Lab 2 report

Simon Jorstedt & Siddhesh Sreedar

2023-11-14

Question 1

We consider the function

$$g(x,y) = -x^2 - x^2y^2 - 2xy + 2x + 2.$$

The gradient to g is

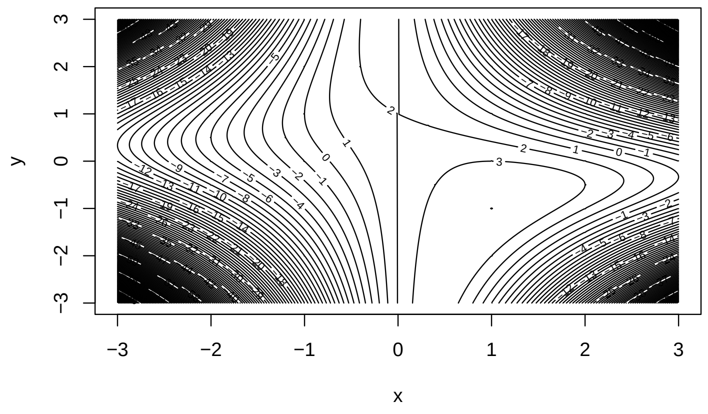
$$\nabla g(x,y) = (-2x - 2y^2x - 2y + 2, -2x^2y - 2x).$$

The Hessian matrix corresponding to g is

$$H_g = \begin{bmatrix} -2 - 2y^2 & -4xy - 2 \\ -4xy - 2 & -2x^2 \end{bmatrix}.$$

See the Appendix for the R implementation of these functions. We will now proceed to plot a contour plot of g in the region $x, y \in [-3, 3]$, with the precision of a square with side 0.01. See Figure 1. We are interested in finding local maximums of g. To achieve this, we will implement the Newton method in a code chunk which can be found in the Appendix.

Fig 1. Contour plot of g(x,y)



When the Newton method is run on the starting points $(2,0), (-1,-2), (0,1)$ and $(0.5,1)$, the following results are obtained. There is convergence to two separate points, $(1,-1)$ and $(0,1)$. The gradient is zero (vector) at both points. When analysing the respective Hessian matrices for these points, we find that the eigenvalues for the Hessian at $(1,-1)$ can be rounded to -0.76 and -5.24. Both eigenvalues being negative indicates that the point $(1,-1)$ is a local maximum. This is not true for $(0,1)$ where the Hessians eigenvalues can be rounded to 0.83 and -4.83, indicating that this point is neither a local maximum or minimum. It is thus likely a saddle point. This is supported by Figure 1, where there appears to be an intersection between two contour curves (both at height 2) at $(0,1)$.

In this application, we could have used an implementation of the steepest ascent method instead. This would have some advantages over the Newton method, specifically it guarantees that the function g increases in every step of the algorithm. The accompanying disadvantages would likely be a slower convergence, but it is possible that convergence would not be guaranteed. In addition, it introduces a "hyperparameter" which must be specified by the model designer.

```
## ----------------------------------------
## Local extrema found: g(1, -1) = 4
## Gradient(1, -1) = (0, 0)
## Hessian(1, -1) =
##      [,1] [,2]
## [1,]   -4    2
## [2,]    2   -2
```

```
## ----------------------------------------
## Local extrema found: g(0, 1) = 2
## Gradient(0, 1) = (0, 0)
## Hessian(0, 1) =
##      [,1] [,2]
## [1,]   -4   -2
## [2,]   -2    0
```

```
## ----------------------------------------
## Local extrema found: g(0, 1) = 2
## Gradient(0, 1) = (0, 0)
## Hessian(0, 1) =
##      [,1] [,2]
## [1,]   -4   -2
## [2,]   -2    0
```

```
## ----------------------------------------
## Local extrema found: g(0, 1) = 2
## Gradient(0, 1) = (0, 0)
## Hessian(0, 1) =
##      [,1] [,2]
## [1,]   -4   -2
## [2,]   -2    0
```

