GR5206 Final Exam

Yuhao Wang and yw3204

Dec. 14, 2018

The STAT GR5206 Fall 2018 Final 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 exam. This includes emailing fellow students, using WeChat and other similar forms of communication.

Before the exam, the students should **turn off** their cellphones and pass them to the left side of each row. At the same time, please **close** the mailbox and **log out** WeChat and all the other apps for messaging and chatting.

If there is any suspicion of one or more 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 1:10PM and 1:20PM for group 1 and group 2 respectively. Group 1 and group 2 students are required to submit both the .pdf and .Rmd files on GradeScope or Canvas (or .html if you must) by 3:10PM and 3:20PM respectively. Late exams will not be accepted.

If you have trouble submitting your final, please send the .pdf and .Rmd file to our course mailing list: `gr5206_course_staff@columbia.edu`.

Part 1: Simulation [16 pts]

In this section, we consider a Dirichlet distribution, which is a multivariate generalization of the beta distribution. Here we assume that the K-dimensional random variable $X = (X_1, \ldots, X_K)$ satisfying $0 < X_i < 1$ for $1 \leq i \leq K$ and $\sum_{i=1}^{K} X_i = 1$ is governed by the probability density $f(\mathbf{u})$, where $\mathbf{u} = (u_1, \ldots, u_K)^\top$. f is defined by

$$\begin{aligned} f(\mathbf{u}) &= f(\mathbf{u}; \alpha) \\ &= \frac{1}{B(\alpha)} \prod_{i=1}^{K} u_i^{\alpha_i - 1}, \end{aligned}$$

where $0 < u_1, \ldots, u_K < 1$ and $\sum_{i=1}^{K} u_i = 1$, $\alpha = (\alpha_1, \ldots, \alpha_K)^\top$ is a vector of parameters satisfying $\alpha_i > 0$ for $1 \leq i \leq K$, and $B(\alpha) = (\prod_{i=1}^{K} \Gamma(\alpha_i))/\Gamma(\sum_{i=1}^{K} \alpha_i)$. Note that $\Gamma(\cdot)$ is the gamma function, and in `R` we can use the build-in function `gamma()` to calculate the value.

In this part, we assume $K = 3$, and $\alpha_1 = \alpha_2 = \alpha_3 = 2$. This distribution is essentially a bivariate one, as $X_3 = 1 - X_1 - X_2$. Therefore, drawing samples from this Dirichlet distribution is equivalent to drawing samples of (X_1, X_2).

The heatmap of the density (joint density of (X_1, X_2)) is shown below.

```
### DO NOT run this chunk of code during the exam
library(MCMCpack)
```

```
## Loading required package: coda
```

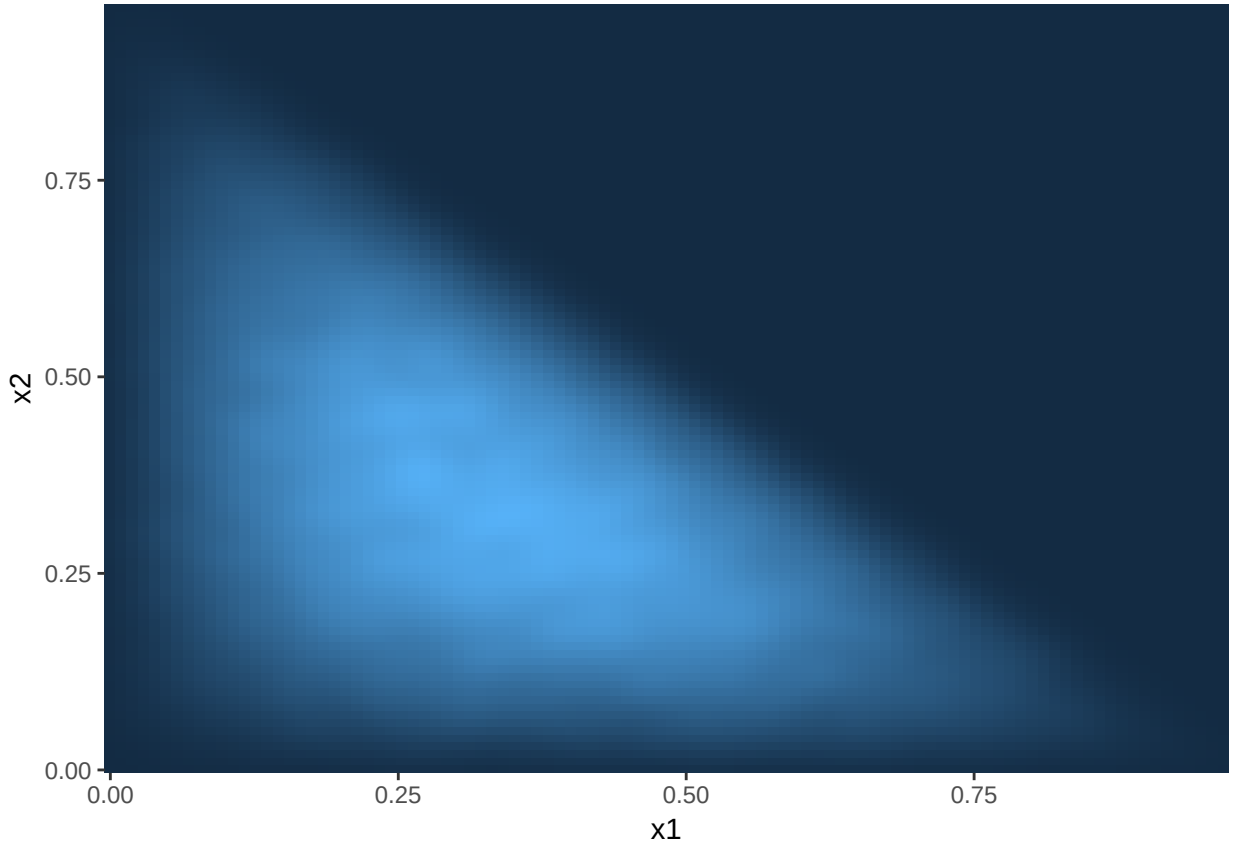
```
## Loading required package: MASS
```

```
## ##
## ## Markov Chain Monte Carlo Package (MCMCpack)
```

```
## ## Copyright (C) 2003-2018 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
```

```
## ##
## ## Support provided by the U.S. National Science Foundation
## ## (Grants SES-0350646 and SES-0350613)
## ##
```

```r
alpha<-rep(2, 3)
X<-rdirichlet(1e5, alpha)
colnames(X)<- c("x1", "x2", "x3")
X <- data.frame(X)

library(ggplot2)
ggplot(X, aes(x=x1, y=x2) ) +
  stat_density_2d(aes(fill = ..density..), geom = "raster", contour = FALSE) +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0)) +
  theme(
    legend.position='none'
  )
```



