A2

Yuqi Gu

6/9/2024

QUESTION 1: Classification with Logistic Regression

(a)

$$\begin{aligned}
l(\alpha,\beta;P) &= \ln\left(L(\alpha,\beta;P)\right)\\
&= \ln\left(\prod_{u\in P} e^{y_u(\alpha+\beta x_u)}(1+e^{\alpha+\beta x_u})^{-1}\right)\\
&= \sum_{u\in P}\left(\log\left(e^{y_u(\alpha+\beta x_u)}\right)+\log\left((1+e^{\alpha+\beta x_u})^{-1}\right)\right)\\
&= \sum_{u\in P}\left(y_u(\alpha+\beta x_u)-\log(1+e^{\alpha+\beta x_u})\right)
\end{aligned}$$

(b)

```r
logistic_nll <- function(theta, x, y) {
  alpha <- theta[1]
  beta <- theta[2]
  aplusbx <- alpha+beta * x
  log_likelihood <- sum(y*aplusbx - log(1+exp(aplusbx)))
  return(-log_likelihood)
}
```

Since maximizing the likelihood means minimizing the negative likelihood, it is important to return the negative log-likelihood so that we could use optim() which will minimize the objective function, the negative log-likelihood we get from logistic_nll().

(c)

```r
setwd('/Users/yuki/Desktop/STAT 341/Assignments/A2')
kvd <- read.csv("kendrick_v_drake.csv")

x <- kvd$Plays/1000000
kvd$Main.Artist = c(rep("Kendrick", 128), rep("Drake", 141))
y <- 1 * (kvd$Main.Artist == "Kendrick")

theta_0 <- c(0, 0)
```

```r
result <- optim(par=theta_0, fn=logistic_nll, x=x, y=y)

theta <- result$par
theta
```

```
## [1]  0.073138048 -0.001864294
```

```r
# check
glm(y ~ x, family = binomial(link = logit))
```

```
##
## Call:  glm(formula = y ~ x, family = binomial(link = logit))
##
## Coefficients:
## (Intercept)            x
##    0.072744    -0.001864
##
## Degrees of Freedom: 268 Total (i.e. Null);  267 Residual
## Null Deviance:       372.3
## Residual Deviance: 366.1     AIC: 370.1
```

(d)

```r
alpha_hat <- theta[1]
beta_hat <- theta[2]

cprob <- function(x, alpha, beta) {
  apbx <- alpha+beta*x
  return(exp(apbx) / (1 + exp(apbx)))
}

p_hat <- cprob(x, alpha_hat, beta_hat)

c <- 0.5
y_hat <- 1*(p_hat>=c)

cm <- table(y, y_hat)
print(cm)
```

```
##    y_hat
## y    0  1
##   0 74 67
##   1 32 96
```

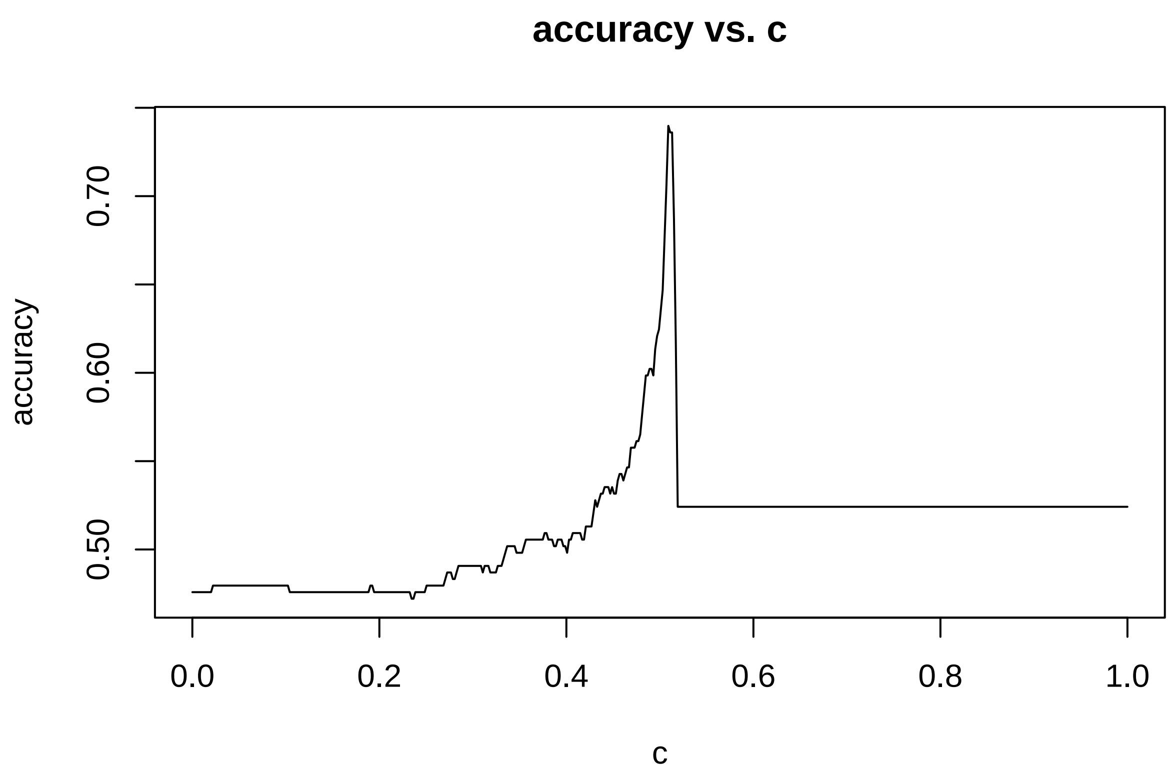
```r
TN <- cm[1, 1]
FP <- cm[1, 2]
FN <- cm[2, 1]
TP <- cm[2, 2]
accuracy <- (TN + TP)/(TN + FP + FN + TP)
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.6319703
```

