ satisfies

$$f(x) \leq e(x) = g(x)/\alpha, \quad \text{where} \quad 0 < \alpha < 1.$$

Note that you must choose α so that $e(x)$ is close to $f(x)$. Show that your *alpha* is **suitable** using a plot.

2.iii Simulate 10,000 draws from the mixture distribution using the **accept-reject** algorithm. Display the first 20 simulated values. Also, using **ggplot** or **base R**, construct a histogram of the simulated mixture distribution with the true mixture pdf $f(x)$ overlayed on the plot.

2.i) [5 pts]

```
x[which.max(f(x))]
```

```
## [1] 64.65
```

```
f(x[which.max(f(x))])
```

```
## [1] 0.09069388
```

Notice that $f(x)$ is maximized around 64-65. I am choosing a $g(x)$ to be a Cauchy distribution centered at 66 with scale parameter 7.

```
g <- function(x) {

  s=7
  l=66
  return(1/(pi*s*(1+((x-l)/s)^2)))

}
```

2.ii) [10 pts] By inspection, we see that choosing $\alpha = .44$ allows $e(x)$ to be greater than $f(x)$ for all x and the envelope sits relatively close to the target distribution.

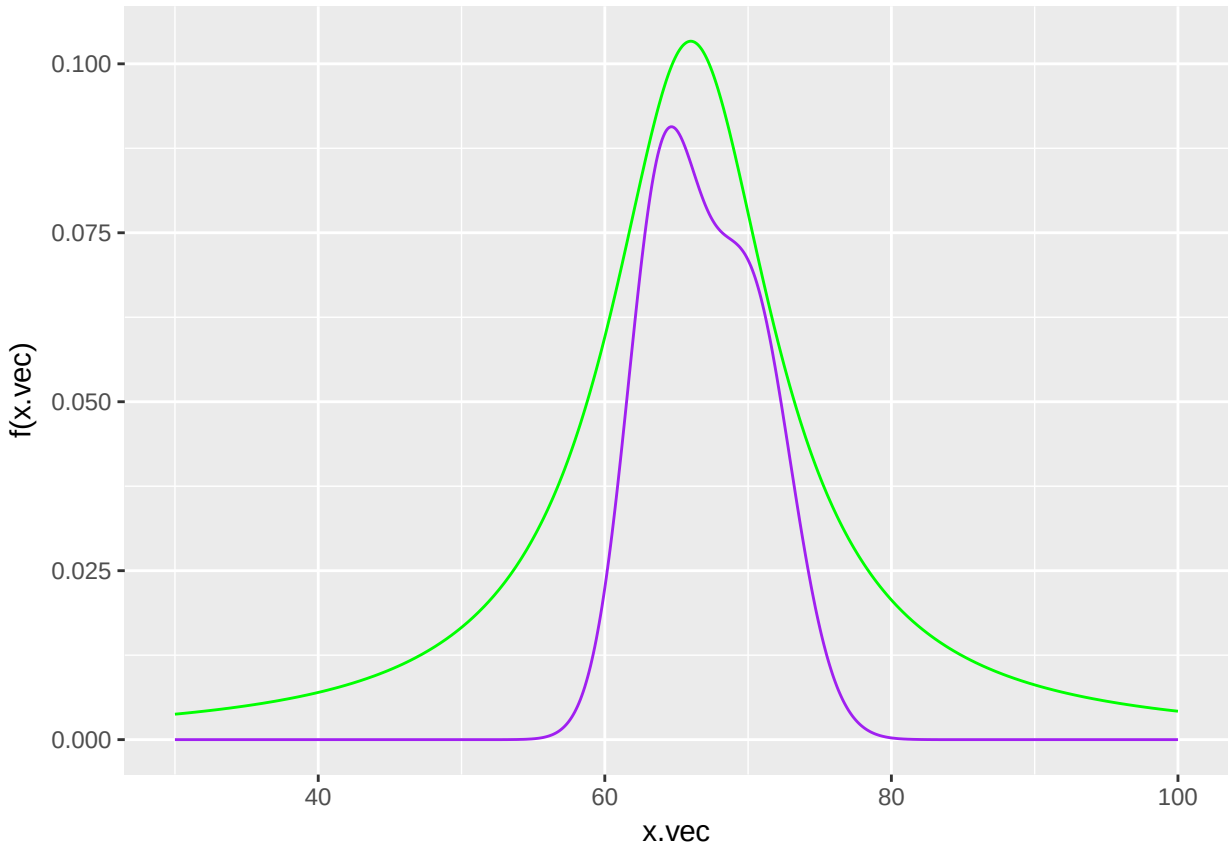
```
# Choose alpha
alpha <- .44

# Define envelope e(x)
e <- function(x) {

  return(g(x)/alpha)

}

# Plot
x.vec <- seq(30,100,by=.1)
ggplot() +
   geom_line(mapping = aes(x = x.vec, y = f(x.vec)),col="purple")+
   geom_line(mapping = aes(x = x.vec, y = e(x.vec)),col="green")
```




```
   # labs(title = paste("Envelope: alpha=",alpha))

# Is g(x)>f(x)?
all(e(x.vec)>f(x.vec))
```