Perform the following tasks:

1) [12 pts] Use the **accept-reject** algorithm to simulate from this bivariate distribution of (X_1, X_2) through the following three steps:
   1.i Clearly identify an **easy to simulate** distribution $g(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2)^\top$.
   1.ii Identify a **suitable** value of γ such that your envelope function $e(\mathbf{x})$ satisfies

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

where $\mathbf{x} = (x_1, x_2)^\top$. Note that you need choose γ so that $e(\mathbf{x})$ is close to $f(\mathbf{x})$. Hint: if you have difficulty finding such a γ, just try to use a small value to ensure $f(\mathbf{x}) \leq e(\mathbf{x})$ for partial credit.

1.iii Simulate 10,000 draws from the bivariate normal distribution using the **accept-reject** algorithm. Display the first 20 simulated values. Also, using **ggplot**, construct a scatter plot of the samples. Compare it to the heatmap.

1.i) [2 pts]

```
# Solution goes here
g <- function(x) {
  return(dunif(x[1], 0, 1) * dunif(x[2], 0, 1))
}
```

1.ii) [4 pts]

```
# Solution goes here
f <- function(x) {
  return(ddirichlet(c(x, 1-x[1]-x[2]), alpha))
}

# determine gamma
# nf <- function(x) {
#   return(-ddirichlet(c(x, 1-x[1]-x[2]), alpha))
# }
# nlm(f, c(0.1, 0.2))

e <- function(x, ga = 1/4.5) {
  return(g(x)/ga)
}
```

We choose γ as 1/4.5 since the maximum of f(x) is 4.44.

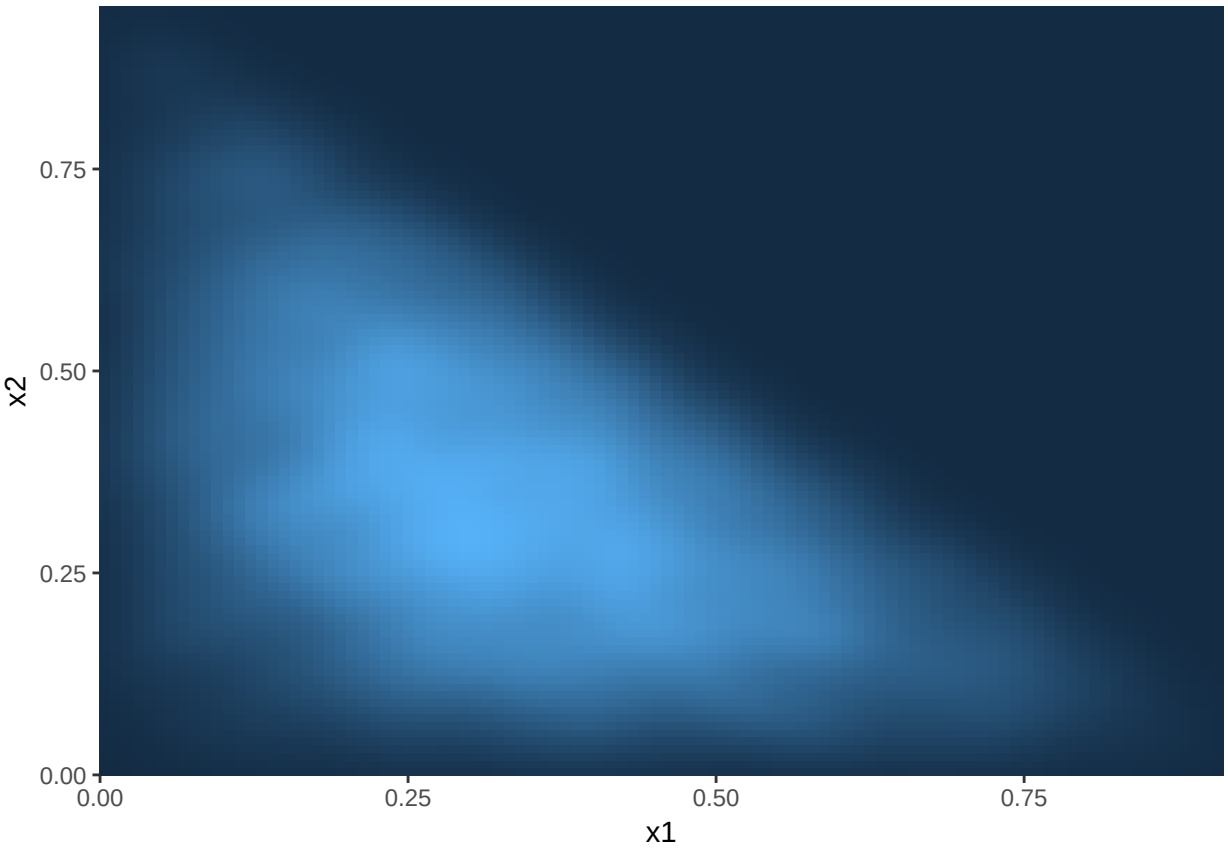
1.iii) [6 pts] The **Accept-Reject** algorithm is coded below:

```
# Solution goes here
k <- 0
bi <- matrix(nrow = 10000, ncol = 2)
while(k < 10000) {
  x <- runif(2, 0, 1)
  if(x[1] + x[2] < 1) {
    u <- runif(1, 0, 1)
    if(u * e(x) < f(x)) {
      k <- k + 1
      bi[k, ] <- x
    }
  }
}
```

