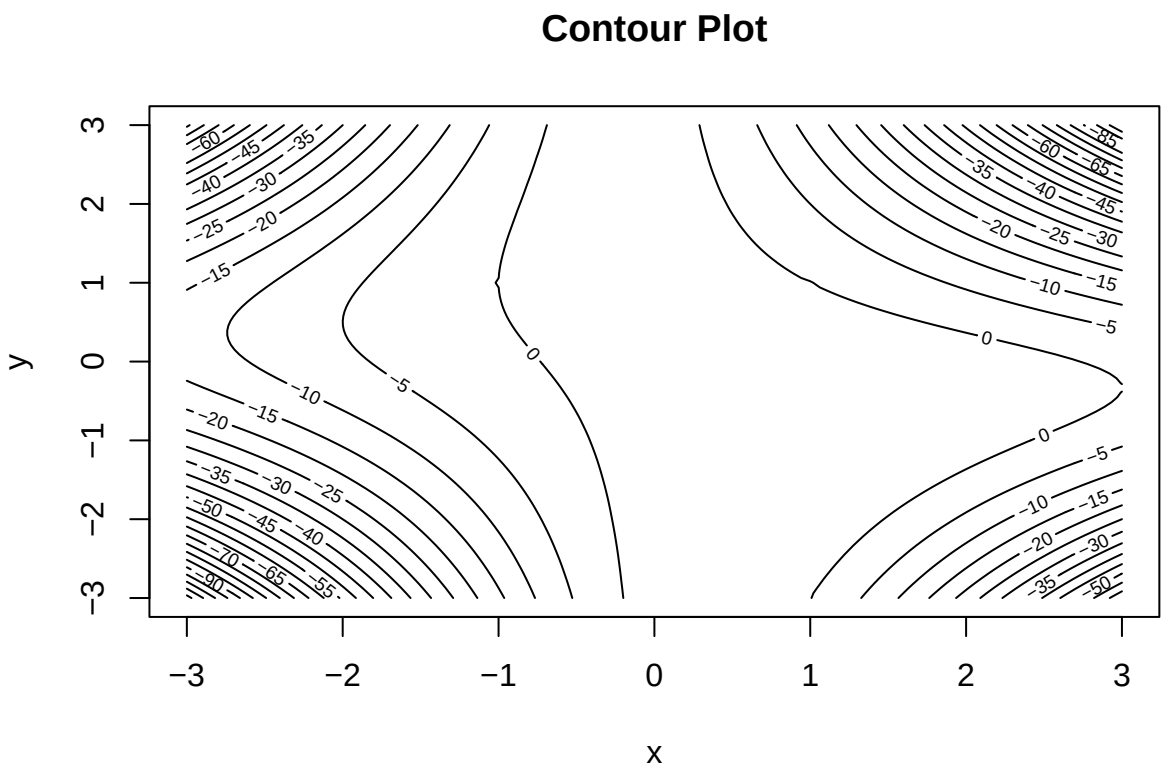
Lab 2

Ayesha Gamage-(ayega981) ,Muditha Cherangani-(mudch175)

2023-11-14

Question 1

a).




```
#Hessian Matrix
H <- function(x,y) {
  matrix(c((-2-2*y^2), (-4*x*y-2), (-4*x*y-2), (-2*x^2)), ncol = 2)
}
```

b).

```
# Newton's method for finding a local maximum
newton_method <- function(initial_guess, eps = 0.0001) {
  xy <- initial_guess
  xy1 <- initial_guess + c(2, 2)  # xy1 is xy(t-1); starting value here just to ensure
                                  #that while-condition is met

  while (sum(abs(xy - xy1)) > eps) {
    xy1 <- xy
    gradient <- c(Dg_dx(xy[1], xy[2]), Dg_dy(xy[1], xy[2]))
    hessian <- H(xy[1],xy[2])

    # Update using Newton's method
    xy <- xy1 - (solve(hessian) %*% gradient)
  }
  xy
}

# Initial guess
initial_guess <- c(0, 0)

# Apply Newton's method
result <- newton_method(initial_guess)

cat("Local maximum found at:(", result[1],",",result[2], ")\n")
```

```
## Local maximum found at:( 0 , 1 )
```

```
cat("Value of g at the local maximum:", g(result[1], result[2]), "\n")
```

```
## Value of g at the local maximum: 2
```