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

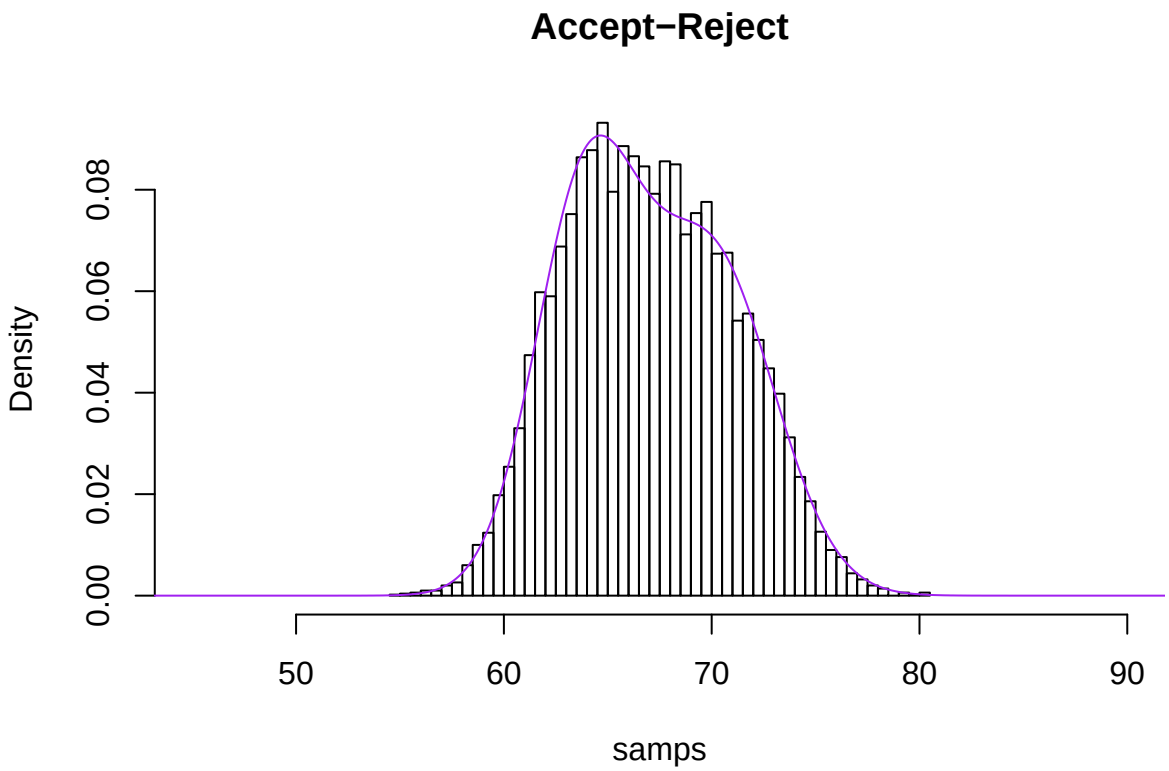
2.iii) [10 pts] The **Accept-Reject** algorithm is coded below:

```
n.samps <- 10000   # number of samples desired
n       <- 0             # counter for number samples accepted
samps   <- numeric(n.samps) # initialize the vector of output
while (n < n.samps) {
  y <- rcauchy(1, location = 66, scale = 7)    #random draw from g
  u <- runif(1)
  if (u < f(y)/e(y)) {
     n        <- n + 1
        samps[n] <- y
    }
  }
head(samps,20)
```

```
##  [1] 65.26493 70.84870 69.17894 59.36893 74.08740 72.19642 61.40206
##  [8] 71.76737 70.05914 67.51548 63.41013 70.48924 77.99430 62.15789
## [15] 71.11836 72.50705 63.77986 67.35420 70.96588 70.14943
```