Plot:

```
# Solution goes here
df_bi <- as.data.frame(bi)
names(df_bi) <- c("x1", "x2")

ggplot(df_bi, aes(x=x1, y=x2) ) +
  stat_density_2d(aes(fill = ..density..), geom = "raster", contour = FALSE) +
  scale_x_continuous(expand = c(0, 0)) +
  scale_y_continuous(expand = c(0, 0)) +
```

```
  theme(
    legend.position='none'
  )
```



2) [4 pts] Slightly change the **Accept-Reject** algorithm from Part (1.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? What is the theoretical value of the acceptance rate?

```
# Solution goes here
k <- 0
all <- 0
bi <- matrix(nrow = 10000, ncol = 2)
while(k < 10000) {
  x <- runif(2, 0, 1)
  if(x[1] + x[2] < 1) {
    u <- runif(1, 0, 1)
    if(u * e(x) < f(x)) {
      k <- k + 1
      bi[k, ] <- x
    }
  }
  all <- all+1
}

# acceptance rate
k/all
```

```
## [1] 0.2205023
```

The total time is about 44000 and the acceptance time is 10000. The rate is about 0.22 while the theretical one should be around 1/4.5. They are quite close.

Part 2: Penalized logistic regression [12 pts + 4 extra pts]

We look at the voting records of the US Congress of 2017. The votes were compiled from http://clerk.house.gov.

We have the votes (or absence of a vote) for each member of the House of Representatives on over 709 roll call votes in 2017 with 424 members having all votes. Voting results are summarized in `2017_cleaned_votes.csv`. We want to use them to predict whether a member is Republican or Democrat. We consider a L_2-penalized model, and use the first 100 votes as predictors in order to save computational time.

```
votes <- read.table("2017_cleaned_votes.csv", header = TRUE, sep = ";")
dim(votes)
```

```
## [1] 424 710
```

```
X <- votes[, 2:101]
```

In the data set, the first column is the party of each member, and the other columns are voting results over 709 roll calls (roll call 2 result is missing). "1" represents "Yes", "-1" represents "No", and "0" means "Not Voting".

3) [2 pts] Create a new binary variable `y`, which takes value 1 if the party is Republican ("R" in the data), and 0 if it is "Democrat" ("D").

```
# Solution goes here
y <- ifelse(votes$party == "R", 1, 0)
```

Insteading of minimizing negative log-likehood, we minimize the penalized objective function for this problem

$$l(\beta_0, \beta) = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i(\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}) - \log(1 + \exp(\beta_0 + \sum_{j=1}^{p}\beta_j x_{ij}))\right) + \lambda^* \sum_{j=1}^{p}\beta_j^2, \tag{1}$$

where $\beta = (\beta_1, \ldots, \beta_p)^\top$ is a vector. x_{ij} is the voting result of the ith member for jth roll call.

4) [4 pts] Write a function `neg.ll.l2`. The function should take β_0 (a scalar), β (a vector of lenght 100) and λ (a scalar) as inputs, and return the value of $l(\beta_0, \beta)$ with $\lambda^* = \lambda$. In your function, you can combine β_0 and β together as a single argument, which is a vector of 101 parameters.

```
# Solution goes here
neg.ll.l2 <- function(beta, lamd) {
  beta_0 <- beta[1]
  beta_1 <- beta[2:101]
  Y <- y
  X.mat <- as.matrix(X)
  linear.component <- beta_0 + X.mat%*%beta_1
  p.i <- exp(linear.component)/(1+exp(linear.component))
  pl <- lamd * sum(beta_1 ** 2)
  return(-mean(dbinom(Y, size=1, prob=p.i,log=TRUE)) + pl)
}
```