		Prediction	
		$\hat{y}_u = 1$	$\hat{y}_u = 0$
Truth	$y_u = 1$	96	32
	$y_u = 0$	67	74

(e)

```r
c <- seq(from = 0, to = 1, length.out = 500)

calc.acc <- function(y, ystar) {
  TN <- sum((ystar == 0) & (y == 0))
  TP <- sum((ystar == 1) & (y == 1))
  FP <- sum((ystar == 1) & (y == 0))
  FN <- sum((ystar == 0) & (y == 1))
  accuracy <- (TN + TP)/(TN + FP + FN + TP)
  return(accuracy)
}

i <- 1
accuracy <- rep(0, length(c))
for (m in c) {
  accuracy[i] <- calc.acc(y, 1 * (p_hat >= m))
  i = i + 1
}

plot(c, accuracy, type = "l", xlab = "c", ylab = "accuracy", main = "accuracy vs. c")
```



```r
mcval <- c[which.max(accuracy)]
mcval
```

```
## [1] 0.509018
```

The value of c that appears to maximize the accuracy of the classification is 0.509018.

QUESTION 2: The Six-Hump Camel Function

(a)

$$\begin{aligned}\boldsymbol{g} &= \nabla\rho(\boldsymbol{\theta}) \\ &= \begin{bmatrix}\frac{\partial\rho(\boldsymbol{\theta})}{\partial\theta_1} \\ \frac{\partial\rho(\boldsymbol{\theta})}{\partial\theta_2}\end{bmatrix} \\ &= \begin{bmatrix}8\theta_1 - 8.4\theta_1^3 + 2\theta_1^5 + \theta_2 \\ \theta_1 - 8\theta_2 + 16\theta_2^3\end{bmatrix}\end{aligned}$$

(b)

```r
rho <- function(theta) {
  theta1 <- theta[1]
  theta2 <- theta[2]
  rho_theta <- 4*theta1^2 - 2.1*theta1^4 + (theta1^6)/3 + theta1 *theta2 - 4 *theta2^2 +4 *theta2^4
  return(rho_theta)
}


gradient <- function(theta) {
  theta1 <- theta[1]
  theta2 <- theta[2]
  g1 <- 8*theta1- 8.4*theta1^3 + 2*theta1^5 + theta2
  g2 <- theta1 - 8*theta2 + 16*theta2^3
  g <- c(g1, g2)
  return(g)
}
```