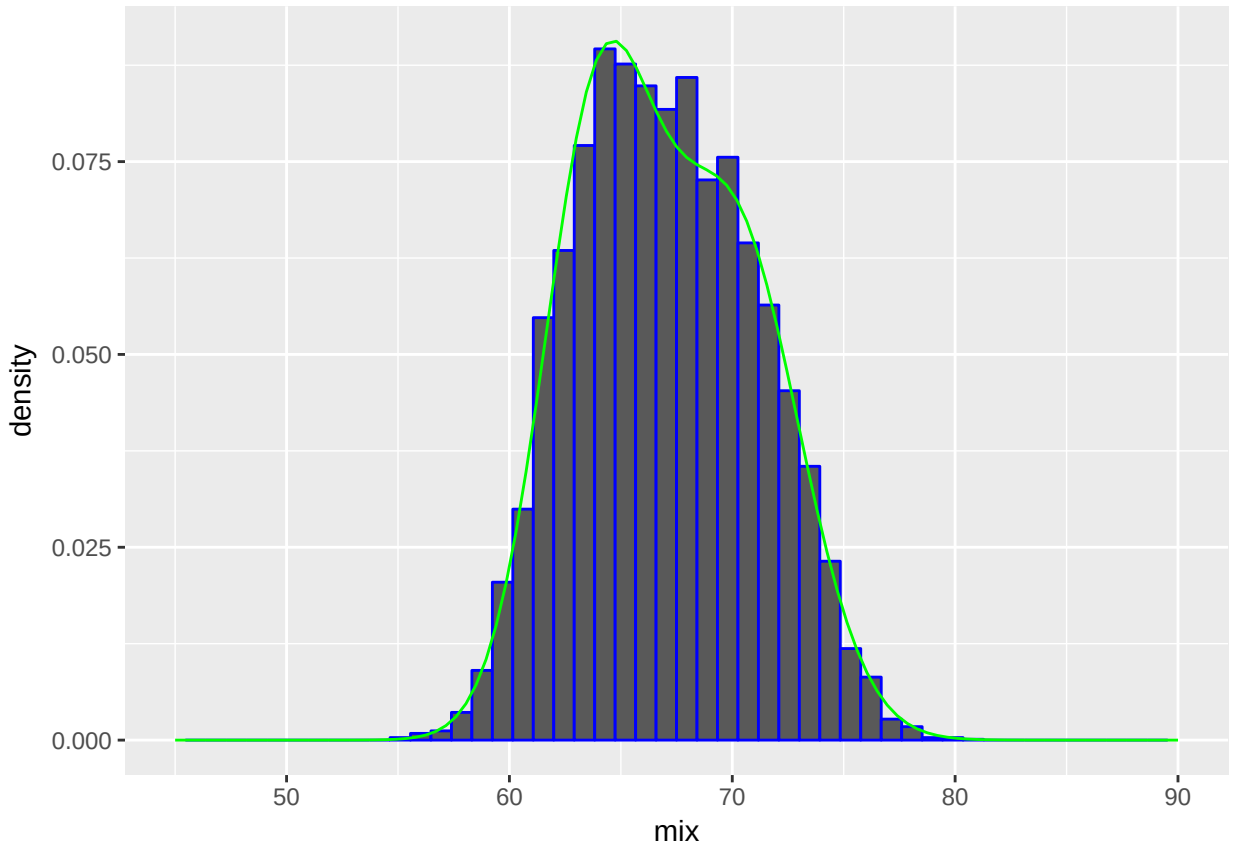
The base R plot is displayed below:

```r
hist(samps,breaks=50,probability = TRUE,main="Accept-Reject",xlim=c(45,90))
lines(x.vec,f(x.vec),col="purple")
```



