Lab2

Cui Qingxuan, Nisal Amashan

2025-02-04

Contents

1 Collaborations **1**

2 Question 1 **1**

2.1 Contour Plot for the Function 1

2.2 Derive the Gradient and Hessian Matrix 2

2.3 Implement Newton Method 2

2.4 Test Method 3

2.5 Summary 3

3 Question2 **4**

3.1 Compute the ML-estimator 4

3.2 Comparison of Two Backtracking Line Search Variants 5

3.3 Comparison Using `optim` with BFGS and Nelder-Mead 5

3.4 ML Estimation Using `glm` and Comparison with Previous Results 6

4 Appendix **6**

1 Collaborations

Cui Qingxuan: Responsible for the question 1.

Nisal Amashan: Responsible for the question 2.

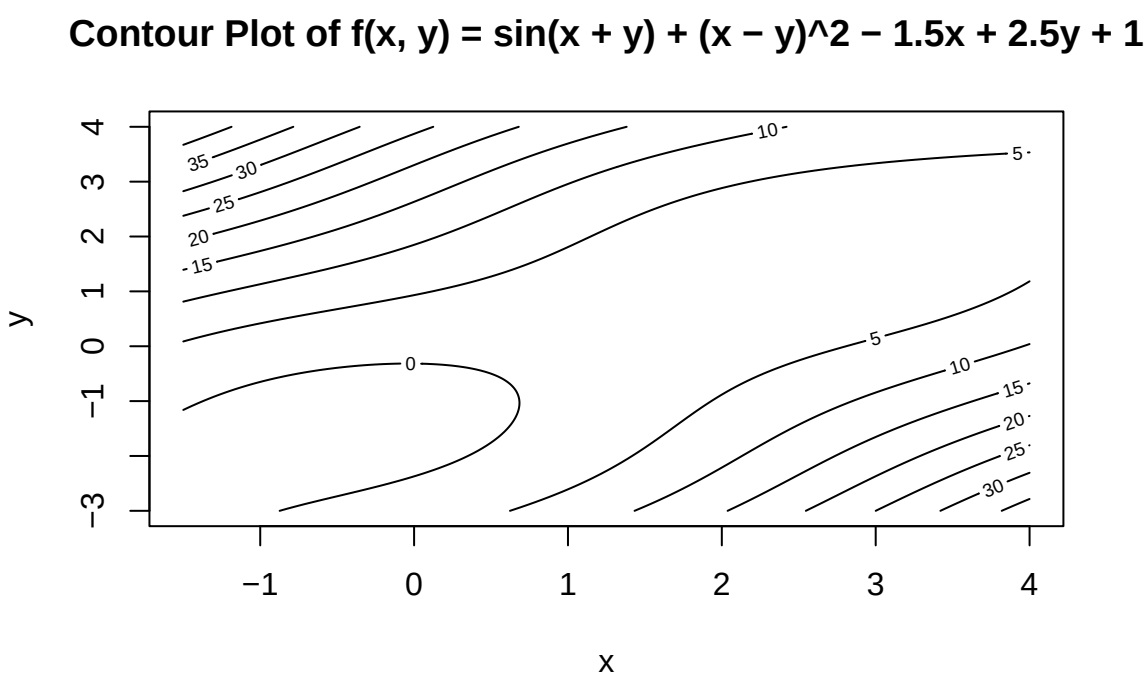
2 Question 1

2.1 Contour Plot for the Function

The function:

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

The plot:



2.2 Derive the Gradient and Hessian Matrix

```
## Assume that x = 1.5 y = 2
```

```
## Gradient:
```

-3.436
2.564

```
## Hessian Matrix
```

2.351	-1.649
-1.649	2.351

2.3 Implement Newton Method

```
newton = function(x0, y0){
  cad = list()
  for(i in 1:length(x0)){
    xt = matrix(c(x0[i], y0[i]), ncol=1)
    while(TRUE){
      dev_mul = solve(hessian_M(xt[1], xt[2])) %*% gradient(xt[1], xt[2])
      xt1 = xt - dev_mul
```

```
      if(all(abs(xt - xt1) < 0.001)){
        cad = append(cad, list(xt1))
        break
      }
      xt = xt1
    }
  }
  return(cad)
}
```

