Bios 301: Assignment 3

Due Thursday, 21 November, 12:00 PM

50 points total.

Submit a single knitr (either `.rnw` or `.rmd`) file, along with a valid PDF output file. Inside the file, clearly indicate which parts of your responses go with which problems (you may use the original homework document as a template). Raw R code/output or word processor files are not acceptable.

Question 1

20 points

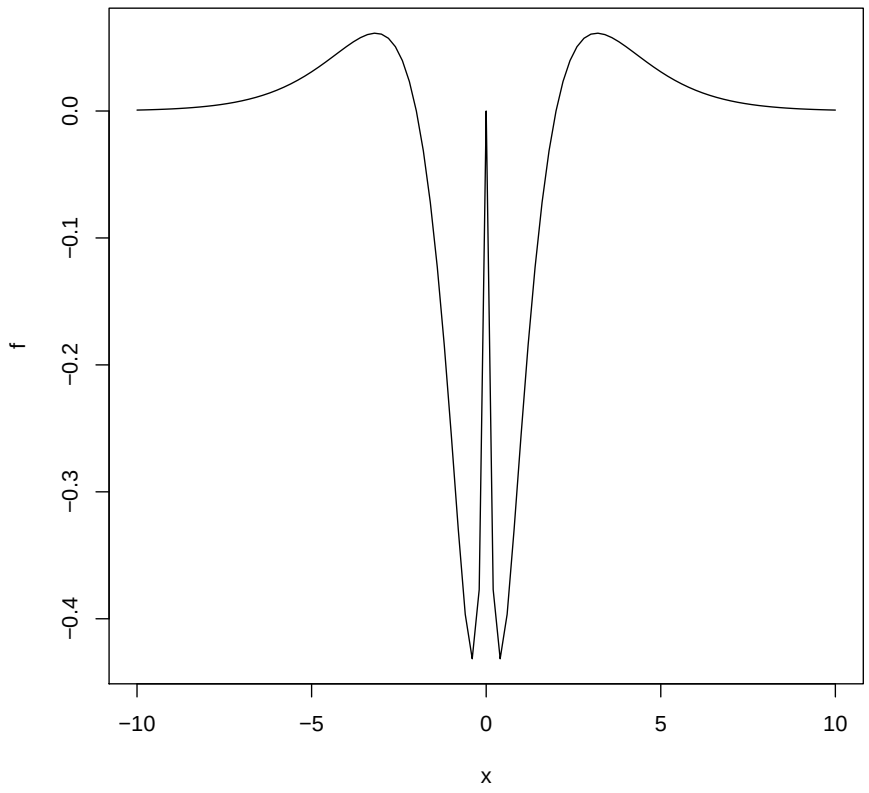
Code a function that does golden section search, and use this function to find all of the local maxima on the following function:

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ |x| \log\left(\frac{|x|}{2}\right) e^{-|x|} & \text{otherwise} \end{cases}$$

on the interval [-10, 10].

To get an idea of what the function looks like, it might be helpful to plot it.

```
f <- function(x) {
    ifelse(x == 0, 0, abs(x) * log(abs(x)/2) * exp(-abs(x)))
}
plot(f, xlim = c(-10, 10))
```




```r
golden_section <- function(f, xa, xb, xc, tol = 1e-09) {
    stopifnot(xc < xb)
    stopifnot(xc > xa)
    # Calculate golden ratio (plus 1)
    ratio <- 1 + (1 + sqrt(5))/2
    # Initial values of function at xa, xb, xc
    fa <- f(xa)
    fb <- f(xb)
    fc <- f(xc)
    # Loop until stopping rule
    while (abs(xb - xa) > tol) {
        # xb to xc larger than xc to xa
        if ((xb - xc) > (xc - xa)) {
            # Calculate new value of y
            y <- xc + (xb - xc)/ratio
            fy <- f(y)
```

```
            if (fy >= fc) {
                # Assign xc to xa
                xa <- xc
                fa <- fc
                # Assign y to xc
                xc <- y
                fc <- fy
            } else {
                # Assign y to xb, xa and xc stay the same
                xb <- y
                fb <- fy
            }
            # xc to xa larger than xb to xc
        } else {
            # Calculate new value of y
            y <- xc - (xc - xa)/ratio
            fy <- f(y)
            if (fy >= fc) {
                # Assign xc to xb
                xb <- xc
                fb <- fc
                # Assign y to xc
                xc <- y
                fc <- fy
            } else {
                # Assign y to xa, others stay the same
                xa <- y
                fa <- fy
            }
        }
    }
    return(xc)
}
# Left maximum:
golden_section(f, -10, 0, -5)

## [1] -3.17

f(-3.170464)

## [1] 0.06133

# Middle maximum:
golden_section(f, -1, 1, 0)
```

```
## [1] 0

f(0)

## [1] 0

# Right maximum:
golden_section(f, 1, 10, 5)

## [1] 3.17

f(3.170464)

## [1] 0.06133
```