The ggplot plot is displayed below:

```r
mix.sim <- data.frame(mix=samps)
ggplot(mix.sim)+
  geom_histogram(mapping=aes(x=mix,y=..density..),col="blue",bins=50)+
  xlim(45,90)+
  stat_function(fun=f,colour="green")
```



3) [5 pts] Slightly change the **Accept-Reject** algorithm from Part (2.iii) to also include the acceptance rate, i.e., how many times did the algorithm accept a draw compared to the total number of trials performed. What proportion of cases were accepted? Compare this number to your chosen α and comment on the result.

```
n.samps <- 10000   # number of samples desired
n <- 0 # counter for number samples accepted
m <- 0 # counter for number of trials
samps   <- numeric(n.samps) # initialize the vector of output
while (n < n.samps) {
  m <- m +1
  y <- rcauchy(1, location = 66, scale = 7)     #random draw from g
  u <- runif(1)
  if (u < f(y)/e(y)) {
     n        <- n + 1
        samps[n] <- y
    }
  }
n.samps/m
```

```
## [1] 0.441989
```

```
alpha
```

```
## [1] 0.44
```

Notice that α is very close to the proportion of samples accepted. Very cool!

Part II: Maximum Likelihood Estimaiton and Newton's Method [25 pts]

Recall in logistic regression, the likelihood function is derived by **linking** the mean of Y_i with a linear function.

Logistic Regression Model:

Let $Y_1, Y_2, \ldots, Y_n$ be independently distributed Bernoulli random variables with respective success probabilities $p_1, p_2, \ldots, p_n$. Then the **logistic regression model** is:

$$E[Y_i] = p_i = \frac{e^{(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_p x_{i,p})}}{1+e^{(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\cdots+\beta_p x_{i,p})}}, \quad i = 1, 2, \ldots, n.$$

Notice with some simple algebra, the above model can be expresses as:

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{i,p} x_i, \quad i = 1, 2, \ldots, n.$$

The main idea is to **link** the expected value of Y_i ($E[Y_i] = p_i$) to a linear function. This same principle can be applied to other settings.

Data Description

Consider a geriatrics experiment designed as a prospective study to investigate the effects of two interventions on the frequency of falls. One hundred subjects were randomly assigned to one of the two interventions: education only ($X_1 = 0$) and education plus aerobic exercise training ($X_1 = 1$). Subjects were at least 65 years of age and in reasonably good health. Three variables considered to be important as control variables were gender ($X_2 : 0$ =female, 1=male), a balance index (X_3), and a strength index (X_4). The higher the balance index, the more stable the subject: and the higher the strength index, the stronger the subject. Let Y be the number of falls during the six month study.

```
glm.data <- read.table("https://netfiles.umn.edu/users/nacht001/www/nachtsheim/Kutner/Chapter%2014%20Da
names(glm.data) <- c("Y","X1","X2","X3","X4")
head(glm.data)
```

```
##   Y X1 X2 X3 X4
## 1 1  1  0 45 70
## 2 1  1  0 62 66
## 3 2  1  1 43 64
## 4 0  1  1 76 48
## 5 2  1  0 51 72
## 6 1  1  1 73 39
```

In this setting, the response variable takes on discrete count values, therefore it is reasonable to assume $Y_1, Y_2, \ldots, Y_{100}$ are independent Poisson random variables with mean $E[Y_i] = \lambda_i$. Here we can choose a link function that relates λ_i to the linear function $\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4}$.

Perform the follwoing task:

4) [25 pts] Assume $Y_1, Y_2, \ldots, Y_{100}$ are independent Poisson random variables with mean

$$E[Y_i] = \lambda_i = \exp\left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4}\right)$$

Note that the link function is $exp(u)$. Use maximum likelihood estimation to estimate the Poisson regression model. To receive full credit:

4.i Define the negative log-likelihood function in R. Name the function **pois.neg.ll**.