5) [2 pts] Minimize l with $\lambda^* = 0.12$. You can use any base `R` function for optimization, such as `nlm()`. What are the corresponding values of β_0 and β at the minimum?

```r
# Solution goes here
nlm(neg.ll.l2, rep(0, 101), lamd = 0.12)
```

```
## $minimum
## [1] 0.0437469
##
## $estimate
##   [1] -5.368672e-02 -5.005668e-07  5.652121e-02  6.050883e-02 -6.038522e-02
##   [6]  5.326267e-02 -5.540779e-02  5.141397e-02  5.318021e-02  5.291918e-02
##  [11]  1.724587e-02  5.356328e-02 -5.210134e-02 -5.257560e-02 -5.307916e-02
##  [16] -5.277659e-02 -5.725977e-02 -5.201875e-02 -5.738116e-02 -5.656783e-02
##  [21]  5.588418e-02 -5.738116e-02  5.286042e-02  1.633295e-03  1.244692e-03
##  [26]  5.358753e-02  5.421880e-02  8.538014e-03 -4.614716e-02 -4.960329e-02
##  [31]  2.263712e-02  5.368003e-02  5.285463e-02 -2.192811e-03  5.246636e-02
##  [36]  3.889770e-02 -4.886787e-02 -5.275398e-02 -5.346960e-02 -4.955192e-02
##  [41] -4.804169e-02 -5.360790e-02 -5.170936e-02 -5.670829e-02  5.101296e-02
##  [46] -5.536023e-02 -5.374750e-02 -5.509482e-02 -5.449580e-02 -5.509622e-02
##  [51]  5.113616e-02  5.498840e-02 -5.914902e-02  5.073222e-02  5.516170e-02
##  [56]  5.677755e-02 -4.065652e-02  5.065477e-02  3.826416e-02  7.697900e-04
##  [61]  1.394240e-03  5.523811e-02  5.174758e-02 -5.527684e-02  5.035779e-02
##  [66]  6.395829e-03 -1.643062e-03  5.373270e-02  5.718245e-02  5.947236e-02
##  [71]  5.944429e-02  4.687855e-02  4.838034e-02  5.496067e-02  5.544239e-02
##  [76]  5.124018e-02  4.869608e-02  4.548127e-02  3.083194e-03  3.616045e-03
##  [81]  5.707163e-02  5.678409e-02  4.989727e-02  5.719496e-02  5.462137e-02
##  [86]  2.835321e-03  2.779776e-03  5.610329e-02  5.576889e-02  5.617108e-02
##  [91]  5.618388e-02  4.855187e-02  5.962450e-02  5.936728e-02  5.807909e-02
##  [96]  5.385906e-02  5.555911e-02  4.777942e-02  5.264864e-02  2.388676e-03
## [101]  5.454625e-02
##
## $gradient
##   [1]  6.529499e-09 -1.387779e-10  2.326056e-07  3.172185e-07 -3.060330e-07
##   [6]  6.212184e-07 -5.636464e-08 -3.697528e-07  4.255207e-07  4.538314e-07
##  [11]  5.871484e-07 -3.097383e-07  6.215028e-07 -1.411163e-07  4.069245e-07
##  [16] -3.395478e-07  1.230266e-08 -2.832040e-07  3.975986e-09  3.726186e-09
##  [21] -1.155048e-07  3.975986e-09 -3.677614e-07  1.176836e-07  9.523632e-08
##  [26]  1.966066e-07  1.911318e-07  9.533069e-07 -8.905654e-07 -3.888140e-07
##  [31]  2.132322e-08 -3.034170e-07 -6.644685e-08  5.128398e-07 -8.704149e-08
##  [36] -2.559897e-07 -7.563394e-10 -2.472536e-07  3.562220e-07 -4.728856e-08
##  [41]  4.482525e-09  3.497341e-07  1.929984e-07  1.337749e-07 -4.277551e-07
##  [46] -2.135445e-07 -2.617420e-07 -2.136902e-07 -1.249834e-07 -2.446099e-07
##  [51] -1.448702e-07  2.007908e-07  2.081946e-07 -2.029280e-07 -1.872530e-07
##  [56] -1.597264e-07  4.110878e-07  3.355718e-07  3.187728e-08  3.499978e-07
##  [61]  1.391942e-07  3.088363e-07 -5.842549e-08 -4.286710e-07 -6.555173e-08
##  [66] -1.169898e-08  5.414003e-07 -1.927764e-07  4.303502e-08 -1.960238e-08
##  [71] -5.387357e-08 -9.665879e-09 -8.695128e-07  2.365746e-07  2.232520e-07
##  [76] -1.831452e-07 -1.096068e-07 -3.675046e-07 -1.556463e-07 -1.957740e-07
##  [81]  3.359812e-08  9.575674e-10 -3.979803e-07  9.255791e-08  1.356068e-07
##  [86] -1.987993e-07 -2.835371e-07 -1.629946e-07 -1.685735e-07 -1.503658e-07
##  [91] -6.551010e-08 -6.018797e-07  1.772194e-08 -3.041317e-08 -2.901707e-07
##  [96]  7.271267e-08 -4.780204e-07 -7.373824e-07 -3.655132e-07 -1.179820e-07
## [101]  3.227696e-07
##
## $code
## [1] 1
```

```
##
## $iterations
## [1] 27
```

```
beta <- nlm(neg.ll.l2, rep(0, 101), lamd = 0.12)$estimate
beta
```

```
##   [1] -5.368672e-02 -5.005668e-07  5.652121e-02  6.050883e-02 -6.038522e-02
##   [6]  5.326267e-02 -5.540779e-02  5.141397e-02  5.318021e-02  5.291918e-02
##  [11]  1.724587e-02  5.356328e-02 -5.210134e-02 -5.257560e-02 -5.307916e-02
##  [16] -5.277659e-02 -5.725977e-02 -5.201875e-02 -5.738116e-02 -5.656783e-02
##  [21]  5.588418e-02 -5.738116e-02  5.286042e-02  1.633295e-03  1.244692e-03
##  [26]  5.358753e-02  5.421880e-02  8.538014e-03 -4.614716e-02 -4.960329e-02
##  [31]  2.263712e-02  5.368003e-02  5.285463e-02 -2.192811e-03  5.246636e-02
##  [36]  3.889770e-02 -4.886787e-02 -5.275398e-02 -5.346960e-02 -4.955192e-02
##  [41] -4.804169e-02 -5.360790e-02 -5.170936e-02 -5.670829e-02  5.101296e-02
##  [46] -5.536023e-02 -5.374750e-02 -5.509482e-02 -5.449580e-02 -5.509622e-02
##  [51]  5.113616e-02  5.498840e-02 -5.914902e-02  5.073222e-02  5.516170e-02
##  [56]  5.677755e-02 -4.065652e-02  5.065477e-02  3.826416e-02  7.697900e-04
##  [61]  1.394240e-03  5.523811e-02  5.174758e-02 -5.527684e-02  5.035779e-02
##  [66]  6.395829e-03 -1.643062e-03  5.373270e-02  5.718245e-02  5.947236e-02
##  [71]  5.944429e-02  4.687855e-02  4.838034e-02  5.496067e-02  5.544239e-02
##  [76]  5.124018e-02  4.869608e-02  4.548127e-02  3.083194e-03  3.616045e-03
##  [81]  5.707163e-02  5.678409e-02  4.989727e-02  5.719496e-02  5.462137e-02
##  [86]  2.835321e-03  2.779776e-03  5.610329e-02  5.576889e-02  5.617108e-02
##  [91]  5.618388e-02  4.855187e-02  5.962450e-02  5.936728e-02  5.807909e-02
##  [96]  5.385906e-02  5.555911e-02  4.777942e-02  5.264864e-02  2.388676e-03
## [101]  5.454625e-02
```