(c)

```r
theta_0_list <- list(c(-1.5, 0), c(-1, 0), c(1, 0), c(1.5, 0))
results <- lapply(theta_0_list, function(init) {
  gradientDescent(theta = init, rhoFn = rho, gradientFn = gradient,
                  lineSearchFn = gridLineSearch, testConvergenceFn = testConvergence)
})
results
```

```
## [[1]]
## [[1]]$theta
## [1] -1.7002036  0.7956117
##
## [[1]]$converged
## [1] TRUE
##
## [[1]]$iteration
## [1] 5
##
## [[1]]$fnValue
## [1] -0.2153531
##
##
## [[2]]
## [[2]]$theta
## [1] -0.09226694  0.71103469
##
## [[2]]$converged
## [1] TRUE
##
## [[2]]$iteration
## [1] 5
##
## [[2]]$fnValue
## [1] -1.03158
##
##
## [[3]]
## [[3]]$theta
## [1]  0.09226694 -0.71103469
##
## [[3]]$converged
## [1] TRUE
##
## [[3]]$iteration
## [1] 5
##
## [[3]]$fnValue
## [1] -1.03158
##
##
## [[4]]
## [[4]]$theta
## [1]  1.7002036 -0.7956117
##
## [[4]]$converged
## [1] TRUE
##
## [[4]]$iteration
## [1] 5
##
## [[4]]$fnValue
## [1] -0.2153531
```

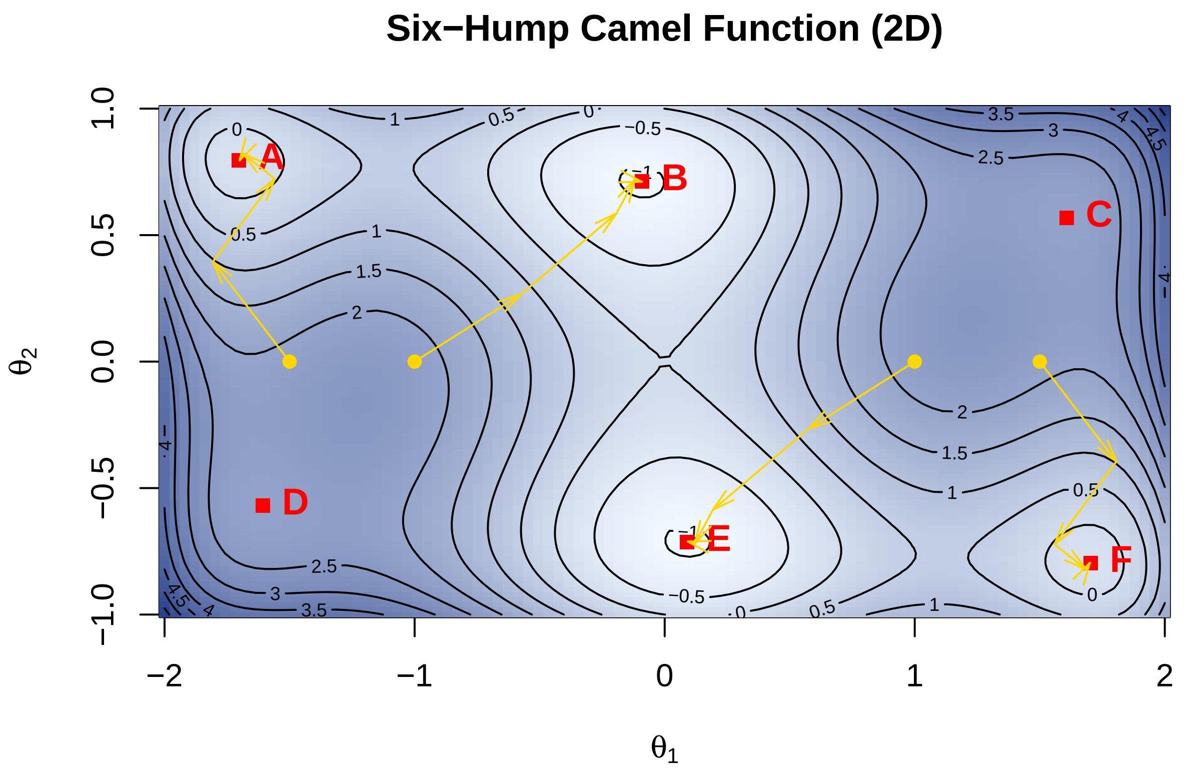
i. It converged to A and the value of the objective function is -0.2153531.
ii. It converged to B and the value of the objective function is -1.03158.
iii. It converged to E and the value of the objective function is -1.03158.
iv. It converged to F and the value of the objective function is -0.2153531.