4.ii Test the negative log-likelihood function at the parameter point **rep(0,5)**.

4.iii Use the **Newton's Method** or **Gradient Descent** algorithm from class to estimate coefficients $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$. Display the estimated parameters and the number of iterations the algorithm took to converge. For partial credit, you can use **nlm()**.

4.i) [10 pts]

```
neg.ll <- function(params=rep(0,5),data=glm.data) {

  beta.0 <- params[1]
  beta.1 <- params[2]
  beta.2 <- params[3]
  beta.3 <- params[4]
  beta.4 <- params[5]

  linear.term <- beta.0+beta.1*data$X1+beta.2*data$X2+beta.3*data$X3+beta.4*data$X4
  mean <- exp(linear.term)
  return(sum(-dpois(data$Y,lambda=mean,log=TRUE)))

}
```

4.ii) [5 pts]

```
neg.ll(params=rep(0,5))
```

```
## [1] 362.8506
```

4.iii) [10 pts]

```
library(numDeriv)
Newton.method <- function(f, x0, max.iter = 200, stopping.deriv = 0.01, ...) {

  n    <- length(x0)
  xmat <- matrix(0, nrow = n, ncol = max.iter)
  xmat[,1] <- x0

  for (k in 2:max.iter) {
    # Calculate the gradient
    grad.cur <- grad(f, xmat[ ,k-1], ...)

    # Calculate the hessian
    hess.cur <- hessian(f, xmat[ ,k-1], ...)

    # Should we stop?
    if (all(abs(grad.cur) < stopping.deriv)) {
      k <- k-1; break
    }

    # Move in the opposite direction of the grad
    xmat[ ,k] <- xmat[ ,k-1] - solve(hess.cur)%*%grad.cur
  }

  xmat <- xmat[ ,1:k] # Trim
  return(list(x = xmat[,k],
              xmat = xmat,
              k = k,
```

```
              minimum=f(xmat[,k],...)
  )
  )
}
```

Run Newton's method below:

```
Newton.method(neg.ll,x0=rep(0,5))$x
```

```
## [1]  0.489467146 -1.069402560 -0.046606074  0.009469987  0.008565829
```

```
Newton.method(neg.ll,x0=rep(0,5))$k
```

```
## [1] 8
```

Check the result with **glm()** (optional)

```
glm(Y~X1+X2+X3+X4,data=glm.data,family="poisson")
```

```
##
## Call:  glm(formula = Y ~ X1 + X2 + X3 + X4, family = "poisson", data = glm.data)
##
## Coefficients:
## (Intercept)           X1           X2           X3           X4
##    0.489467    -1.069403    -0.046606     0.009470     0.008566
##
## Degrees of Freedom: 99 Total (i.e. Null);  95 Residual
## Null Deviance:       199.2
## Residual Deviance: 108.8     AIC: 377.3
```