c).

```
## Starting value:: 2 0
## Converged to: 1 -1
## Gradient at converged point: -1.927586e-09 -5.43327e-09
## Hessian at converged point:
##      [,1] [,2]
## [1,]   -4    2
## [2,]    2   -2
## ****Local Maximum****
##
## Starting value:: -1 -2
## Converged to: 1.361391e-22 1
## Gradient at converged point: 0 -2.722781e-22
## Hessian at converged point:
##      [,1]          [,2]
## [1,]   -4 -2.000000e+00
## [2,]   -2 -3.706769e-44
```

```
## ****Saddle Point****
##
## Starting value:: 0 1
## Converged to: 0 1
## Gradient at converged point: 0 0
## Hessian at converged point:
##      [,1] [,2]
## [1,]   -4   -2
## [2,]   -2    0
## ****Saddle Point****
##
## Starting value:: -2 2
## Converged to: 2.409018e-15 1
## Gradient at converged point: 2.68896e-13 -4.818035e-15
## Hessian at converged point:
##      [,1]          [,2]
## [1,]   -4 -2.000000e+00
## [2,]   -2 -1.160673e-29
## ****Saddle Point****
```

The function plot near x=1 and y=-1, we'll see that the value of f(x,y) is higher. It is the Local Maximum. The other value of f(x,y) is Saddle Point. So we can find the global maximum for x, y $\in$ [-3, 3].

d)

The decision between employing the steepest ascent (gradient ascent) algorithm and Newton's algorithm is contingent upon the nature of the optimization problem under consideration. Here, we outline the merits and drawbacks associated with each approach.

Steepest Ascent (Gradient Ascent) Algorithm:

Advantages:

Memory Efficiency: In scenarios involving a high number of dimensions, the computation and storage of the Hessian matrix can be memory-intensive. The steepest ascent mitigates this demand. Disadvantages:

Sensitivity to Step Size: The efficacy of the steepest ascent is reliant on the chosen step size. Small step sizes may result in sluggish convergence, while large step sizes may lead to overshooting the optimal point. Newton's Algorithm:

Advantages:

Faster Convergence: Newton's method typically achieves a faster rate of convergence compared to the steepest ascent, particularly when the objective function exhibits well-behaved and well-conditioned contours. Disadvantages: Hessian Computation: Calculating and inverting the Hessian matrix can pose computational challenges, particularly in situations involving a high number of dimensions.

Question 2

a).

Following function for the steepest ascent method

```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)

logistic_function <- function(beta) {
  return(1 / (1 + exp(-(beta[1] + beta[2] * x))))
}
#loglikelihood funtion
g <- function(beta) {
  p <- logistic_function(beta)
  return(sum((y * log(p)) + ((1 - y) * log(1 - p))))
}
#partial derivatives
dg1 <- function(beta)
{
  p <- logistic_function(beta)
  return(sum(p - y))
}

dg2 <- function(beta)
{
  p <- logistic_function(beta)
  return(sum((p - y) * x))
}
#gradian mtrix
gradient <- function(beta)
{
  c(dg1(beta), dg2(beta))
}

#Steepest ascent function:
steepestasc <- function(beta0, eps=1e-8, alpha0=1)
{

  no_of_gradient_evaluation<-1
  no_of_function_evaluation<-0
  beta   <- beta0
  conv <- 999
  while(conv>eps)
  {
    alpha <- alpha0
    beta1   <- beta
    beta    <- (beta1 - alpha*gradient(beta1))

    while (g(beta)<g(beta1))
    {
      alpha <- (alpha/2)
      beta    <- (beta1 - alpha*gradient(beta1))
      no_of_gradient_evaluation <-no_of_gradient_evaluation+1
      no_of_function_evaluation<-no_of_function_evaluation + 2
    }
    no_of_function_evaluation<-no_of_function_evaluation + 2
    conv <- sum((beta-beta1)*(beta-beta1))
  }
```

```
  return(list(beta=beta,nof=no_of_function_evaluation,nog=no_of_gradient_evaluation))
}
```

b).

Here first I have tried starting value ($beta_0$, $beta_1$) = (-0.2, 1) and then tried with parameters ($beta_0$, $beta_1$) = (-1.9, 2).Following result list given by function. $beta_0$, $beta_1$ given in with name "beta" , number of function given by "nof" and number of gradient given by "nog".

```
## Given values for starting values -0.2, 1 :
```

```
##        beta1        beta2          nof          nog
## -0.007299539  1.254971983 42.000000000  8.000000000
```

```
## Given values for starting values -1.9, 2 :
```

```
##       beta1       beta2         nof         nog
## -0.01162386  1.26780485 66.00000000 10.00000000
```

The optimization algorithm may converge faster or slower depending on the local minima or flat regions in close of each starting point.The choice of the initial point might influence the convergence path, and the algorithm might get stuck in different regions of the parameter space.Given result shows that and the number of function evaluations is less than for stating point (-0.2,1) than stating point (-1.9,2).Since (-0.2,1) is close to maxima ~(0,1).

c).

