HW2_STAT600

Katy Miles

2024-02-13

Problem 1

a)

$$l(\theta) = -\sum_{i=1}^{n} log(\pi(1 + (x_i - \theta)^2))$$

$$l'(\theta) = 2\sum_{i=1}^{n} \frac{1}{(1 + (x_i - \theta)^2)}(x_i - \theta)$$

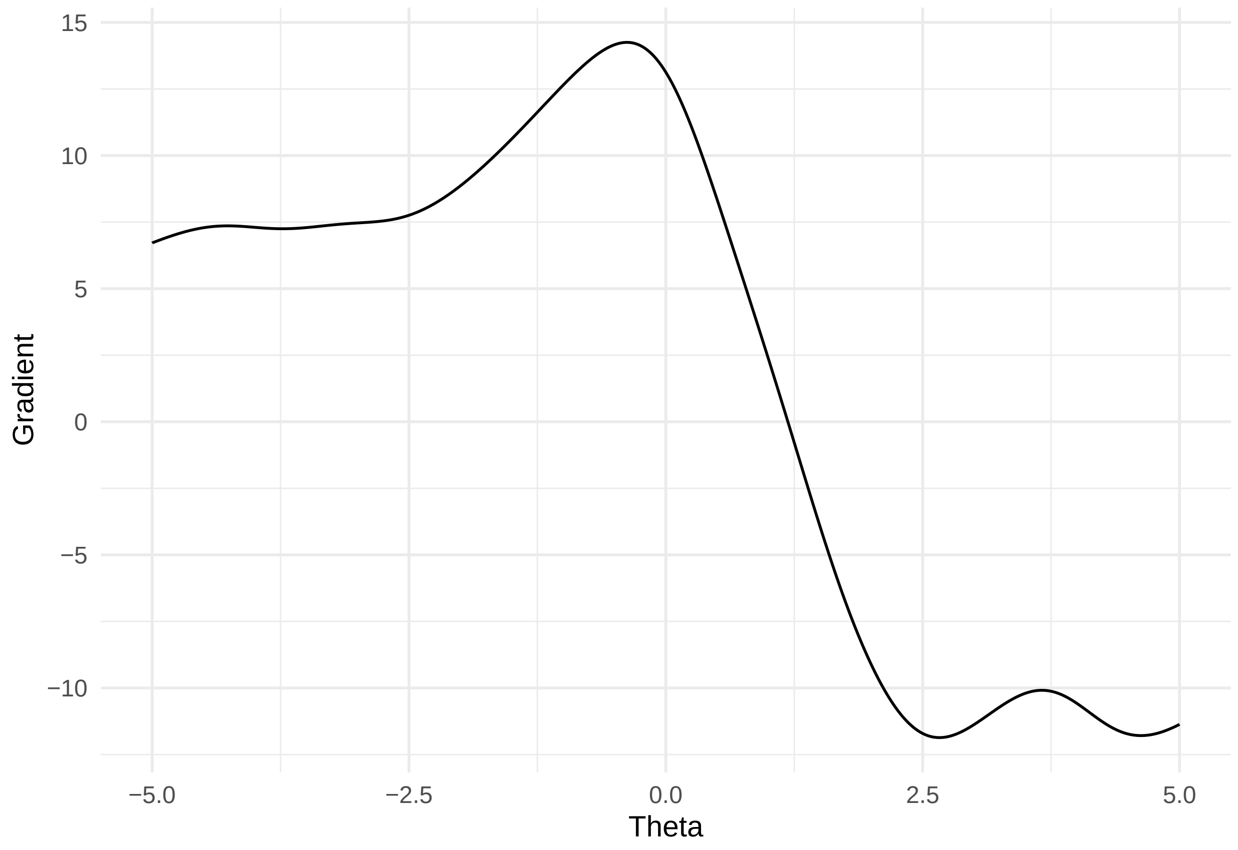
```r
library(tidyverse)
library(Rcpp)

data = c(-8.86, -6.82, -4.03, -2.84, 0.14, 0.19, 0.24, 0.27, 0.49, 0.62, 0.76, 1.09,
1.18, 1.32, 1.36, 1.58, 1.58, 1.78, 2.13, 2.15, 2.36, 4.05, 4.11, 4.12,
6.83)

loglike_grad = function(theta, data) {
  return(2*sum((data - theta) / (( 1 + (data - theta)^2))))
}

theta = seq(-5, 5, 0.001)
output = sapply(theta, loglike_grad, data = data)

ggplot() +
  geom_line(aes(theta, output)) +
  xlab("Theta") +
  ylab("Gradient")+
  theme_minimal()
```