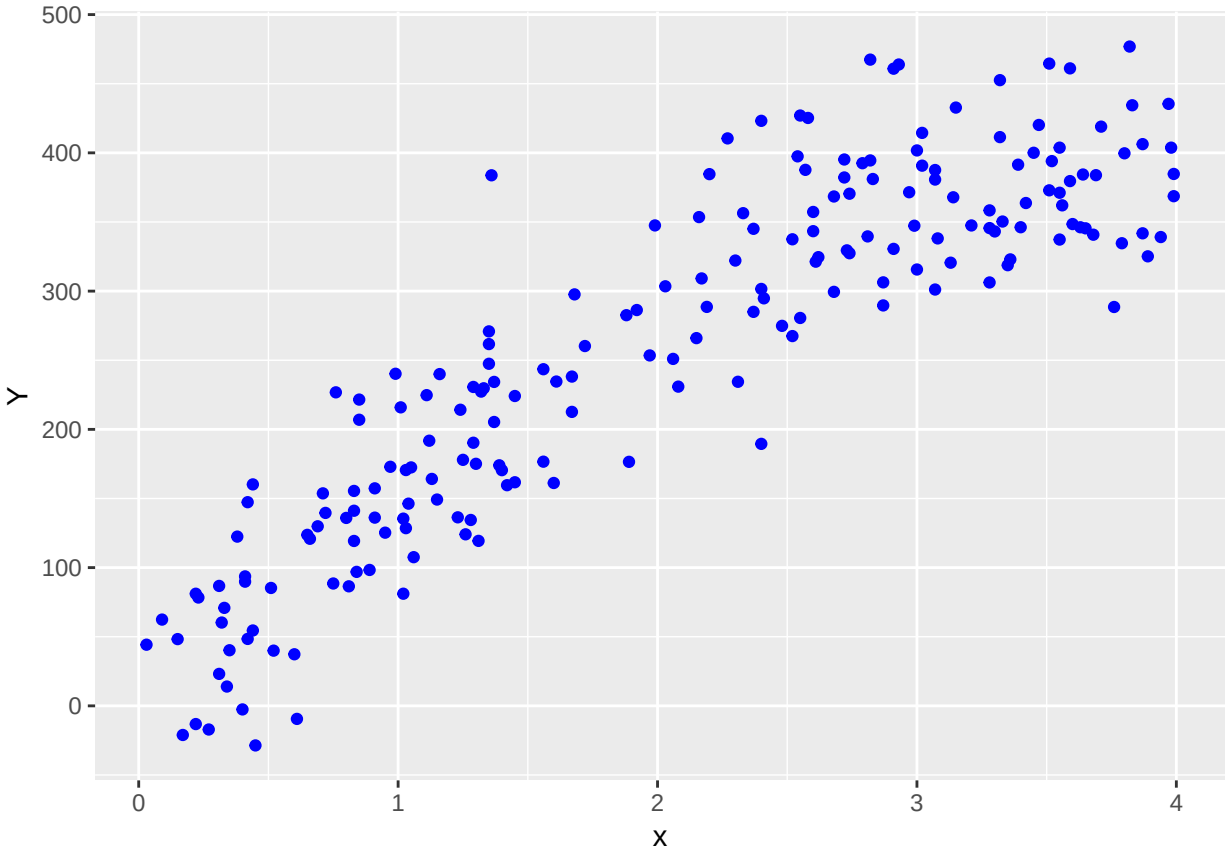
Part III: Cross Validadtion and Linear Regression [35 pts]

The goal of this section is to illustrate how the **degree** of a polynomial model can be thought of as a tuning parameter.

Perform the follwoing task:

5) [5 pts] Upload the dataset **finalexamtrain.csv** and plot Y versus x using **ggplot**. Change the points to blue in the plot.

```
data.train <- read.csv("finalexamtrain.csv")
library(ggplot2)
ggplot(data=data.train)+
  geom_point(mapping=aes(x=x,y=Y),col="blue")
```



6) [5 pts] In this setting, we regress the response variable Y against a single predictor x using five different models:

Linear Model: degree=1

$$Y_i = \beta_0 + \beta_0 x_i + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2)$$

Quadratic Model: degree=2

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2)$$

Cubic Model: degree=3

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2)$$

Quartic Model: degree=4

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2)$$

Quintic Model: degree=5

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 x_i^4 + \beta_5 x_i^5 + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2)$$

Fit each model using the training data. To make your life easier, I recommend using the Inhibit Interpretation function **I()**. Display the estimated coefficients for each model.

```
model.linear <- lm(Y~x,data=data.train)
model.quad <- lm(Y~x+I(x^2),data=data.train)
model.cubic <- lm(Y~x+I(x^2)+I(x^3),data=data.train)
model.quartic <- lm(Y~x+I(x^2)+I(x^3)+I(x^4),data=data.train)
model.quintic <- lm(Y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5),data=data.train)
```

```
model.linear$coefficients
```

```
## (Intercept)           x
##    58.35156    97.34397
```

```
model.quad$coefficients
```

```
## (Intercept)           x      I(x^2)
##   -11.10461   197.43481   -24.43306
```

```
model.cubic$coefficients
```

```
## (Intercept)           x      I(x^2)      I(x^3)
##    8.515224  146.827641    5.518712   -4.858338
```

```
model.quartic$coefficients
```

```
## (Intercept)           x      I(x^2)      I(x^3)      I(x^4)
##    14.12952   123.27218    30.79111   -14.42940     1.17410
```

```
model.quintic$coefficients
```

```
## (Intercept)           x      I(x^2)      I(x^3)      I(x^4)      I(x^5)
##    2.750585  188.784838  -72.142734   50.038779  -16.216959    1.687486
```

