LAB 2

Greeshma Jeev Koothuparambil(greko370), Sangeeth Sankunny Menon(sansa237)

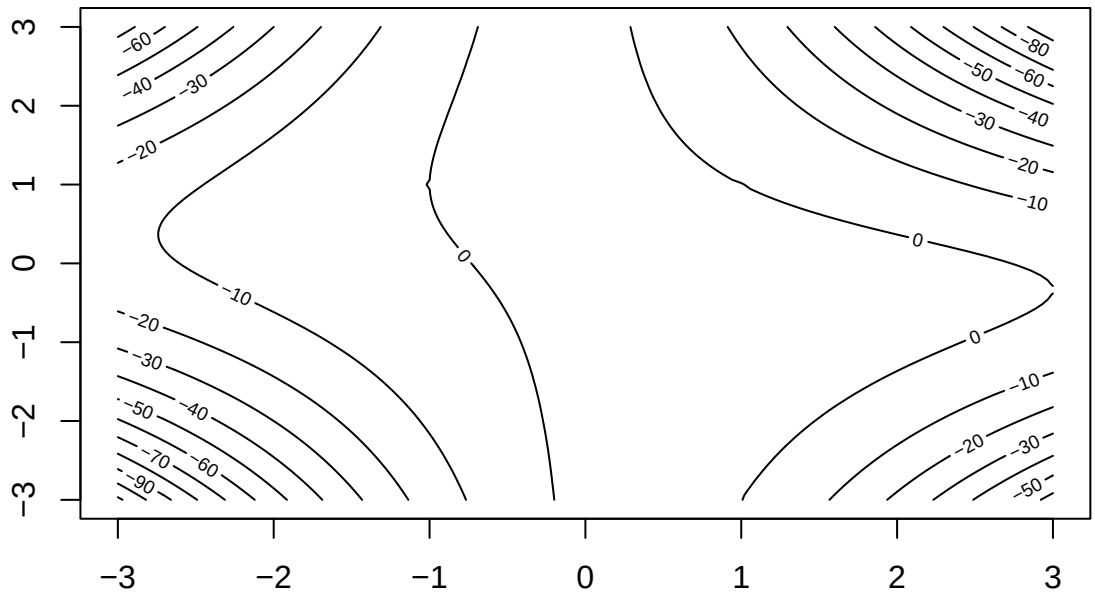
2023-11-13

Question 1: Optimisation of a two-dimensional function

Consider the function $g(x,y) = -x^2 - x^2y^2 - 2xy + 2x + 2$ *It is desired to determine the point* $(x,y), x,y \in [-3.3]$ *, where the function is maximized. a. Derive the gradient and the Hessian matrix in dependence of* x,y*. Produce a contour plot of the function g*

```
#1.a
# g(x,y)
gfunc <- function(x,y){
   valmat<- (-x^2)-(x^2*y^2)-(2*x*y)+(2*x)+2
   return(valmat)
}

#gradient function
gderfunc <- function(x,y){
  valueX <- (-x*2)-(x*2*y^2)-(2*y)+2
  valueY <- (-x*2)-(x^2*y*2)
  dermax <-matrix(c(valueX,valueY),nrow = 2)
  return(dermax)
}

#hessian matrix function
gdubderfunc <- function(x,y){
  x2 <- -2-(2*y^2)
  y2 <- -2*x^2
  xy <- (-4*x*y)- 2
  yx <-(-4*x*y)- 2
  hmat <- matrix(c(x2,xy,yx,y2),ncol = 2, byrow = T)
}


#contour plot
range_x<-seq(-3,3,length.out=100)
range_y<-seq(-3,3,length.out=100)

gmat <- matrix(rep(NA, (length(range_x)*length(range_y))), nrow=length(range_x))
for (i in 1:length(range_x)) {
  for (j in 1:length(range_y)) {

    gmat[i,j] <- gfunc(range_x[i], range_y[j])
  }
```

```
}

contour(range_x, range_y, gmat)
```