(d)

```r
gradientDescentWithSolutionPath <- function(theta,
      rhoFn, gradientFn, lineSearchFn, testConvergenceFn,
      maxIterations = 100,
      tolerance = 1E-6, relative = FALSE,
      lambdaStepsize = 0.01, lambdaMax = 0.5) {

  SolutionPath = matrix(NA,nrow = maxIterations,ncol = length(theta))
  SolutionPath[1,] = theta
  converged <- FALSE
  i <- 0

  while (!converged & i <= maxIterations) {
    g <- gradientFn(theta) ## gradient
    glength <-  sqrt(sum(g^2)) ## gradient direction
    if (glength > 0) g <- g /glength

    lambda <- lineSearchFn(theta, rhoFn, g,
                lambdaStepsize = lambdaStepsize, lambdaMax = lambdaMax)

    thetaNew <- theta - lambda * g
    converged <- testConvergenceFn(thetaNew, theta,
                                   tolerance = tolerance,
                                   relative = relative)
    theta <- thetaNew
    i <- i + 1
    SolutionPath[(i+1),] = theta
  }
  SolutionPath = SolutionPath[1:(i+1),]
  ## Return last value and whether converged or not
  list(theta = theta, converged = converged, iteration = i, fnValue = rhoFn(theta) ,
       SolutionPath = SolutionPath
       )
}

Optim1 <- gradientDescentWithSolutionPath(theta = c(-1.5, 0), rhoFn = rho,
                                          gradientFn = gradient,
                                          lineSearchFn = gridLineSearch,
                                          testConvergenceFn = testConvergence)
Optim2 <- gradientDescentWithSolutionPath(theta = c(-1, 0), rhoFn = rho,
                                          gradientFn = gradient,
                                          lineSearchFn = gridLineSearch,
                                          testConvergenceFn = testConvergence)
Optim3 <- gradientDescentWithSolutionPath(theta = c(1, 0), rhoFn = rho,
                                          gradientFn = gradient,
                                          lineSearchFn = gridLineSearch,
                                          testConvergenceFn = testConvergence)
Optim4 <- gradientDescentWithSolutionPath(theta = c(1.5, 0), rhoFn = rho,
                                          gradientFn = gradient,
                                          lineSearchFn = gridLineSearch,
                                          testConvergenceFn = testConvergence)
```

```r
rhoplot <- function(theta1, theta2) {
  rho_theta <- 4*theta1^2 - 2.1*theta1^4 + (theta1^6)/3 +
    theta1 *theta2 - 4 *theta2^2 +4 *theta2^4
  return(rho_theta)
}

theta2 <- seq(-1, 1, length.out = 100)
theta1 <- seq(-2, 2, length.out = 100)

Rho <- outer(theta1,theta2,"rhoplot")
color_palette <- colorRampPalette(c("aliceblue", "royalblue4"))
image(theta1, theta2, Rho, col = color_palette(100),
      xlab = bquote(theta[1]),
      ylab = bquote(theta[2]), main = "Six-Hump Camel Function (2D)")
contour(theta1,theta2,Rho, add=T, levels = seq(-1, 5.5, by = 0.5))

minima_points <- list(
  c(-1.703, 0.796),  # A
  c(-0.09, 0.713),   # B
  c(1.607, 0.569),   # C
  c(-1.607, -0.569), # D
  c(0.09, -0.713),   # E
  c(1.703, -0.796)   # F
)
points(sapply(theta_0_list, `[[`, 1),
       sapply(theta_0_list, `[[`, 2), pch = 16, col = "gold")
minima_labels <- c("A", "B", "C", "D", "E", "F")
points(sapply(minima_points, `[[`, 1),
       sapply(minima_points, `[[`, 2), pch = 15, col = "red")
text(sapply(minima_points, `[[`, 1), sapply(minima_points, `[[`, 2),
     labels = minima_labels, pos = 4, col = "red", cex = 1.2, font = 2)

n.arrows = dim(Optim1$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim1$SolutionPath[i,1],Optim1$SolutionPath[i,2],
       Optim1$SolutionPath[(i+1),1],Optim1$SolutionPath[(i+1),2],
       length = 0.12,angle = 15, col = "gold")
}

n.arrows = dim(Optim2$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim2$SolutionPath[i,1],Optim2$SolutionPath[i,2],
       Optim2$SolutionPath[(i+1),1],Optim2$SolutionPath[(i+1),2],
       length = 0.12,angle = 15, col = "gold")
}

n.arrows = dim(Optim3$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim3$SolutionPath[i,1],Optim3$SolutionPath[i,2],
       Optim3$SolutionPath[(i+1),1],Optim3$SolutionPath[(i+1),2],
       length = 0.12,angle = 15, col = "gold")
}
```

```
n.arrows = dim(Optim4$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim4$SolutionPath[i,1],Optim4$SolutionPath[i,2],
       Optim4$SolutionPath[(i+1),1],Optim4$SolutionPath[(i+1),2],
       length = 0.12,angle = 15, col = "gold")
}
```