7) [10 pts] Consider the dataframe **finalexamtest.csv** that contains four validation (test) sets each consisting of 50 cases. Notice that the variable **ValSet** contains four levels. More specifically, the first 50 cases belong to **TestSet1**, the next 50 cases belong to **TestSet2**, . . . etc.

```
data.test <- read.csv("finalexamtest.csv")
head(data.test)
```

```
##      x         Y   ValSet
## 1 0.67  45.00438 TestSet1
## 2 3.23 386.60818 TestSet1
## 3 1.54 284.04469 TestSet1
## 4 2.42 391.53059 TestSet1
## 5 0.50   1.37337 TestSet1
## 6 1.18 262.60680 TestSet1
```

```
tail(data.test)
```

```
##        x        Y   ValSet
## 195 1.00 147.2985 TestSet4
## 196 3.12 296.8062 TestSet4
## 197 3.42 302.9193 TestSet4
## 198 2.09 263.6752 TestSet4
## 199 1.69 212.8365 TestSet4
## 200 1.65 303.2247 TestSet4
```

```
levels(data.test$ValSet)
```

```
## [1] "TestSet1" "TestSet2" "TestSet3" "TestSet4"
```

Write a function named **test.error** that inputs a dataframe and outputs the **test error** for each model. The function should give four predictions errors corresponding to each model, i.e., linear, quadratic, cubic, quartic, and quintic. Note that you are still using the trained models from Part (6). You are allowed to hard code some of this function. Test the **test.error* function on the full training data.

Hint: y.test <- predict(model,newdata = data.test["x"])

```
prediction.error <- function(data) {

  y.linear <- predict(model.linear,newdata = data["x"])
  y.quad <- predict(model.quad,newdata = data["x"])
  y.cubic <- predict(model.cubic,newdata = data["x"])
  y.quartic <-predict(model.quartic,newdata = data["x"])
  y.quintic <- predict(model.quintic,newdata = data["x"])

  return(c(linear=mean((y.linear-data["Y"])^2),
           quad=mean((y.quad-data["Y"])^2),
           cubic=mean((y.cubic-data["Y"])^2),
           quartic=mean((y.quartic-data["Y"])^2),
           quintic=mean((y.quintic-data["Y"])^2)
           )
  )
}
# Test
prediction.error(data.train)
```

```
##   linear     quad    cubic  quartic  quintic
## 3211.680 2485.083 2452.323 2450.637 2446.708
```

8) [10 pts] Use the **Split/Apply/Combine** model to compute the test error for each test set: **TestSet1**, **TestSet2**, **TestSet3**, **TestSet4**. You can exhaustively perform this task for partial credit. Display the test errors for each test set and each model. Also display the average test error over each validation set.

```
# Split/Apply/Combine
test.split <- split(data.test,data.test$ValSet)
sapply(test.split,prediction.error)
```