b. Write an own algorithm based on the Newton method in order to find a local maximum of g.

```
#1.b
maxfun <- function(x,y){
  xt <- matrix(c(x,y), nrow = 2)
  xpt <- matrix(c(4,5), nrow = 2)
  #mat0 <- matrix(c(0,0), nrow = 2)
  while (t(xt-xpt)%*%(xt-xpt)> 0.0001) {
    tempxt <- xt
    der1 <- gderfunc(xt[1],xt[2])
    der2 <- gdubderfunc(xt[1],xt[2])
    invder2 <- solve(der2)
    xt <- xt-(invder2 %*% der1)
    #xt <- round(xt, 6)
    xpt <- tempxt
    print(xt)
  }

  return(xt)
}
```

c. Use different starting values: use the three points $(x, y) = (2, 0), (-1, -2), (0, 1)$ and a fourth point of

your choice. Describe what happens when you run your algorithm for each of those starting values. If your algorithm converges to points (x, y), compute the gradient and the Hessian matrix at these points and decide about local maximum, minimum, saddle point, or neither of it. Did you find a global maximum for $x, y \in [-3, 3]$?

```
#1.c
maxima1 <- maxfun(2,0)
```

```
##             [,1]
## [1,]  1.3333333
## [2,] -0.3333333
##             [,1]
## [1,]  1.2410901
## [2,] -0.7442348
##             [,1]
## [1,]  1.0237697
## [2,] -0.9252917
##             [,1]
## [1,]  1.0044994
## [2,] -0.9932296
##             [,1]
## [1,]  1.0000256
## [2,] -0.9999341
```

```
grad1 <- gderfunc(2,0)
hess1 <- gdubderfunc(2,0)
eigenvalues1 <- eigen(hess1)$values

maxima2 <- maxfun(-1,-2)
```

```
##       [,1]
## [1,] -0.65
## [2,] -0.75
##             [,1]
## [1,] -0.4212980
## [2,]  0.4693814
##             [,1]
## [1,] -0.2944317
## [2,]  1.9415362
##             [,1]
## [1,] -0.1463403
## [2,]  3.6411701
##             [,1]
## [1,] -0.1708772
## [2,]  6.7581129
##             [,1]
## [1,] -0.2562939
## [2,]  2.0211055
##             [,1]
## [1,] -0.1872322
## [2,]  3.9396145
##             [,1]
## [1,] -0.1059095
```

```
## [2,]  6.4434362
##            [,1]
## [1,] -0.1008975
## [2,]  9.6050448
##             [,1]
## [1,]  0.05997255
## [2,] 24.73735499
##             [,1]
## [1,]  0.03238945
## [2,] 13.74953864
##            [,1]
## [1,] 0.01204251
## [2,] 5.79567256
##             [,1]
## [1,] 0.001278017
## [2,] 1.548633061
##              [,1]
## [1,] 3.408517e-06
## [2,] 1.002152e+00
##              [,1]
## [1,] 1.166791e-11
## [2,] 1.000000e+00
```

```
grad2 <- gderfunc(-1,-2)
hess2 <- gdubderfunc(-1,-2)
eigenvalues2 <- eigen(hess2)$values

maxima3 <- maxfun(0,1)
```

```
##      [,1]
## [1,]    0
## [2,]    1
```

```
grad3 <- gderfunc(0,1)
hess3 <- gdubderfunc(0,1)
eigenvalues3 <- eigen(hess3)$values

maxima4 <- maxfun(1,-2)
```

```
##       [,1]
## [1,]  0.75
## [2,] -1.75
##            [,1]
## [1,]  0.1483516
## [2,] -3.0714286
##            [,1]
## [1,]  0.4311002
## [2,] -7.8802676
##            [,1]
## [1,]  0.3128723
## [2,] -6.0057683
##            [,1]
## [1,]  0.2351958
```

```
## [2,] -5.3847688
##            [,1]
## [1,]  0.1497591
## [2,] -6.6194051
##            [,1]
## [1,]  0.6319525
## [2,] 14.4490280
##           [,1]
## [1,] 0.4056384
## [2,] 9.3332263
##           [,1]
## [1,] 0.2464461
## [2,] 5.8278641
##           [,1]
## [1,] 0.1292601
## [2,] 3.4141024
##            [,1]
## [1,] 0.04432517
## [2,] 1.83380722
##             [,1]
## [1,] 0.004338699
## [2,] 1.100313295
##              [,1]
## [1,] 2.237044e-05
## [2,] 1.000900e+00
##              [,1]
## [1,] 5.013144e-10
## [2,] 1.000000e+00
```

```r
grad4 <- gderfunc(1,-2)
hess4 <- gdubderfunc(1,-2)
eigenvalues4 <- eigen(hess4)$values
```