(e)

Different starting locations will result in different local minima. As observed in part (c) and part (d), we find these four starting values result in four different local minimum points. Thus, if we hope to locate the global optimum (i.e. global minimum in our question), it is important for us to select appropriate starting points. As shown in part (c) and part(d), if the starting points are far away from the global optimum that we hope to reach, then we could not find the global optimum points and the method would not work well as expected.

(f)

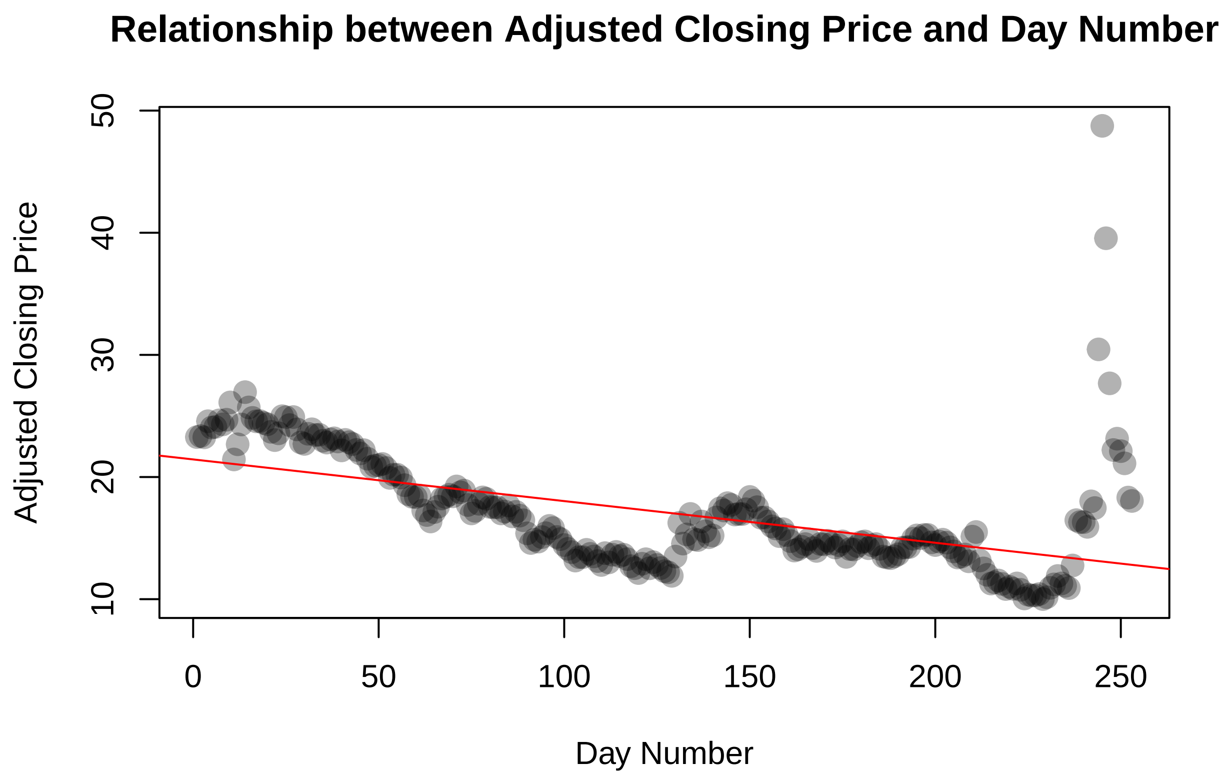
```
gradientDescent(theta = c(0, 0), rhoFn = rho, gradientFn = gradient,
                lineSearchFn = gridLineSearch, testConvergenceFn = testConvergence)
```

This results in an error is because the gradient direction of $\widehat{\boldsymbol{\theta}}_0 = (0,0)^T$ is 0 and so glength $= 0$ when we call gradientDescent(theta $=$ c(0, 0), rhoFn $=$ rho, gradientFn $=$ gradient, lineSearchFn $=$ gridLineSearch, testConvergenceFn = testConvergence). Since glength = 0, it is not greater than 0 and so d cannot be defined as g/glength.