b)

Derivation of $l''(\theta)$ for Newton-Raphson:

$$l'(\theta) = 2\sum_{i=1}^{n} \frac{1}{(1+(x_i-\theta)^2)}(x_i-\theta)$$

$$l''(\theta) = -2\sum_{i=1}^{n} \frac{1}{(1+(x_i-\theta)^2)} + 4\sum_{i=1}^{n} \frac{(x_i-\theta)^2}{((1+(x_i-\theta)^2))^2}$$

```
#include <Rcpp.h>
using namespace Rcpp;

// helper function
namespace helper {
  double loglike_grad(double theta, NumericVector data) {
    return 2*sum((data - theta) / ((1 + pow(data - theta, 2))));
  }

  double loglike_grad2(double theta, NumericVector data) {
    double first = 4*sum(pow(data - theta, 2)
                     / pow((1 + pow(data - theta, 2)), 2));
    double second = -2*sum(1 / ((1 + pow(data - theta, 2))));
    return first + second;
  }
}

// Bisection method
//[[Rcpp::export]]
List bisection (NumericVector data, double a, double b, double epsilon) {
  for (int i = 0; i < 1000; i++) {
    double x = (a + b) / 2;
    // Updating equations
    double grad_eval = helper::loglike_grad(x, data)*helper::loglike_grad(a, data);
    if (grad_eval <= 0) {
      b = x;
    } else {
      a = x;
    }
    // Stoping criteria
    if (abs((((a + b) / 2) - x)) < epsilon) {
      return List::create((a + b) / 2, i);
    }
  }
  return List::create((a + b) / 2, 1000);
}

// Newton-Raphson
//[[Rcpp::export]]
List newtonRaphson (NumericVector data, double x, double epsilon) {
  for (int i = 0; i < 1000; i++) {
    double h = -helper::loglike_grad(x, data)/helper::loglike_grad2(x, data);
    // Stopping criteria
    if (abs(((x + h) - x)) < epsilon) {
      return List::create(x + h, i);
    }
    // Updating equation
    x = x + h;
  }
  return List::create(x, 1000);
}

// Fisher Scoring
//[[Rcpp::export]]
List fisher (NumericVector data, double x, double epsilon) {
  for (int i = 0; i < 1000; i++) {
    // Updating equations
    double h = 2*helper::loglike_grad(x, data)/data.size();
    // Stopping criteria
    if (abs(((x + h) - x)) < epsilon) {
      return List::create(x + h, i);
    }
    // Updating equation
    x = x + h;
  }
  return List::create(x, 1000);
}
```

```
// Secant Method
//[[Rcpp::export]]
List secant (NumericVector data, double x_0, double x_1, double epsilon) {
  for (int i = 0; i < 1000; i++) {
    // Updating equations
    double prev = x_1;
    x_1 = x_1 - helper::loglike_grad(x_1, data)*(x_1 - x_0) / (helper::loglike_grad(x_1, data) - helper
    x_0 = prev;
    // Stopping criteria
    if (abs(x_1 - x_0) < epsilon) {
      return List::create(x_1, i);
    }
  }
  return List::create(x_1, 1000);
}
```

```
bisection_output = bisection(data, -5, 5, .0001)
newtonRaphson_output = newtonRaphson(data, 1, .0001)
secant_output = secant(data, 0, 1, .0001)
fisher_output = fisher(data, 1, .0001)
```

c)

```
library(knitr)
library(kableExtra)
table = t(matrix(data = c("Bisection", unlist(bisection_output),
                 "Newton Raphson", unlist(newtonRaphson_output),
                 "Secant", unlist(secant_output),
                 "Fisher", unlist(fisher_output)), nrow = 3))

kable(table, "latex", digits = 5, booktabs = TRUE, col.names = c("Method", "Estimate", "Iterations")) %
    kable_styling(latex_options = "HOLD_position")
```

