Lab2

Cui Qingxuan, Nisal Amashan

2025-02-03

Contents

1 Collaborations

Cui Qingxuan: Responsible for the question 1.

Nisal Amashan: Responsible for the question 2.

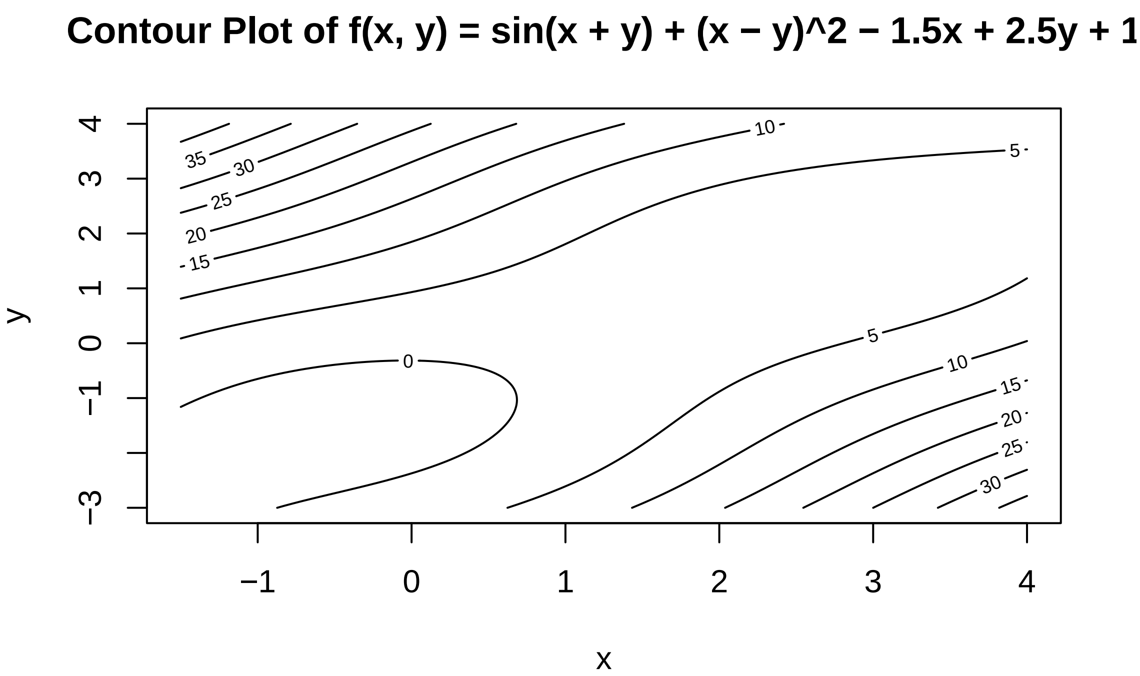
2 Question 1

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
(2.87, 2.69)	(2.59, 1.59)	(4.00, 1.72)	1.23
(-0.71, -0.24)	(-0.55, -1.55)	(4.00, 1.73)	-1.91
(2.92, 1.19)	(2.59, 1.59)	(4.00, 1.72)	1.23
(2.64, -1.66)	(1.55, 0.55)	(4.00, -1.73)	1.91
(-0.13, 1.16)	(1.55, 0.55)	(4.00, -1.73)	1.91
(2.18, -1.72)	(2.59, 1.59)	(4.00, 1.72)	1.23
(0.25, 0.30)	invalid	invalid	invalid
(-0.70, -2.72)	invalid	invalid	invalid
(1.90, 2.25)	(2.59, 1.59)	(4.00, 1.72)	1.23

2.5 Summary

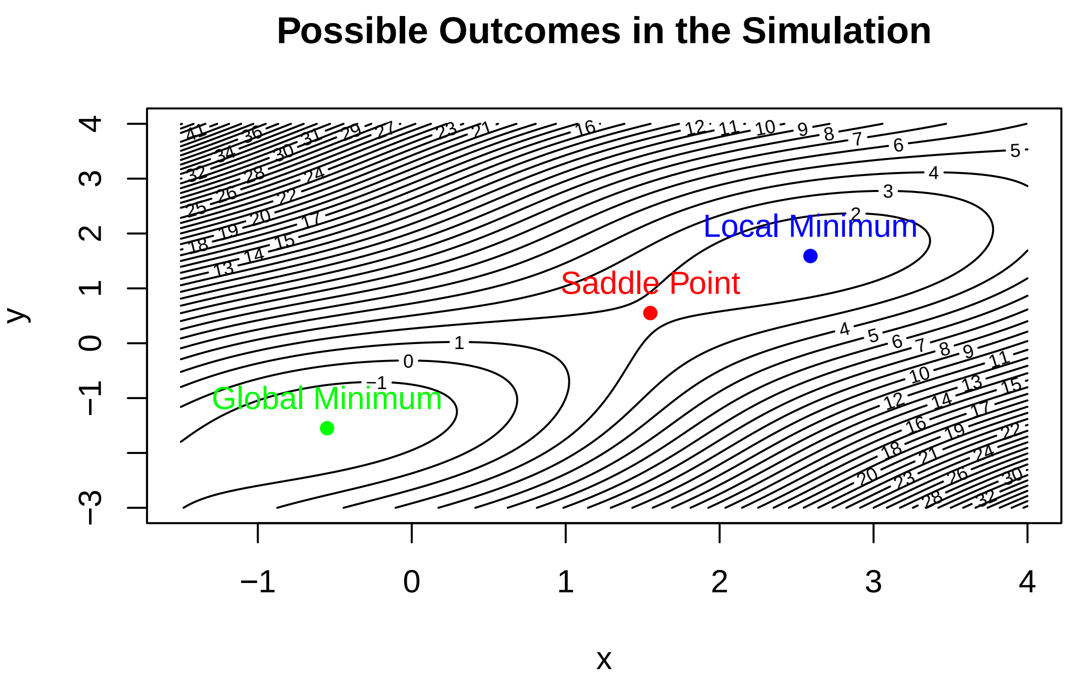
There are four possible outcomes in the simulation:

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

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)
```

```r
    }
  }
}

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", xl

# Compute gradient and Hessian matrix
library(pander)
x = 1.5
y = 2
cat("Assume that x =",x, "y =",y,"\n")
cat("Gradient:\n")
gra = gradient(x, y)
pander(gra)
```

```r
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")
```