QUESTION 3: GameStop Stock Volatility

(a)

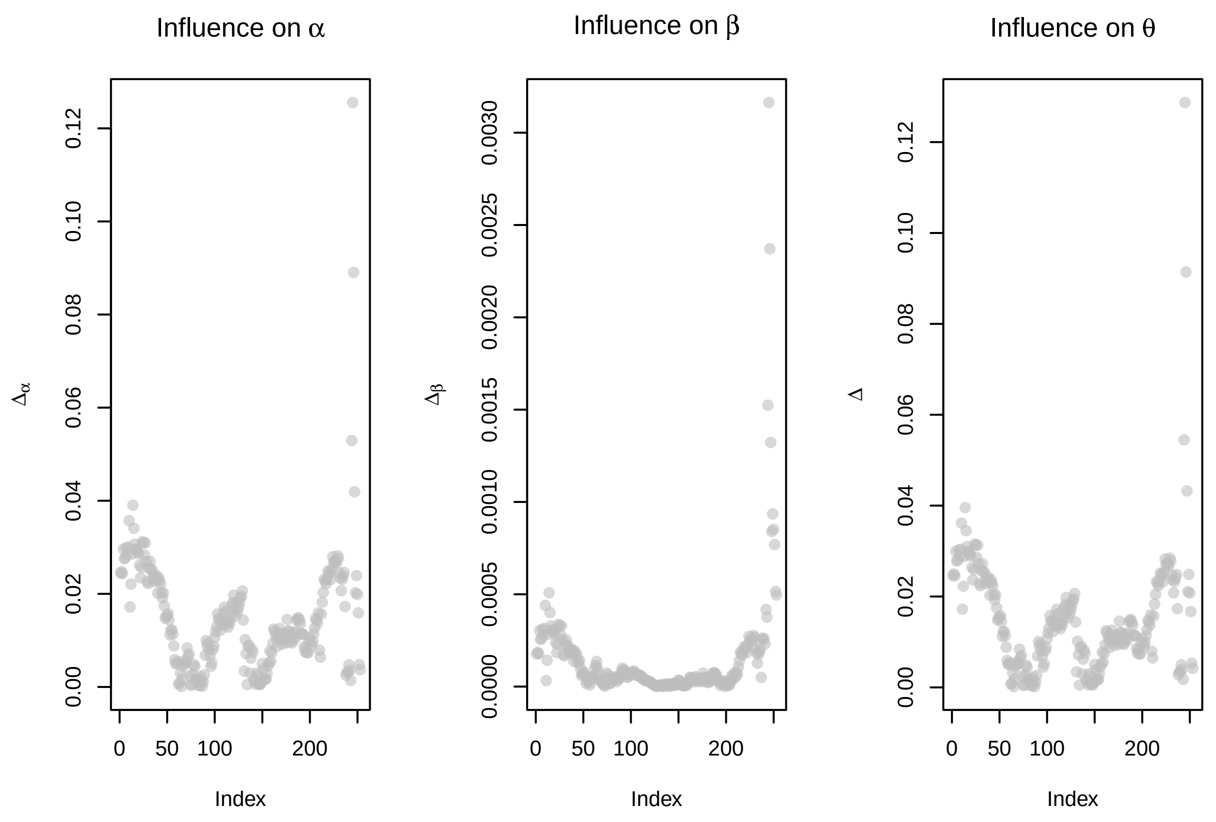
```
gme <- read.csv("GME.csv")
plot(x=gme$Day_Num, y=gme$Adj_Close, main =
       "Relationship between Adjusted Closing Price and Day Number",
     pch = 19, cex = 1.5, col = adjustcolor("black", alpha = 0.3),
     xlab = "Day Number", ylab = "Adjusted Closing Price")
x_bar <- mean(gme$Day_Num)
x_u <- gme$Day_Num
xc <- x_u - x_bar
y_u <- gme$Adj_Close
model1 <- lm(y_u ~ xc)
abline(a = model1$coef[1] - model1$coef[2] * x_bar, b = model1$coef[2], col = "red")
```