Method	Estimate	Iterations
Bisection	1.18797302246094	15
Newton Raphson	1.18794279110024	2
Secant	1.18794279110816	3
Fisher	1.18794275189527	2

d)

For my convergence criteria, I used the absolute convergence criterion for which $\|x^{(t+1)} - x^{(t)}| < \epsilon$. I chose $\epsilon = 0.0001$ for all methods.

e)

```
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
double loglike_grad2(double theta, NumericVector data) {
  double first = 4*sum(pow(data - theta, 2)
                       / pow((1 + pow(data - theta, 2)), 2));
  double second = -2*sum(1 / ((1 + pow(data - theta, 2))));
  return first + second;
}
```

```
est_hessian = loglike_grad2(newtonRaphson_output[[1]], data)
se = sqrt(-1/est_hessian)
```

The best estimate of θ is 1.188 and the standard error is 0.281.

f)

I chose to initialize the methods based off of plot in part a). A root looked to be around 1, so I used 1 as my initialization value. For the secant method, I chose to use 0.5 and 1 as my initial values, again based off of the visual of the gradient. The results may be sensitive to the initialization value if for instance for the bisection method, the initial bounds do not include the root. There could also be an issue of getting stuck in a local maximum depending on the initialization value.

g)

```
new_data = c(-8.34, -1.73, -0.40, -0.24, 0.60, 0.94, 1.05, 1.06, 1.45, 1.50,
1.54, 1.72, 1.74, 1.88, 2.04, 2.16, 2.39, 3.01, 3.01, 3.08,
4.66, 4.99, 6.01, 7.06, 25.45, data)

theta = seq(-10, 10, 0.001)
output = sapply(theta, loglike_grad, data = new_data)


bisection_output = bisection(new_data, -5, 5, .0001)
newtonRaphson_output = newtonRaphson(new_data, 1, .0001)
secant_output = secant(new_data, 0, 1, .0001)
fisher_output = fisher(new_data, 1, .0001)

table = t(matrix(data = c("Bisection", unlist(bisection_output),
                "Newton Raphson", unlist(newtonRaphson_output),
                "Secant", unlist(secant_output)), nrow = 3))

kable(table, "latex", digits = 5, booktabs = TRUE, col.names = c("Method", "Estimate", "Iterations")) %
    kable_styling(latex_options = "HOLD_position")
```

Method	Estimate	Iterations
Bisection	1.47132873535156	15
Newton Raphson	1.47129874670668	2
Secant	1.47129874662954	4

```r
est_hessian = loglike_grad2(newtonRaphson_output[[1]], data)
se = sqrt(-1/est_hessian)
```

With the updated data, the best estimate of θ is 1.47 with a standard error of 0.285.

Problem 2

We want to prove that the convergence rate for 2 steps of the secant method is better than the convergence rate for 1 step of the Newton Raphson method.

First, we will find the convergence rate for 1 step of the Newton Raphson method:

We note that a method has a convergence of order β if $lim_{t->\infty}\epsilon^{(t)} = 0$ and $lim_{t->\infty}\frac{|\epsilon^{(t+1)}|}{|\epsilon^{(t)}|^{\beta}} = c$

For Newton's algorithm, we have $\frac{|\epsilon^{(t+1)}|}{|\epsilon^{(t)}|^{2}} = \frac{g''(q)}{2g''(x^{(t)})}$ with the right hand converging to $\frac{g''(x^*)}{2g''(x^*)}$. Therefore, Newton's method has quadratic convergence ($\beta = 2$).