Point (2,0):

Converging point:(1.0000256, -0.9999341)
Gradient : (-2, -4)
Hessian Matrix :

$$\begin{bmatrix} -2 & -2 \\ -2 & -8 \end{bmatrix}$$

Eigen Values : -1.3944487, -8.6055513
Since both eigen values are negative, hessian matrix is definite negative. Therefore, Point (1,-1) is a local maxima.

Point (-1,-2):

Converging point:$(1.1667906 \times 10^{-11}, 1)$
Gradient : (16, 6)
Hessian Matrix :

$$\begin{bmatrix} -10 & -10 \\ -10 & -2 \end{bmatrix}$$

Eigen Values : 4.7703296, -16.7703296
Since eigen values exhibit different signs, hessian matrix cannot be declared definite negative or positive. Therefore, Point (0,1) is a saddle point.

Point (0,1):

Converging point:(0, 1)
Gradient : (0, 0)
Hessian Matrix :

$$\begin{bmatrix} -4 & -2 \\ -2 & 0 \end{bmatrix}$$

Eigen Values : 0.8284271, -4.8284271
Since eigen values exhibit different signs, hessian matrix cannot be declared definite negative or positive. Therefore, Point (0,1) is a saddle point.

Point (1,-2):

Converging point:($5.0131443 \times 10^{-10}$, 1)
Gradient : (-4, 2)
Hessian Matrix :

$$\begin{bmatrix} -10 & 6 \\ 6 & -2 \end{bmatrix}$$

Eigen Values : 1.2111026, -13.2111026
Since eigen values exhibit different signs, hessian matrix cannot be declared definite negative or positive. Therefore, Point (0,1) is a saddle point.

Since only one point converges to a local maxima, we could not find any global maxima in [-3.3]

d. What would be the advantages and disadvantages when you would run a steepest ascent algorithm instead of the Newton algorithm?

Advantages:

1.The Steepest Ascent algorithm is relatively simple to implement.
2.It doesn't require the computation of the Hessian matrix, which can be computationally expensive and problematic for large datasets.
3.It avoids Newton's uncertainty on guarantied increase in g(x).

Disadvantages:

1.It might converge slowly according to the number of iterations.
2.It's prone to getting stuck in local optima and might not find the global maximum.
3.No information on the nature of curvature.

Question 2

Three doses (0.1, 0.3, and 0.9 g) of a drug and placebo (0 g) are tested in a study. A dose-dependent event is recorded afterwards. The data of $n = 10$ subjects is shown in Table 1; x_i is the dose in gram; $y_i = 1$ if the event occurred, $yi = 0$ otherwise.

x_grams	0	0	0	0.1	0.1	0.3	0.3	0.9	0.9	0.9
y	0	0	1	0.0	1.0	1.0	1.0	0.0	1.0	1.0

You should fit a simple logistic regression

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

to the data, i.e. estimate β_0 and β_1. One can show that the log likelihood is

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

where $b = (\beta_0, \beta_1)^T$ and the gradient is

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

a. Write a function for an ML-estimator for (β_0, β_1) using the steepest ascent method with a step-size reducing line search (back-tracking). For this, you can use and modify the code for the steepest ascent example from the lecture. The function should count the number of function and gradient evaluations.