Question 2

a) The implemented function can be found in the appendix

b)

The two variants are done by altering the alpha value

```
## $values
## [1] -0.009354408  1.262803143
##
## $`gradient function count`
## [1] 66
##
## $`likelihood function count`
## [1] 132
```

```
## $values
## [1] -0.009352896  1.262807178
##
## $`gradient function count`
## [1] 70
##
## $`likelihood function count`
## [1] 140
```

As we increase the alpha value, the estimated values are around the same but the function and gradient count is slightly different.

c)

```
## $par
## [1] -0.009356112  1.262812883
##
## $value
## [1] 6.484279
##
## $counts
## function gradient
##       12        8
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
## $par
## [1] -0.009423433  1.262738266
##
## $value
## [1] 6.484279
##
## $counts
## function gradient
##       47       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Yes, the results for both are almost the same with some changes in the last few decimal places and both are almost same to the value found in b).
The function and gradient count is drastically lower when compared to b).
Also the gradient count for Nelder-Mead is NA since no computation of derivatives is necessary.

d)

```
##
## Call:  glm(formula = y ~ x, family = "binomial")
##
## Coefficients:
## (Intercept)            x
##    -0.00936      1.26282
##
## Degrees of Freedom: 9 Total (i.e. Null);  8 Residual
## Null Deviance:       13.46
## Residual Deviance: 12.97     AIC: 16.97
```

Based on the output from the “glm” function, we can say that the output compared to b) and c) are almost the same with some changes in the last few decimal places.

Appendix

Question 1

```
# Problem setup
g <- function(x){-x[1]^2 - x[1]^2*x[2]^2 - 2*x[1]*x[2] + 2*x[1] + 2}
g_gradient <- function(x){matrix(c(-2*x[1] - 2*x[2]^2*x[1] - 2*x[2]+2, -2*x[1]^2*x[2] -2*x[1]))}
g_Hessian <- function(x){matrix(c(-2-2*x[2]^2, -4*x[1]*x[2]-2, -4*x[1]*x[2]-2, -2*x[1]^2), nrow=2)} # b
```

```
# Contour plot

# Define the contour grid
x_values <- seq(-3, 3, 0.01)
y_values <- seq(-3, 3, 0.01)
z_values <- matrix(0,
                   nrow=length(x_values),
                   ncol=length(y_values),
                   dimnames = list(x_values, y_values))

# Calculate height over the grid
for (i in 1:length(x_values)){
  for (j in 1:length(y_values)){
    z_values[i, j] <- g(c(x_values[i], y_values[j]))
    #cat("New. x: ", x_i, " y:", y_i, " and value:", g(x_i, y_i), "\n")
```

```r
  }
}

# Make a contour plot
contour(x=x_values,
        y=y_values,
        z = z_values,
        main = "Fig 1. Contour plot of g(x,y)",
        xlab = "x",
        ylab = "y",
        nlevels = 150)
```

```r
library(stringr)

# The Newton method function
Newton <- function(x_start, func, grad, Hess, epsilon = 0.00001){
  # Setup
  estimates <- list(matrix(x_start))
  stop_condition <- FALSE

  while (stop_condition == FALSE){
    # Grab the last/previous x value
    prev_x <- tail(estimates, 1)[[1]]

    # Calculate new x value, and add to estimates list
    new_estim <- prev_x - solve(Hess(prev_x)) %*% grad(prev_x)
    estimates <- c(estimates, list(new_estim))

    # Calculate stopcondition
    first_vec <- tail(estimates, 1)[[1]]
    secon_vec <- tail(estimates, 2)[[1]]
    stop_condition <- (t(first_vec - secon_vec)  %*% (first_vec - secon_vec)) < epsilon
  }
  # Output
  cat("---------------------------------------\n")
  cat("Local extrema found: g(",
      str_c(round(new_estim, 4), collapse=", "),
      ") = ",
      round(func(new_estim), 4), "\n",

      "Gradient(",
      str_c(round(new_estim, 4), collapse=", "),
      ") = (",
      str_c(round(grad(new_estim), 4), collapse=", "), ")\n",