2.4 Test Method

start_points	candidates	eigen_values	function_value
(-0.49, -2.05)	(-0.55, -1.55)	(4.00, 1.73)	-1.91
(1.60, -0.13)	(1.55, 0.55)	(4.00, -1.73)	1.91
(2.53, 2.62)	(2.59, 1.59)	(4.00, 1.72)	1.23
(0.84, 1.12)	(1.55, 0.55)	(4.00, -1.73)	1.91
(0.39, -2.33)	(-0.55, -1.55)	(4.00, 1.73)	-1.91
(1.69, -1.53)	invalid	invalid	invalid
(1.52, 2.44)	(2.59, 1.59)	(4.00, 1.72)	1.23
(-0.03, 2.22)	(1.55, 0.55)	(4.00, -1.73)	1.91
(1.24, 0.12)	(1.55, 0.55)	(4.00, -1.73)	1.91
(0.28, -0.03)	invalid	invalid	invalid

2.5 Summary

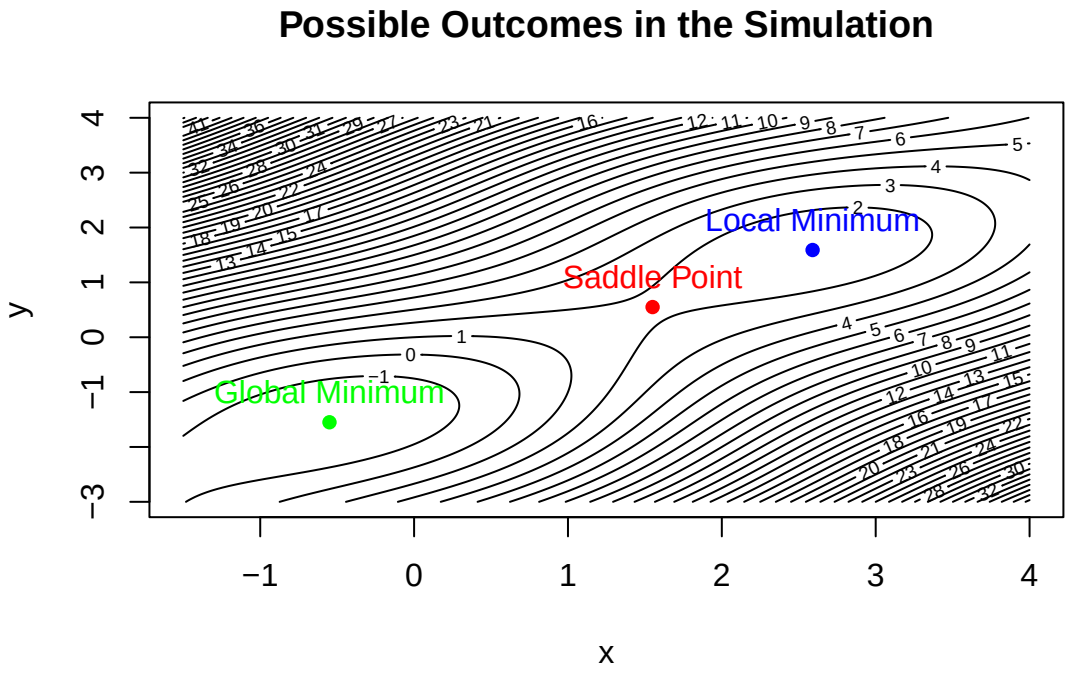
There are four possible outcomes in the simulation:

Global minimum found (green point): The eigenvalues are greater than 0, and the starting point is close to the coordinates of the global minimum.

Local minimum found (blue point): The eigenvalues are greater than 0, but the starting point is farther from the coordinates of the global minimum.

Saddle point (red point): One eigenvalue is greater than 0, while the other is less than 0.

Invalid point: The simulated minimum coordinates fall outside the defined domain.



3 Question2

3.1 Compute the ML-estimator

In this analysis, we estimate the logistic regression parameters (β_0, β_1) by maximizing the log-likelihood function. The logistic regression model is given by

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

and the corresponding log-likelihood function is

$$\ell(\beta_0, \beta_1) = \sum_{i=1}^{n} \Big[y_i \log\big(p(x_i)\big) + (1 - y_i) \log\big(1 - p(x_i)\big) \Big],$$

where y_i is the binary outcome for observation i and x_i is the corresponding predictor (dose).