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

g <- function(b)
{
  b0 <- b[1]
  b1 <- b[2]
  loglikeli <- 0
  for (i in 1:10) {
    expo <- 1/(1+ exp(-b0-(b1*df[i,1])))
    loglikeli <- loglikeli + ((df[i,2]*log(expo))+((1-df[i,2])*log(1-expo)))
  }
  return(loglikeli)
}
#gradient function
gradient <- function(b){
  b0 <- b[1]
  b1 <- b[2]
  grad <- 0
  for (i in 1:10) {
    expo <- 1/(1+ exp(-b0-(b1*df[i,1])))
    gradexp <- (df[i,2]-expo)* matrix(c(1,df[i,1]), nrow = 2)
    grad <- grad + gradexp
```

```
  }

  return(grad)

}


#Steepest ascent function:
steepestasc <- function(x0, eps=1e-8, alpha0=1)
{
  xt   <- x0
  conv <- 999
  gra <- 0
  gfun <- 0
  while(conv>eps)
  {
    alpha <- alpha0
    xt1   <- xt
    xt    <- xt1 + alpha*gradient(xt)
    gra <- gra+1
    gfun <- gfun+1
    while (g(xt)<g(xt1))
    {
      alpha <- alpha/2
      xt    <- xt1 + alpha*gradient(xt)
      gra <- gra+1
    }
    conv <- sum((xt-xt1)*(xt-xt1))
  }
  xt
  retlist <- list(optimum = xt, giter = gfun, graiter = gra)
  return(retlist)
}
```

b. Compute the ML-estimator with the function from a. for the data (x_i, y_i) above. Use a stopping criterion such that you can trust five digits of both parameter estimates for β_0 and $\beta 1$. Use the starting value $(\beta_0, \beta_1) = (-0.2, 1)$. The exact way to use backtracking can be varied. Try two variants and compare number of function and gradient evaluation done until convergence.

```
#2.b
s1 <- steepestasc(c(-0.2, 1))
s2 <- steepestasc(c(2, 0))
s3 <- steepestasc(c(-3, 2))
```

Comparison between different starting points on Steepest Ascent:

point	convergence_value	function_count	gradient_count
(-0.2,1)	(-0.009263641,1.262663241)	79	223
(2,0)	(-0.009270654,1.262684864)	106	304
(-3,2)	(-0.009448644,1.262963710)	92	234

c. Use now the function optim with both the BFGS and the Nelder-Mead algorithm. Do you obtain the same

results compared with b.? Is there any difference in the precision of the result? Compare the number of function and gradient evaluations which are given in the standard output of optim.

```
g1 <- function(b)
{
  b0 <- b[1]
  b1 <- b[2]
  loglikeli <- 0
  for (i in 1:10) {
    expo <- 1/(1+ exp(-b0-(b1*df[i,1])))
    loglikeli <- loglikeli + ((df[i,2]*log(expo))+((1-df[i,2])*log(1-expo)))
  }
  return(-loglikeli)
}

o1BFGS <- optim(c(-0.2, 1), g1, method = "BFGS")
o2BFGS <- optim(c(2, 0), g1, method = "BFGS")
o3BFGS <- optim(c(-3, 2), g1, method = "BFGS")

o1NM <- optim(c(-0.2, 1), g1, method = "Nelder-Mead")
o2NM <- optim(c(2, 0), g1, method = "Nelder-Mead")
o3NM <- optim(c(-3, 2), g1, method = "Nelder-Mead")
```

Comparison of convergence points delivered by Steepest Ascent, bfgs,Nelder Mead is as follows in the table

point	SteepestAscent	BFGS	NelderMead
(-0.2,1)	(-0.009263641,1.262663241)	(-0.009356112,1.262812883)	(-0.009423433,1.262738266)
(2,0)	(-0.009270654,1.262684864)	(-0.009348594,1.262953554)	(-0.008783172,1.261982195)
(-3,2)	(-0.009448644,1.262963710)	(-0.009359106,1.262821223)	(-0.009574578,1.263235817)

Comparison on Function count delivered by Steepest Ascent, bfgs,Nelder Mead is as follows in the table

point	SteepestAscent	BFGS	NelderMead
(-0.2,1)	79	12	47
(2,0)	106	15	57
(-3,2)	92	16	61

Comparison on Gradient count delivered by Steepest Ascent, bfgs,Nelder Mead is as follows in the table

point	SteepestAscent	BFGS	NelderMead
(-0.2,1)	223	8	NA
(2,0)	304	11	NA
(-3,2)	234	10	NA

d. Use the function glm in R to obtain an ML-solution and compare it with your results before.