Optim functions

```
    negative_g <- function(beta) {
      p <- logistic_function(beta)
      return(-sum((y * log(p)) + ((1 - y) * log(1 - p))))
    }
    cat("Optim function with method BFGS \n")
```

```
## Optim function with method BFGS
```

```
    optim_result_BFGS <- optim(par = c(-0.2, 1), fn = negative_g, method="BFGS")
    print(unlist(optim_result_BFGS))
```

```
##            par1            par2           value counts.function counts.gradient
##    -0.009356112     1.262812883     6.484278783    12.000000000     8.000000000
##     convergence
##     0.000000000
```

```
    cat("Optim function with method Nelder-Mead \n")
```

```
## Optim function with method Nelder-Mead
```

```r
    optim_result_Nelder <- optim(par = c(-0.2, 1), fn = negative_g, method = "Nelder-Mead")
    print(unlist(optim_result_Nelder))
```

```
##            par1            par2           value counts.function counts.gradient
##    -0.009423433     1.262738266     6.484278793    47.000000000              NA
##     convergence
##     0.000000000
```

When compare with result given by b,values are not same. $beta_0$ values different and $beta_1$ values are very closed. In b,if we increase the precision we will get a result as close as optim function. Following example given the closed result to optim function,

```
##         beta1         beta2           nof           nog
##  -0.009421232   1.263060632 102.000000000  15.000000000
```

When consider about result of optim function, in "BFGS" method given result has 12 number of function and 8 gradient of evaluations.But in "Nelder-Mead" method has high number of function but no gradient of evaluations.

d).

Glm function,

```
##
## Call:
## glm(formula = y ~ x, family = binomial, data = data)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.00936    0.87086  -0.011    0.991
## x            1.26282    1.86663   0.677    0.499
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13.460  on 9  degrees of freedom
## Residual deviance: 12.969  on 8  degrees of freedom
## AIC: 16.969
##
## Number of Fisher Scoring iterations: 4
```

In this result $beta_0$ given by estimate intercept and $beta_1$ given by estimate of x. Those values are same as given values of c. When compare with values in b, $beta_1$ has close values but $beta_0$ values are different.