Now, we will find the convergence rate for 2 steps of the secant method:

Form derivations shown in the Givens and Hoeting, we find that,

$$\epsilon^{(t+1)} \approx d^{(t)}\epsilon^{(t)}\epsilon^{(t-1)}$$

Thus, $\epsilon^{(t+2)} \approx d^{(t)}d^{(t)}\epsilon^{(t)}\epsilon^{(t-1)}\epsilon^{(t)}$, where $d^{(t)} \to \frac{g''(x^*)}{2g''(x^*)} = d$ as $t \to \infty$.

Now, to find the order of convergence for two steps of the secant method, we find β for which

$$lim_{t->\infty}\frac{|\epsilon^{(t+2)}|}{|\epsilon^{(t)}|^{\beta}} = c$$

$$lim_{t->\infty}\frac{|\epsilon^{(t+2)}|}{|\epsilon^{(t)}|^{\beta}} = c$$

After rearranging terms and leaving $\epsilon^{(t+2)} \approx d^{(t)}d^{(t)}\epsilon^{(t)}\epsilon^{(t-1)}\epsilon^{(t)}$ only in terms of $\epsilon^{(t)}$, we obtain

$$lim_{t\to\infty}|\epsilon^{(t)}|^{-\beta+1/\beta+2} = \frac{c^{1+1/\beta}}{d^2}$$

Solving $2 - \beta + 1/\beta = 0$, we get $\beta = 2 + \frac{\sqrt{8}}{2} \approx 3.414214$.

Thus, we see that the convergence rate for 2 steps of the secant method is better than the convergence rate for 1 step of the Newton Raphson method.

Problem 3

a)

$$l(\beta_0, \beta_1, \beta_2|\boldsymbol{y}) = \sum_{i=1}^{n} log((\frac{exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1 + exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})})^{y_i}(1 - \frac{exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1 + exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})})^{1-y_i})$$

$$= \sum_{i=1}^{n}[y_i(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}) - y_i log(1 + exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})) + (1-y_i) log(1 - \frac{exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})}{1 + exp(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2})})]$$

$$= \sum_{i=1}^{n}[y_i(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}) - y_i log(1+exp(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2})) + (1-y_i) log(\frac{1}{1+exp(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2})})]$$

$$= \sum_{i=1}^{n}[y_i(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}) - y_i log(1+exp(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2})) + (y_i-1) log(1+exp(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}))]$$

$$= \sum_{i=1}^{n}[(y_i(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}) - log(1+exp(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}))]$$

$$= \sum_{i=1}^{n}[(y_i(\boldsymbol{x_i}\boldsymbol{\beta}) - log(1+exp(\boldsymbol{x_i}\boldsymbol{\beta}))]$$

b

For the Newton Raphson algorithm, we will need the Hessian and the gradient vector.

$$\frac{\partial}{\partial \boldsymbol{\beta}} l(\boldsymbol{\beta}|\boldsymbol{y}) = \sum_{i=1}^{n}[y_i - \frac{exp(\boldsymbol{x_i}\boldsymbol{\beta})}{1+exp(\boldsymbol{x_i}\boldsymbol{\beta})}]\boldsymbol{x_i}$$

$$\frac{\partial^2}{\partial \boldsymbol{\beta}^2} l(\boldsymbol{\beta}|\boldsymbol{y}) = \sum_{i=1}^{n}[\frac{exp(\boldsymbol{x_i}\boldsymbol{\beta})}{(1+exp(\boldsymbol{x_i}\boldsymbol{\beta}))^2}]\boldsymbol{x_i}\boldsymbol{x_i}^T$$

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
using namespace arma;

namespace helper{
  colvec loglike_grad(colvec theta, mat x, colvec y) {
        colvec grad = trans(x)*(y - (exp(x*theta) / (1 + exp(x*theta))));
        return grad;
  }


  mat hessian(colvec theta, mat x, colvec y) {
        mat grad = trans(x)*(-diagmat(exp(x*theta) / pow(1 + exp(x*theta), 2)))*x;
        return grad;
  }
}