Question 2

10 points

Obtain the code for using Newton's Method to estimate logistic regression parameters (`logtistic.r`) and modify it to predict `death` from `weight`, `hemoglobin` and `cd4baseline` in the HAART dataset. Use complete cases only. Report the estimates for each parameter, including the intercept.

```
# Logistic.r:
data2 <- read.table("~/Bios301/datasets/haart.csv", sep = ",", head = T)
# Logistic function
logistic <- function(x) 1/(1 + exp(-x))
# Removing all NA's from data frame:
new.data <- data2[c(6, 7, 4, 11)]
new.data2 <- new.data[complete.cases(new.data), ]
# x contains weight, hemoglobin and cd4 from data2
x <- new.data2[1:3]
# y contains death indicator
y <- new.data2[4]
estimate_logistic <- function(x, y, MAX_ITER = 10) {
    # Assigning n number of rows in x
    n <- dim(x)[1]
    # Assigning k number of columns in x
    k <- dim(x)[2]
    x <- as.matrix(cbind(rep(1, n), x))
    y <- as.matrix(y)
```

```
    # Initialize fitting parameters
    theta <- rep(0, k + 1)
    J <- rep(0, MAX_ITER)
    for (i in 1:MAX_ITER) {
        # Calculate linear predictor
        z <- x %*% theta
        # Apply logit function
        h <- logistic(z)
        # Calculate gradient
        grad <- t((1/n) * x) %*% as.matrix(h - y)
        # Calculate Hessian
        H <- t((1/n) * x) %*% diag(array(h)) %*% diag(array(1 - h)) %*% x
        # Calculate log likelihood
        J[i] <- (1/n) %*% sum(-y * log(h) - (1 - y) * log(1 - h))
        # Newton's method
        theta <- theta - solve(H) %*% grad
    }
    return(theta)
}
estimate_logistic(x, y)
```

```
##                   [,1]
## rep(1, n)    3.576412
## weight      -0.046211
## hemoglobin  -0.350643
## cd4baseline  0.002093
```

```
# Compare with R's built-in linear regression
g <- glm(data2$death ~ data2$weight + data2$hemoglobin + data2$cd4baseline,
    data = data2, family = binomial(logit))
print(g$coefficients)
```

```
##       (Intercept)      data2$weight  data2$hemoglobin data2$cd4baseline
##          3.576412         -0.046211         -0.350643          0.002093
```

Question 3

20 points

Consider the following very simple genetic model (*very* simple – don't worry if you're not a geneticist!). A population consists of equal numbers of two sexes: male and female. At each generation men and women are paired at random, and each pair produces exactly two offspring, one male and one female. We are interested in the distribution of height from one generation to the next. Suppose

that the height of both children is just the average of the height of their parents, how will the distribution of height change across generations?

Represent the heights of the current generation as a dataframe with two variables, m and f, for the two sexes. The command rnorm(100, 160, 20) will generate a vector of length 100, according to the normal distribution with mean 160 and standard deviation 20 (see Section 16.5.1). We use it to randomly generate the population at generation 1:

```
pop <- data.frame(m = rnorm(100, 160, 20), f = rnorm(100, 160, 20))
```

The command `sample(x, size = length(x))` will return a random sample of size `size` taken from the vector `x`. The following function takes the data frame `pop` and randomly permutes the ordering of the men. Men and women are then paired according to rows, and heights for the next generation are calculated by taking the mean of each row. The function returns a data frame with the same structure, giving the heights of the next generation.