      "Hessian(",
      str_c(round(new_estim, 4), collapse=", "),
      ") = \n", sep="")

  print.default(round(Hess(new_estim), 4))

  return(new_estim)
}
```

```r
# Point 1
x1 <- Newton(c(2, 0), func = g, grad = g_gradient, Hess = g_Hessian)

# Point 2
x2 <- Newton(c(-1, -2), func = g, grad = g_gradient, Hess = g_Hessian)

# Point 3
x3 <- Newton(c(0, 1), func = g, grad = g_gradient, Hess = g_Hessian)

# Point 4
x4 <- Newton(c(0.5, -1), func = g, grad = g_gradient, Hess = g_Hessian)
```

Question-2

a)

```r
x <- c(0,0,0,0.1,0.1,0.3,0.3,0.9,0.9,0.9)
y<- c(0,0,1,0,1,1,1,0,1,1)


dose <- function(beta0,beta1,third){

  x <- c(0,0,0,0.1,0.1,0.3,0.3,0.9,0.9,0.9)
  y<- c(0,0,1,0,1,1,1,0,1,1)

  value<-list()

  b0<-beta0
  b1<-beta1
  alpha <- third
  orginal_x <- as.matrix(c(1,0))
  grad_count<-0
  main_count<-0

  derivative<- function(b0,b1){
    for (i in 1:length(x)){
      value[[i]]<-(y[i] - 1/(1+exp(-b0-b1*x[i])))*as.matrix(c(1,x[i]))
    }

    abc<-as.matrix(c(0,0))
    for (i in 1:length(x)){
      abc <- abc+ value[[i]]
    }
    grad_count<<-grad_count +1
    return(abc)
  }


  next_x<- function(b0,b1){
    as.matrix(c(b0,b1)) + alpha * derivative(b0,b1)
  }
  main_function<- function(b0,b1){
    comp<-c()
    for (i in 1:length(x)){
      comp[i]<-y[i]*log(solve(1+exp(-b0-b1*x[i]))) + (1-y[i])*log(1-solve((1+ exp(-b0-b1*x[i]))))
    }
    comp<-sum(comp)

    main_count<<- main_count+1
    return(comp)
  }

  i<-1

  store<-list(as.vector(orginal_x))

  #####
  i <- i + 1
  a<-next_x(orginal_x[1],orginal_x[2])
  while(main_function(a[1],a[2])<main_function(orginal_x[1],orginal_x[2])){
    alpha<-alpha/2
    a<-next_x(orginal_x[1],orginal_x[2])
  }

  orginal_x<-a
  store[[i]]<-as.vector(orginal_x)
  alpha<-1

  #####
  while(any(abs(store[[length(store)]]-store[[length(store)-1]])>0.00001)){

    alpha<-1
    i <- i + 1

    a<-next_x(orginal_x[1],orginal_x[2])

    while(main_function(a[1],a[2])<main_function(orginal_x[1],orginal_x[2])){
      alpha<-alpha/2

      a<-next_x(orginal_x[1],orginal_x[2])
    }


    orginal_x<-a
    store[[i]]<-as.vector(orginal_x)
  }

  #the values of b0 and b1 + the number of time the gradient and log (main) function has been called
  list("values" = store[[length(store)]],"gradient function count" =  grad_count,"likelihood function c
}
```

b)

```r
dose(-0.2,1,1)
dose(-0.2,1,5)
```

c)

```r
opt<- function(a){
  comp<-c()
  for (i in 1:10){
    comp[i]<-y[i]*log(solve(1+exp(-a[1]-a[2]*x[i]))) + (1-y[i])*log(1-solve((1+ exp(-a[1]-a[2]*x[i]))))
  }
  comp<- - sum(comp)
}

optim(c(-0.2,1),fn = opt , method = "BFGS")

optim(c(-0.2,1),fn = opt , method = "Nelder-Mead", hessian = FALSE)
```

d)

```r
glm(y~x,family = "binomial")
```