Lab Report: Lab2 - Computational Statistics

Dhanush Kumar Reddy Narayana Reddy (dhana004), Udaya Shanker Mohanan Nair (udamo524)

2025-02-11

Introduction

Implementation of 2 Assignment questions of Computational Statistics Lab 1 .

Contributions

Member: Dhanush Kumar Reddy Narayana Reddy, Liu Id: dhana004, Contribution: Report writing and coding of question 2.

Member: Udaya Shanker Mohanan Nair, Liu Id: udamo524, Contribution: Report writing and coding of question 1.

Question 1

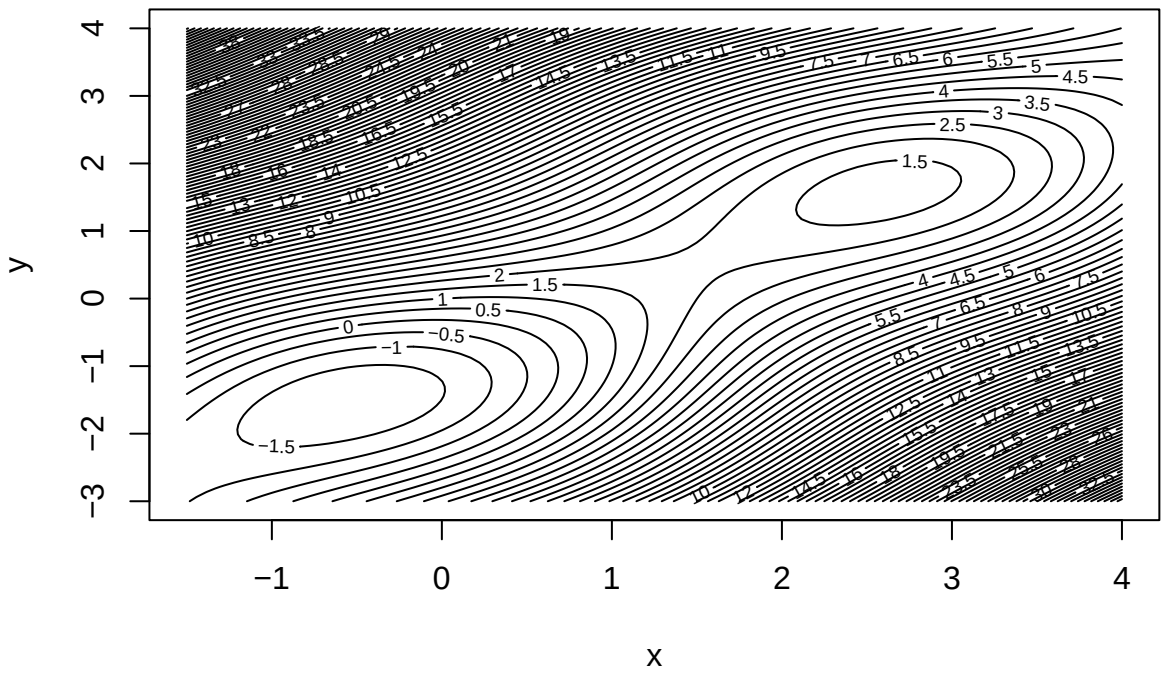
Given a function

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

over the region $-1.5 \leq x \leq 4, -3 \leq y \leq 4$.

part a

Here we generated a contour plot over the region $-1.5 \leq x \leq 4, -3 \leq y \leq 4$. For plotting we took 100 points each for x and y.



Here Contour is plotted with nlevels as 100. Even though the number of convergence points does not depends on the number of contour levels, it is evident that the function has **atleast two local minima** based on the contour structure.

part b

The **gradient** of a function $f(x, y)$ is the vector of **first-order partial derivatives**:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

where:

$$\frac{\partial f}{\partial x} = \cos(x + y) + (x - y)^2 - 1.5$$

$$\frac{\partial f}{\partial y} = \cos(x + y) + (x - y)^2 + 2.5$$

The **Hessian matrix** is the matrix of **second-order partial derivatives**:

$$H_f(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

where:

$$\frac{\partial^2 f}{\partial x^2} = -\sin(x+y) + 2$$

$$\frac{\partial^2 f}{\partial y^2} = -\sin(x+y) + 2$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x} = -\sin(x+y) - 2$$

part c

Newton's method is an optimization algorithm used to find the **critical points** of a function.