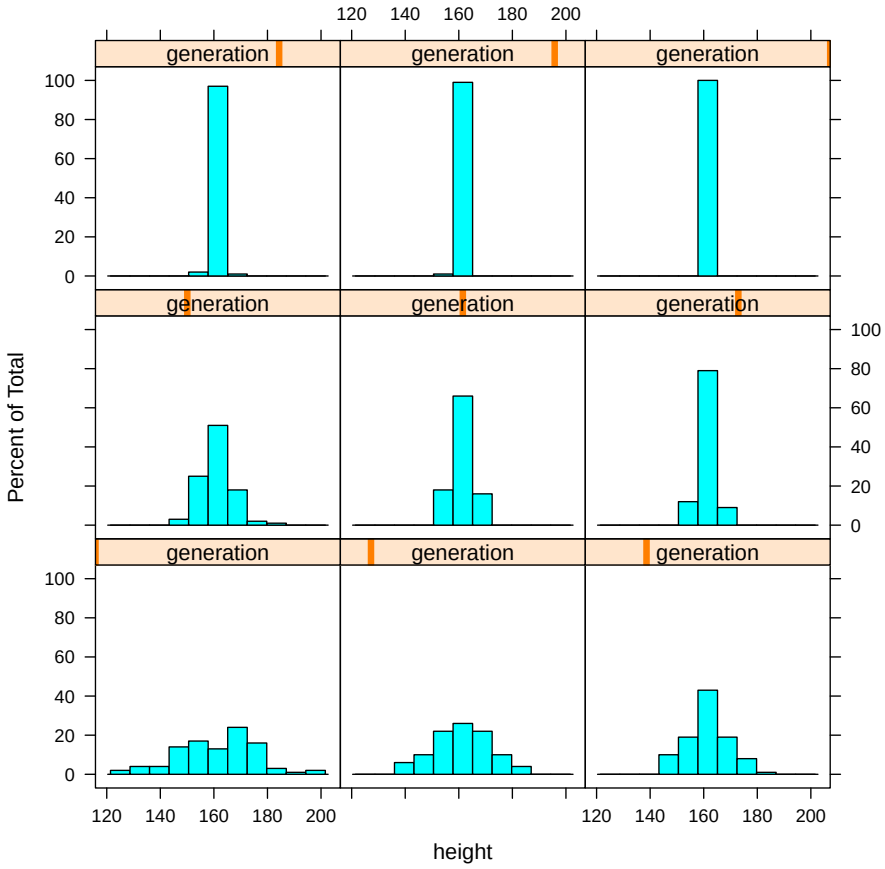
```
next_gen <- function(pop) {
    pop$m <- sample(pop$m)
    pop$m <- apply(pop, 1, mean)
    pop$f <- pop$m
    return(pop)

}
```

Use the function `next_gen` to generate nine generations, then use the function `histogram` from the `lattice` to plot the distribution of male heights in each generation. The phenomenon you see is called regression to the mean.

Hint: construct a data frame with variables height and generation, where each row represents a single man.

```
# creates generation 1: 100 males, 100 females both normal dist. with mean
# 160, and standard deviation 20
pop <- data.frame(m = rnorm(100, 160, 20), f = rnorm(100, 160, 20))
next_gen <- function(pop) {
    pop$m <- sample(pop$m)
    pop$m <- apply(pop, 1, mean)
    pop$f <- pop$m
    return(pop)
}
# 900 rows, column for height and column for generation
D <- data.frame(generation = rep(1:9, each = 100), height = c(0))
# Creating generations 1-9
pop <- next_gen(pop)
```

```
pop2 <- next_gen(pop)
pop3 <- next_gen(pop2)
pop4 <- next_gen(pop3)
pop5 <- next_gen(pop4)
pop6 <- next_gen(pop5)
pop7 <- next_gen(pop6)
pop8 <- next_gen(pop7)
pop9 <- next_gen(pop8)
D[1:100, 2] <- pop[, 1]
D[101:200, 2] <- pop2[, 1]
D[201:300, 2] <- pop3[, 1]
D[301:400, 2] <- pop4[, 1]
D[401:500, 2] <- pop5[, 1]
D[501:600, 2] <- pop6[, 1]
D[601:700, 2] <- pop7[, 1]
D[701:800, 2] <- pop8[, 1]
D[801:900, 2] <- pop9[, 1]
library(lattice)
histogram(~height | generation, data = D, layout = c(3, 3))
```


generation
Percent of Total
height
0
20
40
60
80
100
120
140
160
180
200