We use the steepest ascent method to maximize the log-likelihood. In this method, the parameter estimates are updated in the direction of the gradient.

The algorithm converged to the following estimates:

- $\hat{\beta}_0 \approx -0.00901$
- $\hat{\beta}_1 \approx 1.26125$

The algorithm required 80 evaluations of the log-likelihood function and 28 evaluations of the gradient.

3.2 Comparison of Two Backtracking Line Search Variants

In this part, we compute the maximum likelihood estimator for the logistic regression parameters (β_0, β_1) using the steepest ascent method with a backtracking line search. We use a stopping criterion that ensures the estimates are accurate to five decimal places. The starting value for the parameters is set to $(\beta_0, \beta_1) = (-0.2, 1)$.

We try two variants of the backtracking strategy:

1. **Variant 1:** The step size is halved repeatedly (i.e., $\alpha \leftarrow \alpha/2$) until an improvement in the log-likelihood is observed.
2. **Variant 2:** The step size is reduced by multiplying it by 0.8 (i.e., $\alpha \leftarrow 0.8 \cdot \alpha$) until the log-likelihood increases.

- **Variant 1 (alpha halved):**
  - Estimated parameters:
    $\hat{\beta}_0 \approx -0.00842$ and $\hat{\beta}_1 \approx 1.26052$
  - Function evaluations: **56**
  - Gradient evaluations: **19**
- **Variant 2 (alpha reduced by 20%):**
  - Estimated parameters:
    $\hat{\beta}_0 \approx -0.00968$ and $\hat{\beta}_1 \approx 1.26221$
  - Function evaluations: **140**
  - Gradient evaluations: **28**

Comparing the two variants, Variant 1 converged with significantly fewer function and gradient evaluations than Variant 2. This suggests that, for our dataset, the strategy of halving the step size is more efficient than reducing it by 20%.

3.3 Comparison Using `optim` with BFGS and Nelder-Mead

We now use the function `optim` with both the BFGS and the Nelder-Mead algorithms to compute the ML estimator for the logistic regression parameters. The goal is to compare the results to those obtained in part (b) and to examine the number of function and gradient evaluations reported by `optim`.

BFGS Method:

- **Parameter estimates:**
  $\hat{\beta}_0 \approx -0.008963304$ and $\hat{\beta}_1 \approx 1.262923934$
- **Negative log-likelihood value:** 6.484279
- **Function evaluations:** 9
- **Gradient evaluations:** 6

Nelder-Mead Method:

- **Parameter estimates:**
  $\hat{\beta}_0 \approx -0.007853699$ and $\hat{\beta}_1 \approx 1.262287140$

- **Negative log-likelihood value:** 6.484281
- **Function evaluations:** 33

- **Gradient evaluations:** Not applicable

3.3.0.1 Precision of Results Both methods yield very similar parameter estimates and nearly identical negative log-likelihood values, indicating that the precision of the results is comparable to those obtained in part (b).

3.3.0.2 Efficiency (Evaluations) The BFGS method required significantly fewer function evaluations (9 vs. 33) than the Nelder-Mead method. This suggests that BFGS is more efficient, likely due to its use of gradient information.

3.4 ML Estimation Using `glm` and Comparison with Previous Results

We now use the built-in `glm` function in R to obtain the maximum likelihood solution for the logistic regression model.

The estimated coefficients from `glm` are:

- **Intercept:** $\hat{\beta}_0 \approx -0.00936$
- **Slope:** $\hat{\beta}_1 \approx 1.26282$

The estimates obtained from `glm` are extremely close to those from the BFGS optimization. The differences are only in the fourth or fifth decimal place, which indicates that both methods yield essentially the same maximum likelihood estimates.