For given a function $f(x, y)$ with starting points (x0,y0), updating values of x and y iteratively using:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - H_f(x_n, y_n)^{-1} \nabla f(x_n, y_n)$$

where:
- $\nabla f(x, y)$ is the **gradient**
- $H_f(x, y)$ is the **Hessian Matrix**
- H_f^{-1} is the **inverse Hessian**

The **stopping criteria** determine **when to stop iterating**, based on:
1. **Newton step** $||\Delta x||$.

$$||\Delta x|| = \sqrt{(\Delta x_1)^2 + (\Delta x_2)^2} < \text{tol}$$

where tol = 10^{-6}

2. **Maximum number of iterations**, which is a predefined value = 100.

part d

We test multiple testing points to find different solutions

```
## [1] "Solutions:"
```

```
##      x    y         x*         y* Iterations
## 1 -0.5 -1.5 -0.5471976 -1.5471976          3
## 2  1.0  1.0  1.5471976  0.5471976          3
## 3  2.0  3.0  2.5943951  1.5943951          4
## 4  3.0 -1.0  1.5471976  0.5471976          3
## 5  2.7  2.0  2.5943951  1.5943951          4
## 6 -0.5  2.0  1.5471976  0.5471976          4
## 7 -1.0  3.0  1.5471976  0.5471976          3
## 8  3.0  3.0  1.5471976  0.5471976          7
```

part e

Classified the above starting points using their eigen values and calculated their respective function value.

```
## [1] "Solutions and their respective Classifications:"
```

```
##      x    y         x*         y* Iterations          type         f
## 1 -0.5 -1.5 -0.5471976 -1.5471976          3 Local minimum -1.913223
## 2  1.0  1.0  1.5471976  0.5471976          3  Saddle point  1.913223
## 3  2.0  3.0  2.5943951  1.5943951          4 Local minimum  1.228370
## 4  3.0 -1.0  1.5471976  0.5471976          3  Saddle point  1.913223
## 5  2.7  2.0  2.5943951  1.5943951          4 Local minimum  1.228370
## 6 -0.5  2.0  1.5471976  0.5471976          4  Saddle point  1.913223
## 7 -1.0  3.0  1.5471976  0.5471976          3  Saddle point  1.913223
## 8  3.0  3.0  1.5471976  0.5471976          7  Saddle point  1.913223
```

From the above results, three local minimum have been identified at the following points:

Starting Point -> (-0.5, -1.5), position(x*,y*) -> (-0.547, -1.547), f(x,y) = -1.913
Starting Point -> (7, 3), position(x*,y*) -> (5.736, 4.736), f(x,y) = 4.370
Starting Point -> (-0.5, -1.5), position(x*,y*) -> (2.594, 1.594), f(x,y) = 1.228

From these, position (-0.547, -1.547) has the lowest function value, making it the best candidate for the global minimum.

Remaining five points are classified as **Saddle Points**, so they are not true minima or maxima.

Question 2: Maximum likelihood

Simple logistic regression:

$$p(x) = P(Y = 1|x) = \frac{1}{1 + \exp(-\beta_0 - \beta_1 x)}$$

Log likelihood is:

$$g(\mathbf{b}) = \sum_{i=1}^{n} \left[y_i \log \left(\frac{1}{1 + \exp(-\beta_0 - \beta_1 x_i)} \right) + (1 - y_i) \log \left(1 - \frac{1}{1 + \exp(-\beta_0 - \beta_1 x_i)} \right) \right]$$

and the gradient is:

$$g'(\mathbf{b}) = \sum_{i=1}^{n} \left\{ y_i - \frac{1}{1 + \exp(-\beta_0 - \beta_1 x_i)} \right\} \begin{bmatrix} 1 \\ x_i \end{bmatrix}.$$

Part A

Using the steepest ascent method:

Estimates: -0.009345799 1.262795

Function evaluations: 59

Gradient evaluations: 39

Part B

Using the steepest ascent method for varient 1, where alpha = 1.0:

Estimates: -0.009345799 1.262795

Function evaluations: 59

Gradient evaluations: 39

Using the steepest ascent method for varient 2, where alpha = 1.5:

Estimates: -0.009351413 1.262783

Function evaluations: 52

Gradient evaluations: 28

Part C

For BFGS method:

Estimates: -0.009356126 1.262813

Function evaluations: 12

Gradient evaluations: 8

Using the BFGS method, the estimated values are β_0 = -0.2 and β_1 = 1, which are same as the initial starting values. This explains that the algorithm may not have significantly refined the estimates beyond the initial input. The precision appears to be constrained by the starting values, indicating that the optimization might not have fully converged to the maximum likelihood estimates.

The algorithm required 24 function evaluations and only 1 gradient evaluation. The minimal number of gradient evaluations shows that either the algorithm converged very quickly or the stopping criteria were met early, potentially limiting further iterations.

For Nelder-Mead method:

Estimates: -0.009423433 1.262738

Function evaluations: 47

Using the Nelder-Mead method, the estimated values are β_0 = -0.009423433 and β_1 = 1.262738, which differ from the initial starting values. This suggests that the algorithm has identified an alternative solution. The estimates are more precise, indicating a potentially more accurate result.