```
model <- glm(y~x, data = df, family = "binomial")
summary(model)
```

```
##
## Call:
## glm(formula = y ~ x, family = "binomial", data = df)
##
## 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
```

All the algorithms converges at (-0.009,1.262).

Appendix

QUESTION 1

```
#1.a
# g(x,y)
gfunc <- function(x,y){
   valmat<- (-x^2)-(x^2*y^2)-(2*x*y)+(2*x)+2
   return(valmat)
}

#gradient function
gderfunc <- function(x,y){
  valueX <- (-x*2)-(x*2*y^2)-(2*y)+2
  valueY <- (-x*2)-(x^2*y*2)
  dermax <-matrix(c(valueX,valueY),nrow = 2)
  return(dermax)
}

#hessian matrix function
gdubderfunc <- function(x,y){
  x2 <- -2-(2*y^2)
  y2 <- -2*x^2
  xy <- (-4*x*y)- 2
  yx <-(-4*x*y)- 2
  hmat <- matrix(c(x2,xy,yx,y2),ncol = 2, byrow = T)
}


#contour plot
```

```r
range_x<-seq(-3,3,length.out=100)
range_y<-seq(-3,3,length.out=100)

gmat <- matrix(rep(NA, (length(range_x)*length(range_y))), nrow=length(range_x))
for (i in 1:length(range_x)) {
  for (j in 1:length(range_y)) {

    gmat[i,j] <- gfunc(range_x[i], range_y[j])
  }
}

contour(range_x, range_y, gmat)

#1.b

maxfun <- function(x,y){
  xt <- matrix(c(x,y), nrow = 2)
  xpt <- matrix(c(4,5), nrow = 2)
  #mat0 <- matrix(c(0,0), nrow = 2)
  while (t(xt-xpt)%*%(xt-xpt)> 0.0001) {
    tempxt <- xt
    der1 <- gderfunc(xt[1],xt[2])
    der2 <- gdubderfunc(xt[1],xt[2])
    invder2 <- solve(der2)
    xt <- xt-(invder2 %*% der1)
    #xt <- round(xt, 6)
    xpt <- tempxt
    print(xt)
  }

  return(xt)
}

#1.c
maxima1 <- maxfun(2,0)
grad1 <- gderfunc(2,0)
hess1 <- gdubderfunc(2,0)
eigenvalues1 <- eigen(hess1)$values

maxima2 <- maxfun(-1,-2)
grad2 <- gderfunc(-1,-2)
hess2 <- gdubderfunc(-1,-2)
eigenvalues2 <- eigen(hess2)$values

maxima3 <- maxfun(0,1)
grad3 <- gderfunc(0,1)
hess3 <- gdubderfunc(0,1)
eigenvalues3 <- eigen(hess3)$values

maxima4 <- maxfun(1,-2)
grad4 <- gderfunc(1,-2)
hess4 <- gdubderfunc(1,-2)
eigenvalues4 <- eigen(hess4)$values
```

QUESTION 2