4 Appendix

```
# Question 1
verify = function(x, y){
  x_codt = all(x >= -1.5 & x <= 4)
  y_codt = all(y >= -3 & y <= 4)
  return(x_codt&y_codt)
}
f_xy = function(x, y){
  return(sin(x+y) + (x-y)^2 - 1.5*x + 2.5*y + 1)
}

derv = function(order, var, x, y){
  if(order == 2){
    if(var == "xy"){
      return(-sin(x+y) - 2)
    }
    else{
      return(-sin(x+y) + 2)
    }
  }
  else if(order == 1){
```

```
    if(var == "x"){
      return(cos(x+y) + 2*(x-y) - 1.5)
    }
    else if(var == "y"){
      return(cos(x+y) - 2*(x-y) + 2.5)
    }
  }
}

gradient = function(x, y){
  df_dx = derv(order = 1, var = "x", x, y)
  df_dy = derv(order = 1, var = "y", x, y)

  gradient = matrix(c(df_dx,df_dy), ncol=1)
  return(gradient)
}

hessian_M = function(x, y){
  df2 = derv(order = 2, var = "x", x, y)
  df2_xy = derv(order = 2, var = "xy", x, y)
  hessian_m = matrix(c(df2, df2_xy, df2_xy, df2), ncol=2)
  return(hessian_m)
}
# Implement Newton Method
newton = function(x0, y0){
  cad = list()
  for(i in 1:length(x0)){
    xt = matrix(c(x0[i], y0[i]), ncol=1)
    while(TRUE){
      dev_mul = solve(hessian_M(xt[1], xt[2])) %*% gradient(xt[1], xt[2])
      xt1 = xt - dev_mul
      if(all(abs(xt - xt1) < 0.001)){
        cad = append(cad, list(xt1))
        break
      }
      xt = xt1
    }
  }
  return(cad)
}

# Plot of function
x_vec = seq(from = -1.5, to = 4, length.out = 500)
y_vec = seq(from = -3, to = 4, length.out = 500)
if(verify(x_vec, y_vec)){
  z = outer(x_vec, y_vec, f_xy)
}

contour(x_vec, y_vec, z, main = "Contour Plot of f(x, y) = sin(x + y) + (x - y)^2 - 1.5x + 2.5y + 1", x

# Compute gradient and Hessian matrix
library(pander)
x = 1.5
```

```r
y = 2
cat("Assume that x =",x, "y =",y,"\n")
cat("Gradient:\n")
gra = gradient(x, y)
pander(gra)
hm = hessian_M(x, y)
cat("Hessian Matrix\n")
pander(hm)

# Test on n=5 data
n = 5
x_random = runif(n, min = -1.5, max = 4)
y_random = runif(n, min = -3, max = 4)
start_points = vector("list", n)
candi = vector("list", n)
fxy = rep(0, n)
opti_test = newton(x_random, y_random)

for (i in 1:n) {
  start_points[[i]] = round(c(x_random[i], y_random[i]), 2)
  candi_point = as.vector(opti_test[[i]])
  cadi_x = candi_point[1]
  cadi_y = candi_point[2]
  if(verify(cadi_x, cadi_y)){
    candi[[i]] = round(candi_point, 2)
    fxy[i] = round(f_xy(cadi_x, cadi_y) ,2)
  }
  else{
    candi[[i]] = "invalid"
    fxy[i] = "invalid"
  }

}

outcome = data.frame(
  start_points = format_points(start_points),
  eigen_values = format_points(eigenCompute(start_points)),
  candidates = format_points(candi),
  function_value = fxy
)

contour(x_vec, y_vec, z, main = "Possible Outcomes in the Simulation", xlab="x", ylab="y", nlevels = 50)

points(1.55, 0.55,  col = "red", pch = 16, cex = 1.0)
text(1.55, 0.55, "Saddle Point", pos = 3, col = "red")
points(2.59, 1.59,  col = "blue", pch = 16, cex = 1.0)
text(2.59, 1.59, "Local Minimum", pos = 3, col = "blue")
points(-0.55, -1.55,  col = "green", pch = 16, cex = 1.0)
text(-0.55, -1.55, "Global Minimum", pos = 3, col = "green")



# Question 2
```

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

# Initialize counters
fn_counter <<- 0
gr_counter <<- 0

logistic_probability <- function(params, pred) {
  1 / (1 + exp(- (params[1] + params[2] * pred)))
}

log_likelihood <- function(params) {
  fn_counter <<- fn_counter + 1
  sum(y * log(logistic_probability(params, x)) +
        (1 - y) * log(1 - logistic_probability(params, x)))
}

grad_calc <- function(params) {
  gr_counter <<- gr_counter + 1
  g0 <- sum(y - logistic_probability(params, x))
  g1 <- sum((y - logistic_probability(params, x)) * x)
  c(g0, g1)
}