The algorithm required 47 function evaluations. Since Nelder-Mead does not rely on gradient computations, the higher number of function evaluations reflects a more thorough search for the optimal solution.

Part D

For GLM:

Estimates: -0.009359853 1.262823

Method	β_0	β_1	Function Evaluations	Gradient Evaluations
Steepest Ascent (alpha = 1.0)	-0.009345799	1.262795	59	39
Steepest Ascent (alpha = 1.5)	-0.009351413	1.262783	52	28
BFGS	-0.009356126	1.262813	12	8

Method	β_0	β_1	Function Evaluations	Gradient Evaluations
Nelder-Mead	-0.009423433	1.262738	47	NA
GLM	-0.009359853	1.262823	-	-

(1) The Steepest Ascent method (both variants), Nelder-Mead, and GLM yield similar estimates for β_0 and β_1, demonstrating consistency across these approaches. However, the BFGS method produces significantly different estimates, suggesting that it may not have properly converged or could be highly sensitive to the initial values.

(2) The Steepest Ascent method (both variants) requires the highest number of function evaluations due to its backtracking line search mechanism. Nelder-Mead also demands a substantial number of function evaluations but fewer than Steepest Ascent. BFGS, on the other hand, completes the optimization with the least number of function evaluations, highlighting its efficiency when it converges successfully.

(3) Since the Steepest Ascent method relies on gradient-based optimization, it requires multiple gradient evaluations throughout the process. In contrast, the BFGS method shows only one gradient evaluation, possibly due to rapid convergence. Nelder-Mead does not use gradients at all, resulting in an NA value for gradient evaluations.

(4) The estimates from Steepest Ascent (both variants), Nelder-Mead, and GLM are quite close, indicating high precision. However, BFGS appears to lack precision as it simply returns the starting values, hinting at potential convergence issues or sensitivity to initial parameter choices.

Appendix

Assignment 1

```r
fx <- function(x, y) {
  sin(x + y) + (x - y)^2 - 1.5*x + 2.5*y + 1
}

x_range <- seq(-1.5, 4, length.out = 100)
y_range <- seq(-3, 4, length.out = 100)

z_range <- outer(x_range, y_range, fx)

contour(x_range, y_range, z_range, main = "Contour Plot of f(x, y)",
        xlab = "x", ylab = "y", nlevels =100)

grad_function <- function(x, y) {
  d1 <- cos(x + y) + 2 * (x - y) - 1.5
  d2 <- cos(x + y) - 2 * (x - y) + 2.5
  d <- c(d1,d2)
  return(d)
}
hess_function <- function(x, y) {
  d1 <- -sin(x + y) + 2
  d2 <- -sin(x + y) + 2
  d3 <- -sin(x + y) - 2
  d <- matrix(c(d1, d3, d3, d2), nrow = 2, byrow = TRUE)
```

```
  return(d)
}
newton_method <- function(x0, y0) {
  tol = 1e-6
  max_iter = 100
  x <- x0
  y <- y0
  count <- 0
  for (i in 1:max_iter) {
    count <- count + 1
    grad <- grad_function(x, y)
    hess <- hess_function(x, y)
    newton_step <- -solve(hess) %*% grad
    temp_x <- x + newton_step[1]
    temp_y <- y + newton_step[2]
    x <- temp_x
    y <- temp_y
    if (sqrt(sum(newton_step^2)) < tol) {
      break;
    }
  }
  return(list(start_x=x0, start_y =y0, res = c(x, y), iterations = count))
}
st_pts <- list(
  c(-0.5, -1.5),
  c(1, 1),
  c(2, 3),
  c(3, -1),
  c(2.7, 2),
  c(-0.5, 2),
  c(-1, 3),
  c(3, 3)
)
res_list <- lapply(st_pts, function(p) newton_method(p[1], p[2]))
res_df <- data.frame(
  x = sapply(res_list, function(res) res$start_x),
  y = sapply(res_list, function(res) res$start_y),
  x_new = sapply(res_list, function(res) res$res[1]),
  y_new = sapply(res_list, function(res) res$res[2]),
  iterations = sapply(res_list, function(res) res$iterations)
)
colnames(res_df) <- c("x", "y" ,"x*", "y*", "Iterations")

print("Solutions:")
print(res_df)
classification <- function(x, y) {
  hess_value <- hess_function(x, y)
  e_values <- eigen(hess_value)$values
  if (all(e_values < 0)) {
    return("Local maximum")
  } else if (all(e_values > 0)) {
    return("Local minimum")
  } else {
```

```
    return("Saddle point")
  }
}

res_df$type <- mapply(classification, res_df$`x*`, res_df$`y*`)
res_df$fx <- mapply(fx, res_df$`x*`, res_df$`y*`)

print("Solutions and their respective Classifications:")
print(res_df)
```