```
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)
df <-  data.frame(x=x,y=y)
row.names(df) <- c(1:10)

g <- function(b)
{
  b0 <- b[1]
  b1 <- b[2]
  loglikeli <- 0
  for (i in 1:10) {
    expo <- 1/(1+ exp(-b0-(b1*df[i,1])))
    loglikeli <- loglikeli + ((df[i,2]*log(expo))+((1-df[i,2])*log(1-expo)))
  }
  return(loglikeli)
}
#gradient function
gradient <- function(b){
  b0 <- b[1]
  b1 <- b[2]
  grad <- 0
  for (i in 1:10) {
    expo <- 1/(1+ exp(-b0-(b1*df[i,1])))
    gradexp <- (df[i,2]-expo)* matrix(c(1,df[i,1]), nrow = 2)
    grad <- grad + gradexp
  }

  return(grad)

}

#Steepest ascent function:
steepestasc <- function(x0, eps=1e-8, alpha0=1)
{
  xt   <- x0
  conv <- 999
  gra <- 0
  gfun <- 0
  while(conv>eps)
  {
    alpha <- alpha0
    xt1   <- xt
    xt    <- xt1 + alpha*gradient(xt)
    gra <- gra+1
    gfun <- gfun+1
    while (g(xt)<g(xt1))
    {
      alpha <- alpha/2
      xt    <- xt1 + alpha*gradient(xt)
      gra <- gra+1
    }
```

```
    conv <- sum((xt-xt1)*(xt-xt1))
  }
  xt
  retlist <- list(optimum = xt, giter = gfun, graiter = gra)
  return(retlist)
}

#2.b
s1 <- steepestasc(c(-0.2, 1))
s2 <- steepestasc(c(2, 0))
s3 <- steepestasc(c(-3, 2))

point <- c("(-0.2,1)","(2,0)","(-3,2)")
optimum_value <- c("(-0.009263641,1.262663241)",
                   "(-0.009270654,1.262684864)",
                   "(-0.009448644,1.262963710)")
function_count <- c(s1$giter, s2$giter, s3$giter)
gradient_count <- c(s1$graiter, s2$graiter, s3$graiter)

summarySA <- data.frame(point, optimum_value,function_count, gradient_count)
#2.c
g1 <- function(b)
{
  b0 <- b[1]
  b1 <- b[2]
  loglikeli <- 0
  for (i in 1:10) {
    expo <- 1/(1+ exp(-b0-(b1*df[i,1])))
    loglikeli <- loglikeli + ((df[i,2]*log(expo))+((1-df[i,2])*log(1-expo)))
  }
  return(-loglikeli)
}

o1BFGS <- optim(c(-0.2, 1), g1, method = "BFGS")
o2BFGS <- optim(c(2, 0), g1, method = "BFGS")
o3BFGS <- optim(c(-3, 2), g1, method = "BFGS")

o1NM <- optim(c(-0.2, 1), g1, method = "Nelder-Mead")
o2NM <- optim(c(2, 0), g1, method = "Nelder-Mead")
o3NM <- optim(c(-3, 2), g1, method = "Nelder-Mead")

point <- c("(-0.2,1)","(2,0)","(-3,2)")
SteepestAscent <- c("(-0.009263641,1.262663241)",
                    "(-0.009270654,1.262684864)",
                    "(-0.009448644,1.262963710)")
BFGS <- c("(-0.009356112,1.262812883)",
          "( -0.009348594,1.262953554)",
          "(-0.009359106,1.262821223)")
NelderMead <- c("(-0.009423433,1.262738266)",
                "( -0.008783172,1.261982195)",
                "(-0.009574578,1.263235817)")
convergence <- data.frame(point,SteepestAscent, BFGS, NelderMead )
```

```r
SteepestAscent <- c(s1$giter, s2$giter, s3$giter)
BFGS <- c(12,15,16)
NelderMead <- c(47,57,61)
FunctionCount <- data.frame(point,SteepestAscent, BFGS, NelderMead)

SteepestAscent <- c(s1$graiter, s2$graiter, s3$graiter)
BFGS <- c(8,11,10)
NelderMead <- c(NA,NA,NA)
GradientCount <- data.frame(point,SteepestAscent, BFGS, NelderMead)


#2.d
model <- glm(y~x, data = df, family = "binomial")
summary(model)
```