// Newton-Raphson
//[[Rcpp::export]]
List newtonRaphson (mat data, colvec y, colvec x, double epsilon) {
  for (int i = 0; i < 1000; i++) {
    // Updating equations
    colvec grad = helper::loglike_grad(x, data, y);
    mat grad2 = helper::hessian(x, data, y);
    colvec h = inv(grad2)*grad;
    // Stopping criteria
    double D = sum(abs(((x - h) - x)));
    if (D < epsilon) {
      return List::create(x - h, i);
    }
    // Updating equation
    x = x - h;
  }
  return List::create(x, 1000);
}
```

```
# Create full data
coffee = c(rep(0, 41), rep(2, 213), rep(4, 127), rep(5, 142), rep(0, 67),
           rep(2, 211), rep(4, 133), rep(5, 76))

gender = c(rep(0, 41), rep(0, 213), rep(0, 127), rep(0, 142), rep(1, 67),
           rep(1, 211), rep(1, 133), rep(1, 76))

y = c(rep(1, 9), rep(0, 41-9), rep(1, 94), rep(0, 213-94), rep(1, 53),
rep(0, 127-53), rep(1, 60), rep(0, 142-60), rep(1, 11), rep(0, 67-11),
rep(1, 59), rep(0, 211-59), rep(1, 53), rep(0, 133-53), rep(1, 28), rep(0,76-28))

data = matrix(c(rep(1, 1010),coffee, gender), ncol = 3)

# Call Newton-Raphson
newton_output = newtonRaphson(data, y, c(0,0,0), .0000001)
table = matrix(data = c(unlist(newton_output)), ncol = 4)

kable(table, "latex", digits = 5, booktabs = TRUE,
      col.names = c("Beta_0 Estimate",
                    "Beta_1 Estimate",
                    "Beta_2 Estiamte",
                    "Iterations")) %>%
    kable_styling(latex_options = "HOLD_position")
```

Beta_0 Estimate	Beta_1 Estimate	Beta_2 Estiamte	Iterations
-0.79054	0.1383	-0.39725	4

c)

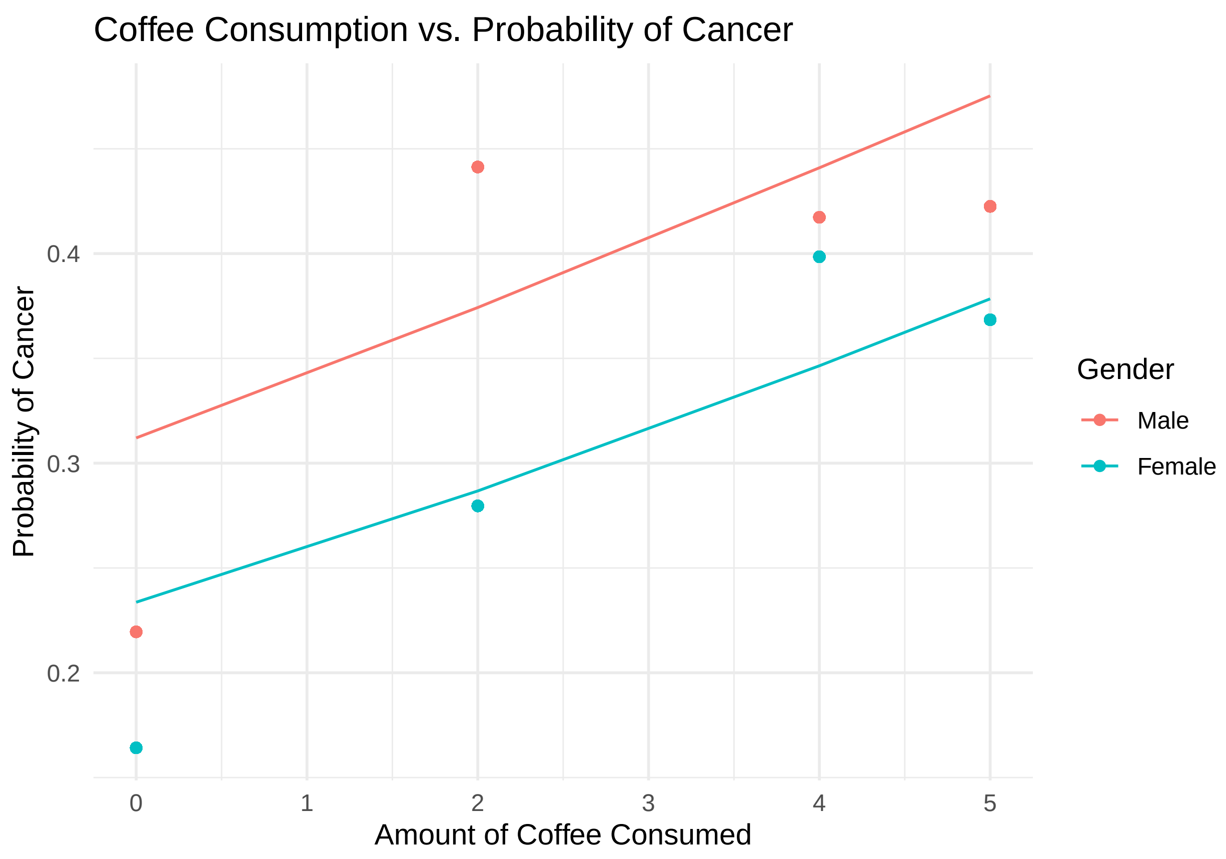
```
# Find probability of cancer for males and female
p = c(rep(9/41, 41), rep(94/213,213), rep(53/127,127), rep(60/142,142),
      rep(11/67, 67), rep(59/211,211), rep(53/133,133), rep(28/76, 76))