Appendix: All r code for this report

```r
knitr::opts_chunk$set(echo = TRUE)
# Define the function g(x, y)
g <- function(x, y) {
```

```
  -x^2 - x^2*y^2 - 2*x*y + 2*x + 2
}

# Partial derivatives
Dg_dx <- function(x, y) {
  -2*x - 2*x*y^2 - 2*y + 2
}
Dg_dy <- function(x, y) {
  -2*x^2*y - 2*x
}

# Second partial derivatives
D2g_dx2 <- function(x, y) {
  -2 - 2*y^2
}
D2g_dy2 <- function(x, y) {
  -2*x^2
}
D2g_dxdy <- function(x, y) {
  -4*x*y - 2
}

gradient <- function(x, y) {
  c((-2*x - 2*x*y^2 - 2*y + 2), (-2*x^2*y - 2*x))
}

# prepare contour/3d-plot
plot_contour <-function(){
  x1grid <- seq(-3, 3, length.out = 100)
  x2grid <- seq(-3, 3, length.out = 100)
  dx1 <- length(x1grid)
  dx2 <- length(x2grid)
  dx  <- dx1*dx2
  gx  <- matrix(rep(NA, dx), nrow=dx1)
  for (i in 1:dx1)
    for (j in 1:dx2)
    {
      gx[i,j] <- g(x1grid[i], x2grid[j])
    }
  mgx <- matrix(gx, nrow=dx1, ncol=dx2)
  contour(x1grid, x2grid, mgx, nlevels=20,xlab="x",ylab="y",main="Contour Plot")
}

plot_contour()
#Hessian Matrix
H <- function(x,y) {
  matrix(c((-2-2*y^2), (-4*x*y-2), (-4*x*y-2), (-2*x^2)), ncol = 2)
}

# Newton's method for finding a local maximum
newton_method <- function(initial_guess, eps = 0.0001) {
  xy <- initial_guess
  xy1 <- initial_guess + c(2, 2)  # xy1 is xy(t-1); starting value here just to ensure
```

```
                                #that while-condition is met

  while (sum(abs(xy - xy1)) > eps) {
    xy1 <- xy
    gradient <- c(Dg_dx(xy[1], xy[2]), Dg_dy(xy[1], xy[2]))
    hessian <- H(xy[1],xy[2])

    # Update using Newton's method
    xy <- xy1 - (solve(hessian) %*% gradient)
  }
  xy
}

# Initial guess
initial_guess <- c(0, 0)

# Apply Newton's method
result <- newton_method(initial_guess)

cat("Local maximum found at:(", result[1],",",result[2], ")\n")
cat("Value of g at the local maximum:", g(result[1], result[2]), "\n")


# Set of different starting values
starting_values <- list(c(2, 0), c(-1, -2), c(0, 1), c(-2, 2))

# Run Newton's method for each starting value
for (start_val in starting_values) {

  cat("Starting value::", start_val, "\n")
  result <- newton_method(start_val)

  cat("Converged to:", result, "\n")

  # Calculate gradient and Hessian at the final converged point
  gradient_final <- c(Dg_dx(result[1], result[2]), Dg_dy(result[1], result[2]))
  hessian_final <- H(result[1], result[2])

  cat("Gradient at converged point:", gradient_final, "\n")
  cat("Hessian at converged point:\n")
  print(hessian_final)
  eigenvalues <- eigen(hessian_final)$values

  # Check the signs of eigenvalues
  if(all(eigenvalues > 0)) {
    cat("****Local Minimum****\n\n")
  } else if(all(eigenvalues < 0)) {
    cat("****Local Maximum****\n\n")
  } else {
    cat("****Saddle Point****\n\n")
  }
}
```

```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)

logistic_function <- function(beta) {
  return(1 / (1 + exp(-(beta[1] + beta[2] * x))))
}
#loglikelihood funtion
g <- function(beta) {
  p <- logistic_function(beta)
  return(sum((y * log(p)) + ((1 - y) * log(1 - p))))
}
#partial derivatives
dg1 <- function(beta)
{
  p <- logistic_function(beta)
  return(sum(p - y))
}

dg2 <- function(beta)
{
  p <- logistic_function(beta)
  return(sum((p - y) * x))
}
#gradian mtrix
gradient <- function(beta)
{
  c(dg1(beta), dg2(beta))
}

#Steepest ascent function:
steepestasc <- function(beta0, eps=1e-8, alpha0=1)
{

  no_of_gradient_evaluation<-1
  no_of_function_evaluation<-0
  beta   <- beta0
  conv <- 999
  while(conv>eps)
  {
    alpha <- alpha0
    beta1   <- beta
    beta    <- (beta1 - alpha*gradient(beta1))

    while (g(beta)<g(beta1))
    {
      alpha <- (alpha/2)
      beta    <- (beta1 - alpha*gradient(beta1))
      no_of_gradient_evaluation <-no_of_gradient_evaluation+1
      no_of_function_evaluation<-no_of_function_evaluation + 2
    }
    no_of_function_evaluation<-no_of_function_evaluation + 2
    conv <- sum((beta-beta1)*(beta-beta1))
  }
```

```
  return(list(beta=beta,nof=no_of_function_evaluation,nog=no_of_gradient_evaluation))
}
cat("Given values for starting values -0.2, 1 :\n")
  unlist(steepestasc(c(-0.2, 1),1e-5))
cat("Given values for starting values -1.9, 2 :\n")
    unlist(steepestasc(c(-1.9, 2),1e-5))
    negative_g <- function(beta) {
      p <- logistic_function(beta)
      return(-sum((y * log(p)) + ((1 - y) * log(1 - p))))
    }
    cat("Optim function with method BFGS \n")
    optim_result_BFGS <- optim(par = c(-0.2, 1), fn = negative_g, method="BFGS")
    print(unlist(optim_result_BFGS))

    cat("Optim function with method Nelder-Mead \n")
    optim_result_Nelder <- optim(par = c(-0.2, 1), fn = negative_g, method = "Nelder-Mead")
    print(unlist(optim_result_Nelder))
unlist(steepestasc(c(-1.9, 2),1e-8))
    data <- data.frame(x = x, y = y)
    model <- glm(y ~ x, data = data, family = binomial)
    summary(model)
```