The estimate of $bets_0$ and $bets$ are given above.

5) [4 pts] Based on the fitted model (1), for each member what is your prediction of her/his party. For how many members, you can make the correct prediction?

```
# Solution goes here
y_tmp <- as.matrix(cbind(rep(1, 424), X)) %*% beta
y_hat <- ifelse(y_tmp>0, 1, 0)
mean(y == y_hat)
```

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

The correct prediction is 100%.

6) [4 extra pts] Use cross-validation to estimate test error (under 0-1 loss) of the model (1) with $\lambda^* = 4$.

```
# Solution goes here
# cross validation
# rep.vec <- rep(1:106, 8)
# re.order <- sample(rep.vec, replace=F)
#
# test.error <- NULL
# for (k in 1:106) {
#     # Choose test and training data from kth fold
#     test  <- votes[re.order==k, ]
#     train <- votes[re.order!=k, ]
#
#     n.test <- nrow(test)
#
```

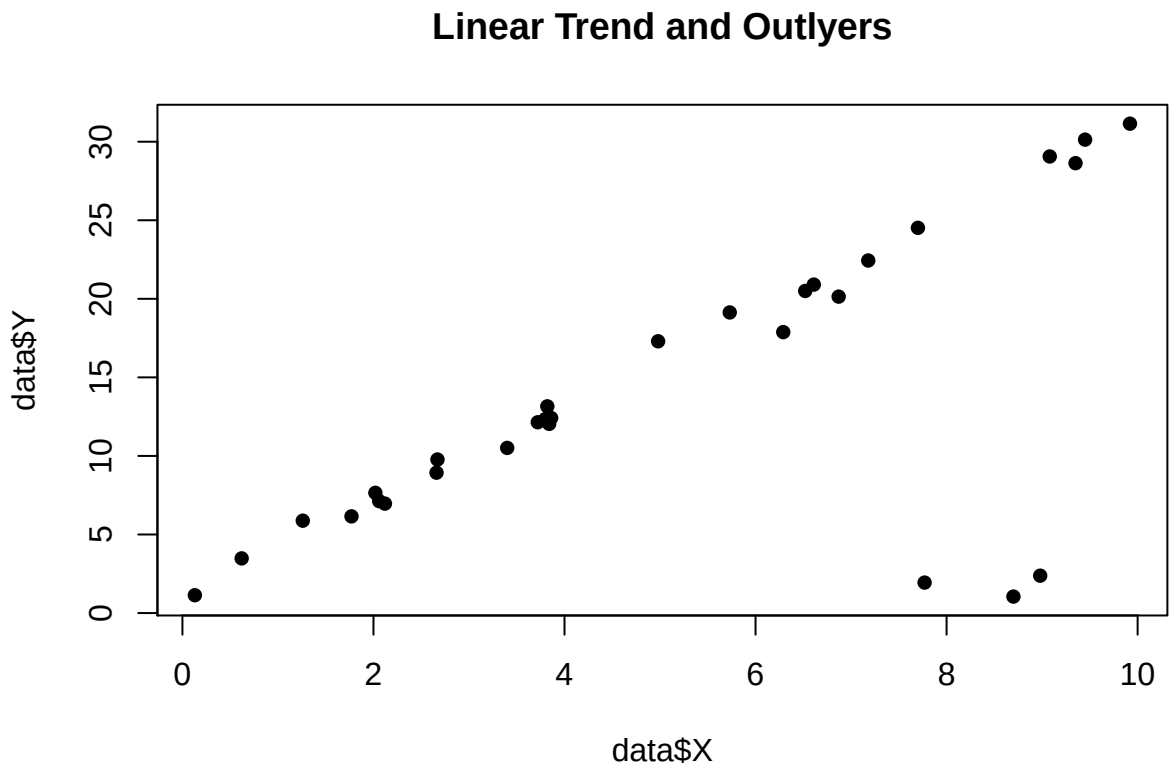
```
#     predictions <- rep(NA, n.test)
#
#     test.error[k] <- sum(predictions != test$Direction)/n.test
# }
#
# mean(test.error)
```