est_p = function(x1, x2) {
  exp_part = exp(newton_output[[1]][1,] +
             newton_output[[1]][2,]*x1 + newton_output[[1]][3,]*x2)
  return(exp_part / (1 + exp_part))
```

```r
}

# Get estimated probabilities
estimates = est_p(coffee, gender)

ggplot() +
  geom_point(aes(coffee, p, color = factor(gender))) +
  geom_line(aes(coffee, estimates, color = factor(gender))) +
  theme_minimal() +
  ggtitle("Coffee Consumption vs. Probability of Cancer") +
  xlab("Amount of Coffee Consumed") +
  ylab("Probability of Cancer") +
  guides(color=guide_legend(title="Gender")) +
  scale_color_discrete(labels = c("Male", "Female"))
```



d)

```cpp
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
using namespace arma;

//[[Rcpp::export]]
```

```
mat hessian(colvec theta, mat x, colvec y) {
      mat grad = trans(x)*(-diagmat(exp(x*theta) / pow(1 + exp(x*theta), 2)))*x;
      return grad;
}
```

```
# Compute hessian
est_hessian = hessian(newton_output[[1]], data, y)
se = diag(sqrt(diag(solve(-est_hessian))))

upper = newton_output[[1]] + 1.96*diag(se)
lower = newton_output[[1]] - 1.96*diag(se)
table = data.frame(matrix(data = c(lower, upper), ncol = 2))
rownames(table) = c("Beta_0", "Beta_1", "Beta_2")

kable(table, "latex", digits = 5, booktabs = TRUE,
      col.names = c("Lower Bound", "Upper Bound")) %>%
    kable_styling(latex_options = "HOLD_position")
```

	Lower Bound	Upper Bound
Beta_0	-1.11104	-0.47004
Beta_1	0.05455	0.22204
Beta_2	-0.65929	-0.13521

We see that all parameter estimates are significantly different from 0 with $\alpha = 0.05$, as none of the confidence intervals contain 0.