ascend_opt <- function(init_params, tol = 1e-6, init_step = 1) {
  params <- init_params
  diff_val <- 999
  while (diff_val > tol) {
    grad_val <- grad_calc(params)
    step_size <- init_step
    new_params <- params + step_size * grad_val
    while (log_likelihood(new_params) < log_likelihood(params)) {
      step_size <- step_size / 2
      new_params <- params + step_size * grad_val
    }
    diff_val <- sum((new_params - params)^2)
    params <- new_params
  }
  params
}

init_params <- c(0, 0)
par_est <- ascend_opt(init_params)
print(par_est)
cat("Estimated coefficients:\n")
print(par_est)
cat("\nNumber of log-likelihood evaluations:", fn_counter, "\n")
cat("Number of gradient evaluations:", gr_counter, "\n\n")

# b

fn_counter <<- 0
gr_counter <<- 0
```

```
ascend_opt_v1 <- function(init_params, tol = 1e-6, init_step = 1) {
  params <- init_params
  diff_val <- 999
  while (diff_val > tol) {
    grad_val <- grad_calc(params)
    step_size <- init_step
    new_params <- params + step_size * grad_val
    while (log_likelihood(new_params) < log_likelihood(params)) {
      step_size <- step_size / 2
      new_params <- params + step_size * grad_val
    }
    diff_val <- sum((new_params - params)^2)
    params <- new_params
  }
  params
}

init_params <- c(-0.2, 1)
par_est_v1 <- ascend_opt_v1(init_params)
fn_counter_v1 <- fn_counter
gr_counter_v1 <- gr_counter

fn_counter <<- 0
gr_counter <<- 0

ascend_opt_v2 <- function(init_params, tol = 1e-6, init_step = 1) {
  params <- init_params
  diff_val <- 999
  while (diff_val > tol) {
    grad_val <- grad_calc(params)
    step_size <- init_step
    new_params <- params + step_size * grad_val
    while (log_likelihood(new_params) < log_likelihood(params)) {
      step_size <- step_size * 0.8
      new_params <- params + step_size * grad_val
    }
    diff_val <- sum((new_params - params)^2)
    params <- new_params
  }
  params
}

par_est_v2 <- ascend_opt_v2(init_params)
fn_counter_v2 <- fn_counter
gr_counter_v2 <- gr_counter

cat("Variant 1 (alpha halved):\n")
print(par_est_v1)
cat("Function evaluations:", fn_counter_v1, "\n")
cat("Gradient evaluations:", gr_counter_v1, "\n\n")

cat("Variant 2 (alpha reduced by 20%):\n")
print(par_est_v2)
```

```r
cat("Function evaluations:", fn_counter_v2, "\n")
cat("Gradient evaluations:", gr_counter_v2, "\n")

# c

logistic_probability <- function(params, pred) {
  1 / (1 + exp(- (params[1] + params[2] * pred)))
}

log_likelihood <- function(params) {
  sum(y * log(logistic_probability(params, x)) +
        (1 - y) * log(1 - logistic_probability(params, x)))
}

grad_calc <- function(params) {
  g0 <- sum(y - logistic_probability(params, x))
  g1 <- sum((y - logistic_probability(params, x)) * x)
  c(g0, g1)
}

init_params <- c(-0.2, 1)

opt_bfgs <- optim(
  par = init_params,
  fn = function(params) -log_likelihood(params),
  gr = function(params) -grad_calc(params),
  method = "BFGS",
  control = list(reltol = 1e-6),
  hessian = TRUE
)

cat("Results from optim with BFGS:\n")
print(opt_bfgs)
cat("\n")

opt_nm <- optim(
  par = init_params,
  fn = function(params) -log_likelihood(params),
  method = "Nelder-Mead",
  control = list(reltol = 1e-6)
)

cat("Results from optim with Nelder-Mead:\n")
print(opt_nm)



cat("BFGS estimates:", opt_bfgs$par, "\n")
cat("Nelder-Mead estimates:", opt_nm$par, "\n")

# d

df_data <- data.frame(x = x, y = y)
```

```r
glm_fit <- glm(y ~ x, family = binomial, data = df_data)

summary(glm_fit)

glm_coef <- coef(glm_fit)
cat("Coefficients from glm:\n")
print(glm_coef)


cat("BFGS estimates:", opt_bfgs$par, "\n")
cat("glm estimates:  ", glm_coef, "\n")
```