Part 3: Bootstrap and robust estimation [12 pts + 4 extra pts]

Problem statement:

Consider the following toy dataset relating response variable Y with covariate X. Note that this dataset is an extreme case of how traditional least squares regression fails to capture the trend of the the data in the presences of outlying observations.

```
data <- read.csv("Problem1.csv")
plot(data$X,data$Y, pch=16, main="Linear Trend and Outlyers")
```



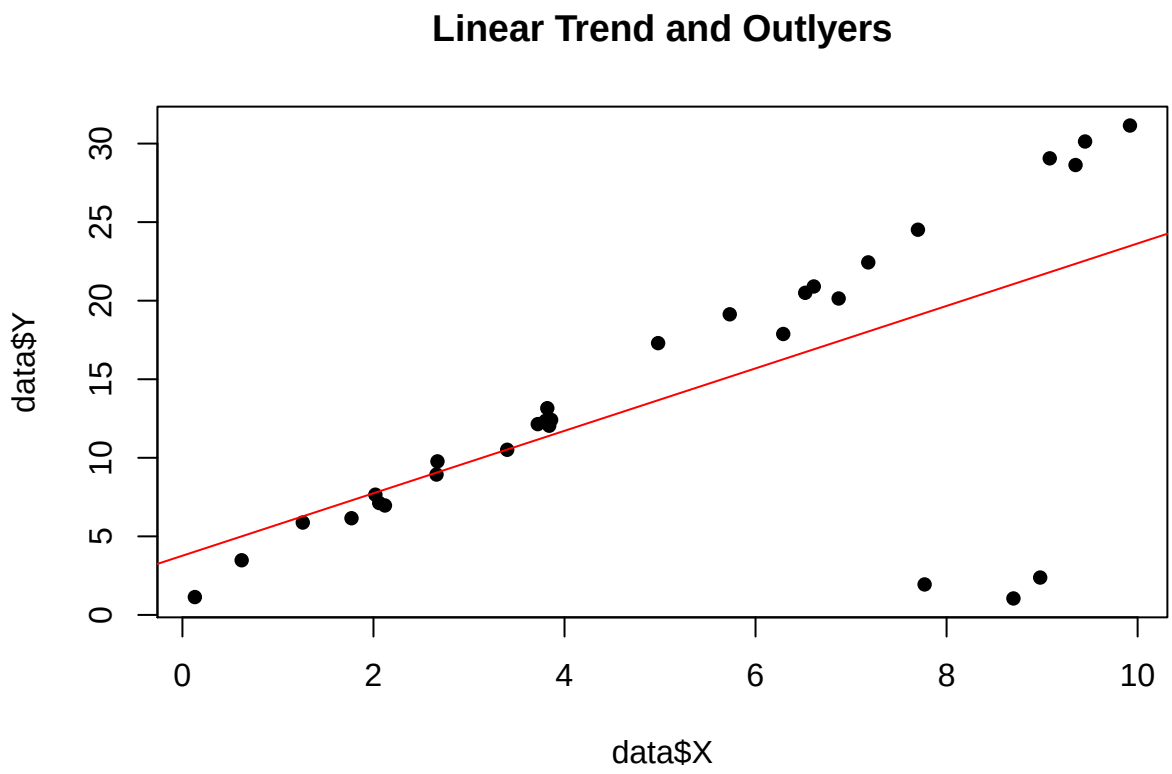
7) [4 pts] Fit a regular linear regression to the above dataset and plot the line of best fit in red.

Also remove the three outlying points and fit the linear model on the remaining 27 cases. Plot this new line of best fit on the same graph as the first model. Create a legend on the plot describing each line. Note: remove the points corresponding to $Y = 1.05, 1.94, 2.38$.

