Lab 1 - Computational Statistics (732A89)

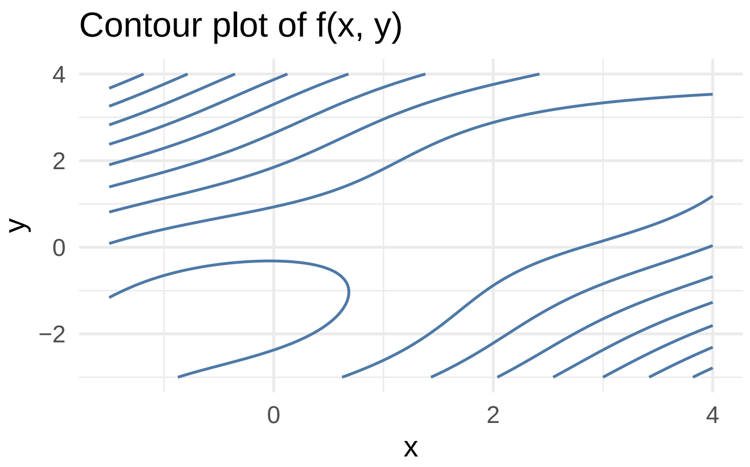
Helena Llorens Lluís (hllor282), Yi Yang (yiyan338)

QUESTION 1

First, we plot the function

$$f(x, y) = sin(x + y) + (x - y)^2 - 1.5x + 2.5y + 1$$

within the intervals $x \in [-1.5, 4]$ and $y \in [-3, 4]$.



With both the gradient and the Hessian matrix calculated, we have been able to compute the Newton algorithm to find local minimums of this function. This function takes as input both the gradient and the Hessian matrix, a pair of starting values x_0 and y_0, and a tolerance (set as default at 0.0001) and a maximum of iterations (set at 100 as default). We tried the function for 3 different starting points and the results are presented in the following table.

Table 1: Newton algorithm results

initial_x	initial_y	final_x	final_y	function_value	eigenvalues1	eigenvalues2
1	3	2.5943951	1.5943951	1.228370	4	1.732051
2	4	1.5471976	0.5471976	1.913223	4	-1.732051
0	-1	-0.5471976	-1.5471976	-1.913223	4	1.732051

A result is considered a local minimum if all its eigenvalues are positive. As shown in the table, two of the three solutions satisfy this condition: (2.5943951, 1.5943951) and $(-0.5471976, -1.5471976)$. Among them, we identify the latter as the global minimum, as it yields the lowest function value, -1.913223.

On the other hand, (1.5471976, 0.5471976) is classified as a saddle point since one of its eigenvalues is negative.

Appendix

```r
# define the function f(x, y) that returns f(x, y) if both x and y are in the interval
f <- function(x, y){
  ifelse(x >= -1.5 & x <= 4 & y >= -3 & y <= 4,
         sin(x + y) + (x - y)^2 - 1.5*x + 2.5*y + 1,
         NA)
}

# data frame with x, y and f(x, y)
x <- seq(-1.5, 4, length.out = 100)
y <- seq(-3, 4, length.out = 100)

grid <- expand.grid(x = x, y = y)
grid$z <- with(grid, f(x, y))

# plot f(x, y)
ggplot(grid, aes(x = x, y = y, z = z)) +
  geom_contour(col = "#4E79A7") +
  labs(x = "x", y = "y", title = "Contour plot of f(x, y)") +
  theme_minimal()

# gradient for f(x, y)
gradient <- function(x, y){
  grad_x <- cos(x + y) + 2*(x - y) - 1.5
  grad_y <- cos(x + y) - 2*(x - y) + 2.5
  return(c(grad_x, grad_y))
}

# hessian matrix for f(x, y)
hessian <- function(x, y){

  m <- matrix(c(-sin(x + y) + 2, -sin(x + y) - 2,
                -sin(x + y) - 2, -sin(x + y) + 2),
                byrow = T, nrow = 2, ncol = 2)

  return(m)
}

# newton algorithm
newton <- function(gradient, hessian, x_init, y_init, tol = 0.0001, maxit = 100){

  # check that the initial values are within the intervals
  if(x_init < -1.5 || x_init > 4 || y_init < -3 || y_init > 4){
    stop("The x and y values have to be within the intervals")
  }

  # set initial values
  x0 <- c(x_init, y_init)

  # while the stopping criteria is not met
  for(it in 1:maxit){
    H <- hessian(x0[1], x0[2])

    # use MASS library in case of a singular Hessian matrix
    if(abs(det(H)) < 1e-6){
      x1 <- x0 - as.vector(MASS::ginv(H) %*% gradient(x0[1], x0[2]))
    } else{
      x1 <- x0 - as.vector(solve(H) %*% gradient(x0[1], x0[2]))
    }

    # set the values within the intervals
    x1[1] <- max(min(x1[1], 4), -1.5)
    x1[2] <- max(min(x1[2], 4), -3)

    # if the stopping criteria is met
    if(norm(x1 - x0, type = "2") < tol){  # t(x1 - x0) %*% (x1 - x0)
      return(list(x = x1[1], y = x1[2], iterations = it,
               gradient = gradient(x1[1], x1[2]), eigenvalues = eigen(hessian(x1[1], x1[2]))$values))
    }

    x0 <- x1
  }

  warning("Maximum number of iterations reached without convergence.")
  return(list(x = x1[1], y = x1[2], iterations = maxit,
           gradient = gradient(x1[1], x1[2]), eigenvalues = eigen(hessian(x1[1], x1[2]))$values))
}

x_init1 <- 1
y_init1 <- 3
res1 <- newton(gradient, hessian, x_init1, y_init1)

x_init2 <- 2
y_init2 <- 4
res2 <- newton(gradient, hessian, x_init2, y_init2)

x_init3 <- 0
y_init3 <- -1
res3 <- newton(gradient, hessian, x_init3, y_init3)

df <- data.frame(initial_x = c(x_init1, x_init2, x_init3),
                 initial_y = c(y_init1, y_init2, y_init3),
                 final_x = c(res1$x, res2$x, res3$x),
                 final_y = c(res1$y, res2$y, res3$y),
                 function_value= c(f(res1$x, res1$y), f(res2$x, res2$y), f(res3$x, res3$y)),
                 eigenvalues1 = c(res1$eigenvalues[1], res2$eigenvalues[1], res3$eigenvalues[1]),
                 eigenvalues2 = c(res1$eigenvalues[2], res2$eigenvalues[2], res3$eigenvalues[2]))
kable(df, caption = "Newton algorithm results")
```