(b)

```
theta.hat <- model1$coef

N = nrow(gme)
delta = matrix(0, nrow = N, ncol = 2)
```

```r
for (i in 1:N) {
    temp.model = lm(Adj_Close ~ I(Day_Num-mean(Day_Num)), data = gme[-i, ])
    delta[i, ] = abs(theta.hat - temp.model$coef)
}

par(mfrow = c(1, 3))
plot(delta[, 1], ylab = bquote(Delta[alpha]), main = bquote("Influence on" ~
    alpha), pch = 19, col = adjustcolor("grey", 0.6))


plot(delta[, 2], ylab = bquote(Delta[beta]), main = bquote("Influence on" ~
    beta), pch = 19, col = adjustcolor("grey", 0.6))


delta2 = apply(X = delta, MARGIN = 1, FUN = function(z) {
    sum(z)
})
plot(delta2, ylab = bquote(Delta), main = bquote("Influence on" ~ theta), pch = 19,
    col = adjustcolor("grey", 0.6))
```



```r
gme[delta2>0.04,]
```

```
##            Date Day_Num  Open  High   Low Close Adj_Close    Volume
## 244 2024-05-13     244 26.34 38.20 24.77 30.45     30.45 187241700
## 245 2024-05-14     245 64.83 64.83 36.00 48.75     48.75 206979100
## 246 2024-05-15     246 40.31 42.35 31.00 39.55     39.55 131790100
## 247 2024-05-16     247 33.98 35.24 27.59 27.67     27.67  76177600
```

The four days that had the largest influence on this regression are Day 244, 245, 246, 247.

(c)

i.

$$
\begin{aligned}
\boldsymbol{g} &= \nabla \rho(\boldsymbol{\theta}; \mathcal{P}) \\
&= \begin{bmatrix} \frac{\partial \rho(\boldsymbol{\theta};\mathcal{P})}{\partial \alpha} \\ \frac{\partial \rho(\boldsymbol{\theta};\mathcal{P})}{\partial \beta} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{u\in\mathcal{P}} \frac{\partial \rho(r_u)}{\partial \alpha} \\ \sum_{u\in\mathcal{P}} \frac{\partial \rho(r_u)}{\partial \beta} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{u\in\mathcal{P}} \frac{\partial \rho(r_u)}{\partial r_u}\frac{\partial r_u}{\partial \alpha} \\ \sum_{u\in\mathcal{P}} \frac{\partial \rho(r_u)}{\partial r_u}\frac{\partial r_u}{\partial \beta} \end{bmatrix}
\end{aligned}
$$

Since $r_u = y_u - \alpha - \beta(x_u - \bar{x})$, we have $\frac{\partial r_u}{\partial \alpha} = -1$ and $\frac{\partial r_u}{\partial \beta} = -(x_u - \bar{x})$. Moreover, since $\rho(r) = \frac{r^2/2}{1+r^2}$, we have $\frac{\partial \rho(r)}{\partial r} = \frac{r}{(1+r^2)^2}$.

Thus, we have

$$
\begin{aligned}
\boldsymbol{g} &= \nabla \rho(\boldsymbol{\theta}; \mathcal{P}) \\
&= \begin{bmatrix} -\sum_{u\in\mathcal{P}} \frac{r_u}{(1+r_u^2)^2} \\ -\sum_{u\in\mathcal{P}} \frac{r_u(x_u-\bar{x})}{(1+r_u^2)^2} \end{bmatrix}
\end{aligned}
$$

ii.

```r
gm.fn <- function(r) {
  (r^2/2) / (1+r^2)
}

gm.fn.prime <- function(r) {
  r / ((1+r^2)^2)
}
createRobustGMRho <- function(x, y) {
    ## local variable
    xbar <- mean(x)
    ## Return this function
    function(theta) {
        alpha <- theta[1]
        beta <- theta[2]
        sum(gm.fn(y - alpha - beta * (x - xbar)))
    }
}
createRobustGMGradient <- function(x, y) {
    ## local variables
    xbar <- mean(x)
    ybar <- mean(y)
    function(theta) {
        alpha <- theta[1]
        beta <- theta[2]
        ru = y - alpha - beta * (x - xbar)
        rhop = gm.fn.prime(ru)
        -1 * c(sum(rhop * 1), sum(rhop * (x - xbar)))
    }
}
```