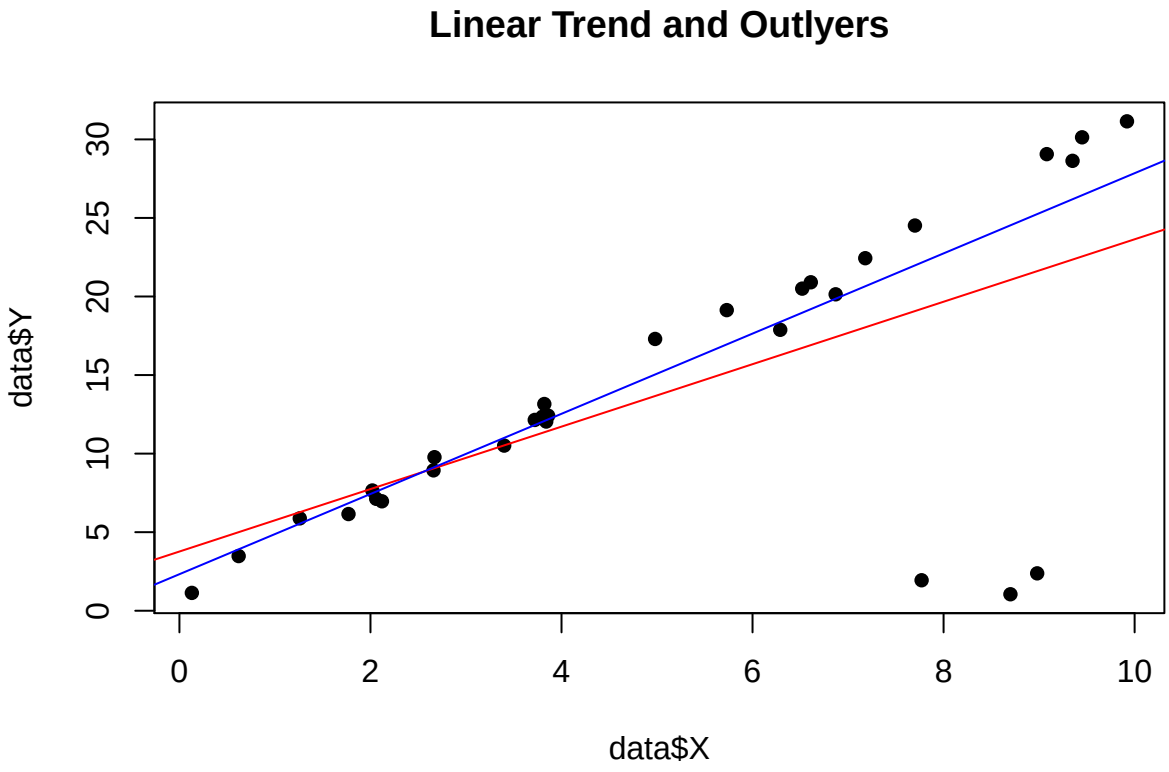
Comment on any interesting features from the graph and estimated models.

```
# Solution goes here
lm1 <- lm(Y~X, data)
```

```r
plot(data$X,data$Y, pch=16, main="Linear Trend and Outlyers")
abline(lm1, col = "red")
```




```r
n_data <- data[data$Y != c(1.05, 1.94, 2.38), ]
lm2 <- lm(Y~X, n_data)
plot(data$X,data$Y, pch=16, main="Linear Trend and Outlyers")
abline(lm1, col = 2)
abline(lm2, col = "blue")
```



Problem 8) set-up:

To fit the linear model, we minimize the total squared Euclidean distance between Y and a linear function of X, i.e., minimize the expression below with respect to β_0, β_1.

$$S(\beta_0, \beta_1) = \sum_{i=1}^{n} (Y_i - (\beta_0 + \beta_1 X_i))^2$$

From the above fit, we see that the outlying Y values are influencing the estimated line, consequently, the linear fit is being pulled down and is not covering the full trend of the data. To remedy this problem, we can perform a robust estimation procedure. More specifically, instead of minimizing squared Euclidean distance (squared loss), we can minimize Huber loss. To estimate our robust model, we minimize $Q(\beta_0, \beta_1)$ with respect to β_0, β_1:

$$Q(\beta_0, \beta_1) = \sum_{i=1}^{n} f(Y_i - (\beta_0 + \beta_1 X_i)), \tag{2}$$

where

$$f(u) = \begin{cases} u^2, & -1 \leq u \leq 1 \\ 2|u| - 1, & u < -1 \text{ or } u > 1 \end{cases}$$

The goal of the next exercise is to write a robust estimation procedure for the coefficients β_0, β_1. In class we performed univariate gradient descent. On this exam, you can use the R base function **nlm()** to perform the optimization task.