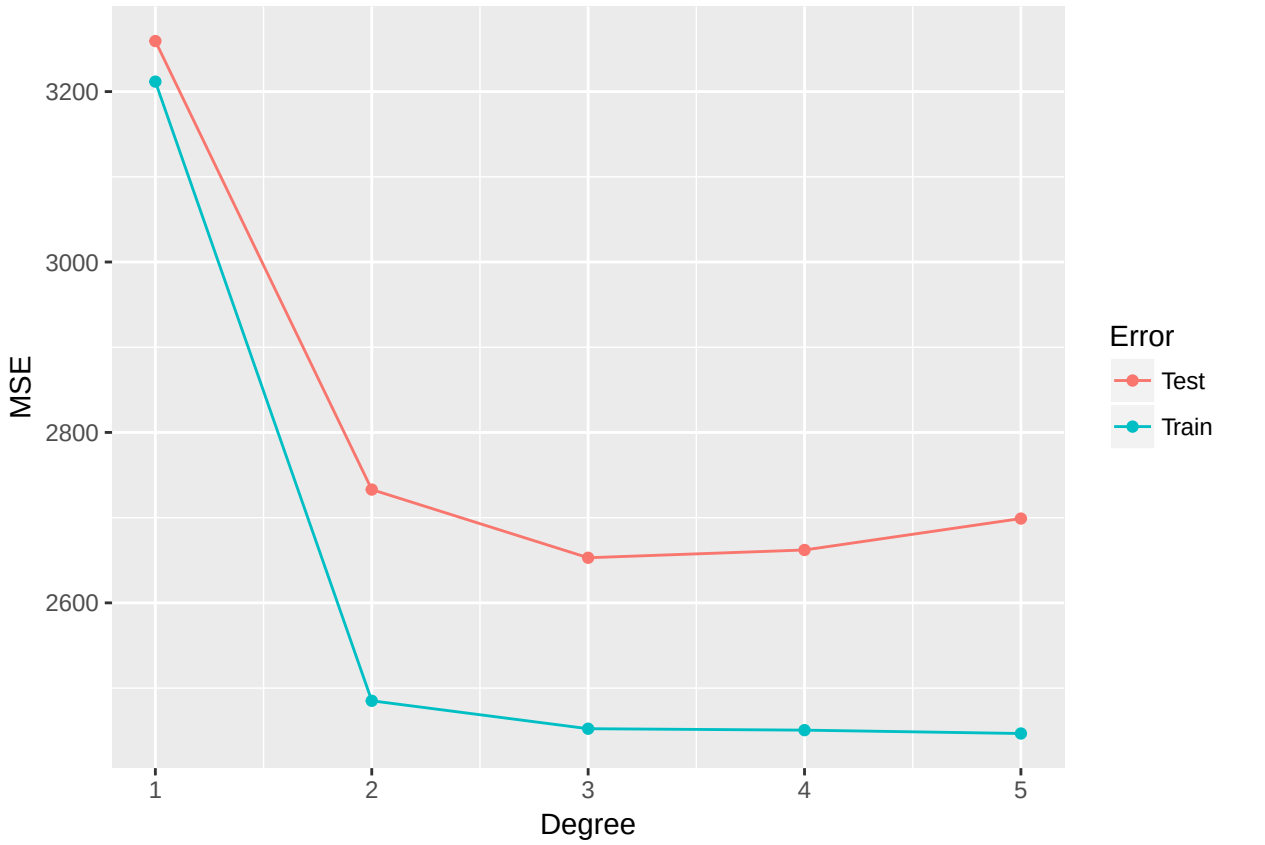
```
##         TestSet1 TestSet2 TestSet3 TestSet4
## linear  3611.566 3466.068 3434.612 2524.945
## quad    3324.065 2832.745 3087.183 1687.839
## cubic   3068.520 2797.414 2951.258 1794.613
## quartic 3074.860 2804.055 2973.750 1795.619
## quintic 3091.755 2828.324 3068.169 1807.527
```

```
# Average test error
apply(sapply(test.split,prediction.error),1,mean)
```

```
##   linear     quad    cubic  quartic  quintic
## 3259.298 2732.958 2652.951 2662.071 2698.944
```

9) [5 pts] Create a plot (base R or ggplot) showing both the **average test error** and the **training error** as a function of the polynomial's **degree**. Briefly comment on this plot.

```
error_df <- data.frame(Degree=c(1:5,1:5),
                       Error=c(rep("Test",5),rep("Train",5)),
                       MSE=c(apply(sapply(test.split,prediction.error),1,mean),
                             prediction.error(data.train)
                             )
                       )
# Plot
ggplot(data=error_df)+
   geom_point(mapping=aes(x=Degree,y= MSE,color=Error))+
   geom_line(mapping=aes(x=Degree,y= MSE,color=Error))
```



The linear degree=1 model does a poor job on both the training and test datasets. The test error achieves its minimum at degree 2. The training error continues to decrease as the degree increases. As the degree increases, we expect for the test error to increase and the training error to decrease.