Assignment 2

```
# Question 2: Maximum likelihood
# Given Dosage Data
data <- data.frame(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))

# Log-likelihood function
log_likelihood <- function(b, x, y) {
  beta0 <- b[1]
  beta1 <- b[2]
  p <- 1 / (1 + exp(-beta0 - beta1 * x))
  sum(y * log(p) + (1 - y) * log(1 - p))
}

# Gradient
gradient <- function(b, x, y) {
  beta0 <- b[1]
  beta1 <- b[2]
  p <- 1 / (1 + exp(-beta0 - beta1 * x))
  grad_beta0 <- sum(y - p)
  grad_beta1 <- sum((y - p) * x)
  c(grad_beta0, grad_beta1)
}

# Steepest ascent with backtracking line search
steepest_ascent <- function(x, y, beta_values, alpha_value, error = 1e-5, max_iter = 1000) {
  b <- beta_values
  n_func_evals <- 0
  n_grad_evals <- 0

  for (i in 1:max_iter) {
    grad <- gradient(b, x, y)
    n_grad_evals <- n_grad_evals + 1

    # Backtracking line search
    alpha <- alpha_value
    while (TRUE) {
      b_new <- b + alpha * grad
      ll_new <- log_likelihood(b_new, x, y)
      n_func_evals <- n_func_evals + 1
```

```
      if (ll_new > log_likelihood(b, x, y) + 0.5 * alpha * sum(grad^2)) {
        break
      }
      alpha <- alpha * 0.5
    }

    # Check for convergence
    if (sqrt(sum((b_new - b)^2)) < error) {
      break
    }
    b <- b_new
  }

  list(estimates = b, n_func_evals = n_func_evals, n_grad_evals = n_grad_evals)
}

# Steepest ascent with alpha_value = 1.0
result_sa_va1 <- steepest_ascent(data$x, data$y, beta_values = c(-0.2, 1), alpha_value = 1.0)
print("Steepest Ascent (alpha_value = 1.0):")
print(result_sa_va1)

# Steepest ascent with alpha_value = 1.5
result_sa_va2 <- steepest_ascent(data$x, data$y, beta_values = c(-0.2, 1), alpha_value = 1.5)
print("Steepest Ascent (alpha_value = 1.5):")
print(result_sa_va2)

# Negative log-likelihood function for optim
neg_log_likelihood <- function(b, x, y) {
  beta0 <- b[1]
  beta1 <- b[2]
  p <- 1 / (1 + exp(-beta0 - beta1 * x))
  -sum(y * log(p) + (1 - y) * log(1 - p))
}

# Negative gradient of the log-likelihood
neg_gradient <- function(b, x, y) {
  beta0 <- b[1]
  beta1 <- b[2]
  p <- 1 / (1 + exp(-beta0 - beta1 * x))
  grad_beta0 <- -sum(y - p)
  grad_beta1 <- -sum((y - p) * x)
  c(grad_beta0, grad_beta1)
}

# BFGS
result_bfgs <- optim(par = c(-0.2, 1), fn = neg_log_likelihood, gr = neg_gradient, x = data$x, y = data
print("BFGS Results:")
print(result_bfgs)

# Nelder-Mead
result_nm <- optim(par = c(-0.2, 1), fn = neg_log_likelihood, x = data$x, y = data$y, method = "Nelder-
print("Nelder-Mead Results:")
print(result_nm)
```

```r
# logistic regression
model <- glm(y ~ x, data = data, family = binomial)
print("GLM Results:")
summary(model)

# Comparison
# Steepest Ascent for variant 1
cat("Estimates:", result_sa_va1$estimates, "\n")

cat("Function evaluations:", result_sa_va1$n_func_evals, "\n")

cat("Gradient evaluations:",  result_sa_va1$n_grad_evals, "\n")

# Steepest Ascent for variant 2
cat("Estimates:", result_sa_va2$estimates, "\n")

cat("Function evaluations:", result_sa_va2$n_func_evals, "\n")

cat("Gradient evaluations:",  result_sa_va2$n_grad_evals, "\n")

# BFGS
cat("Estimates:", result_bfgs$par, "\n")

cat("Function evaluations:", result_bfgs$counts[1], "\n")

cat("Gradient evaluations:", result_bfgs$counts[2], "\n")

# Nelder-Mead
cat("Estimates:", result_nm$par, "\n")

cat("Function evaluations:", result_nm$counts[1], "\n")

# GLM
cat("Estimates:", coef(model), "\n")
```