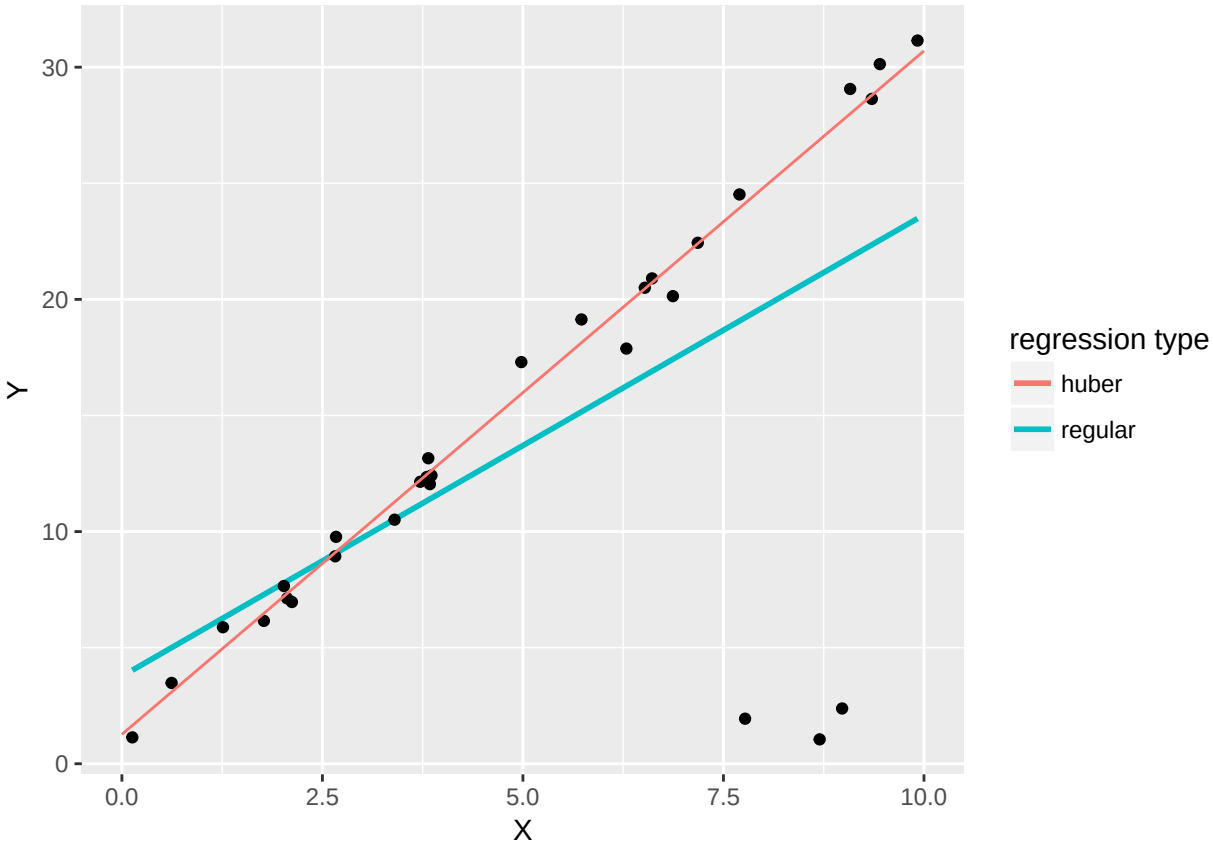
8) [4 pts] Write a `R` function **Q** which computes the Huber loss as a function of the vector $c(\beta_0, \beta_1)$. Note that the Huber loss is defined in Equation (2). This exercise is having you create an objective function $Q(\beta_0, \beta_1)$ so that you can run an optimization procedure later on. Test your function at the point **c(0,0)**.

```r
# Solution goes here
Q <- function(b) {
  lin <- data$Y - (b[1] + b[2] * data$X)
  tmp <- ifelse(abs(lin) <= 1, lin**2, 2*abs(lin)-1)
  return(sum(tmp))
}

Q(c(0, 0))
```

```
## [1] 803.7536
```

9) [4 pts] Optimize Huber loss Q using the **nlm()** function. Use the starting point **c(0,0)** and display your robust estimates. Use `ggplot()`, plot the estimated robust linear model and include the regular least squares regression line on the plot. Create a legend and label the plot appropriately.

```r
# Solution goes here
est <- nlm(Q, c(0, 0))$estimate

ggplot() + geom_smooth(data = data, aes(x = X, y = Y, color = "regular"), method =  "lm", se = F) +
  geom_point(data = data, aes(x = X, y = Y)) +
  geom_line(aes(x = seq(0, 10), y = est[1] + est[2] * seq(0, 10), color = "huber")) +
  labs(color = "regression type")
```



10) [4 extra pts] As statisticians, we must also construct confidence intervals on our parameter estimates to gauge how precise these estimates actually are. Recall the traditional parametric regression model:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i, \quad i = 1, 2, \ldots, n, \quad \epsilon_i \overset{iid}{\sim} N(0, \sigma^2)$$

When using least squares estimation, the normal-error structure ($\epsilon_i \overset{iid}{\sim} N(0, \sigma^2)$) allows us to construct

parametric confidence intervals for the parameters β_0, β_1. This is easily done in **R**. The code below constructs 95% intervals for the coefficients.

```
round(confint(lm(Y~X,data=data),level=.95),4)
```

```
##               2.5 % 97.5 %
## (Intercept) -1.6034 9.1418
## X            1.0711 2.9032
```

Notice from the above output, the slope is almost not within the range of what we expect. Clearly the outliers are impacting our least squares estimators. In the presence of our outlying observations, the normal error structure is clearly not correct. To approximate the correct distribution, we will apply the bootstrap procedure on the robust estimated coefficients computed by minimizing $Q(\beta_0, \beta_1)$.

Run a bootstrap on the robust estimation procedure. Note that this is similar to the regression bootstrap but you will be estimating the parameters by minimizing $Q(\beta_0, \beta_1)$. Use $B = 1000$ bootstrap iterations and confidence level 95%. Use the regular bootstrap intervals (not percentile intervals) and compare your bootstrapped solution to the parametric model.

```
# Solution goes here
```