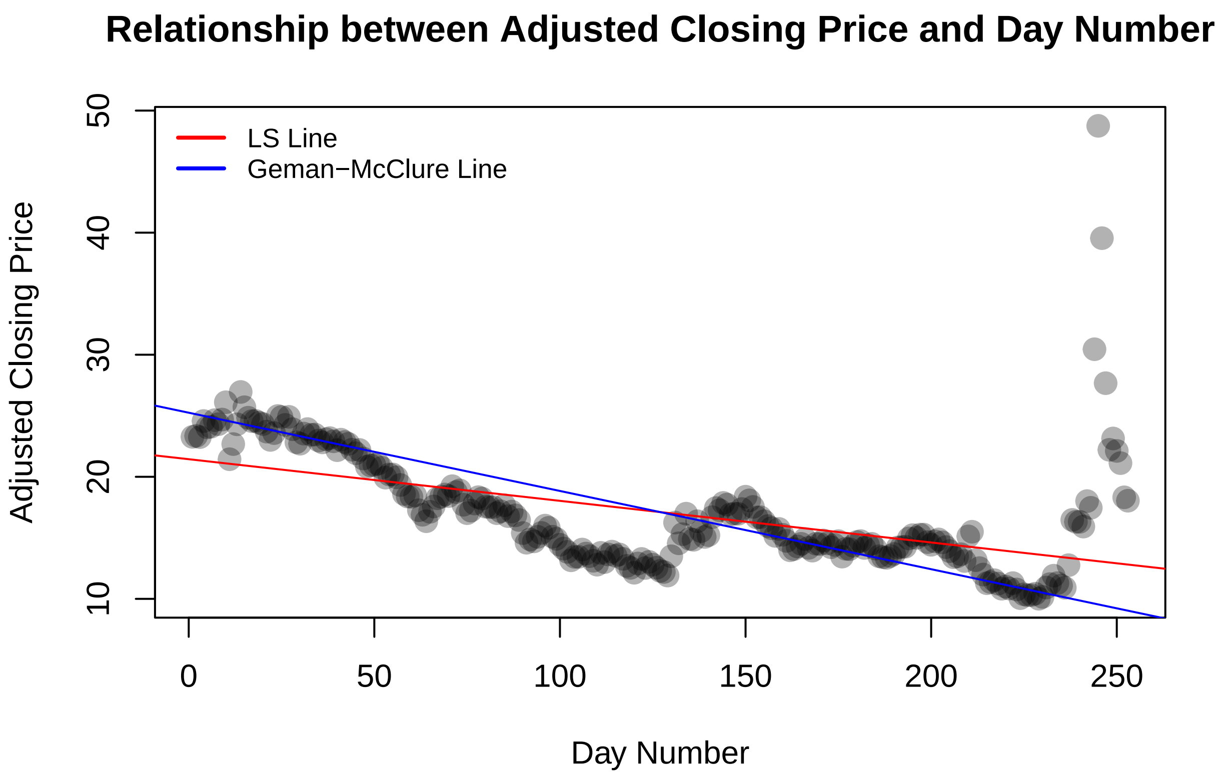
iii.

```r
rho <- createRobustGMRho(x = gme$Day_Num, y = gme$Adj_Close)

gradient <- createRobustGMGradient(x = gme$Day_Num, y = gme$Adj_Close)

result <- gradientDescent(theta = model1$coefficients, rhoFn = rho, gradientFn = gradient,
    lineSearchFn = gridLineSearch, testConvergenceFn = testConvergence)

print(result)
```

```
## $theta
## (Intercept)          xc
##  17.1094316  -0.0641078
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 2
##
## $fnValue
## [1] 73.47534
```

iv.

```r
gme <- read.csv("GME.csv")
plot(x=gme$Day_Num, y=gme$Adj_Close, main =
       "Relationship between Adjusted Closing Price and Day Number",
     pch = 19, cex = 1.5, col = adjustcolor("black", alpha = 0.3),
     xlab = "Day Number", ylab = "Adjusted Closing Price")
x_bar <- mean(gme$Day_Num)
x_u <- gme$Day_Num
xc <- x_u - x_bar
y_u <- gme$Adj_Close
model1 <- lm(y_u ~ xc)
abline(a = model1$coef[1] - model1$coef[2] * x_bar, b = model1$coef[2], col = "red")
abline(a = result$theta[1] - result$theta[2] * x_bar, b = result$theta[2], col = "blue")

legend("topleft", legend = c("LS Line", "Geman-McClure Line"), col = c("red", "blue"),
       bty = "n", lty = 1, lwd=2, cex=0.8)
```



The least sqaures regression line is more sensitive to the outliers. On the contrary, the Geman-McClure regression line calculated in part iii is less sensitive to extreme values and more resistant to outliers.