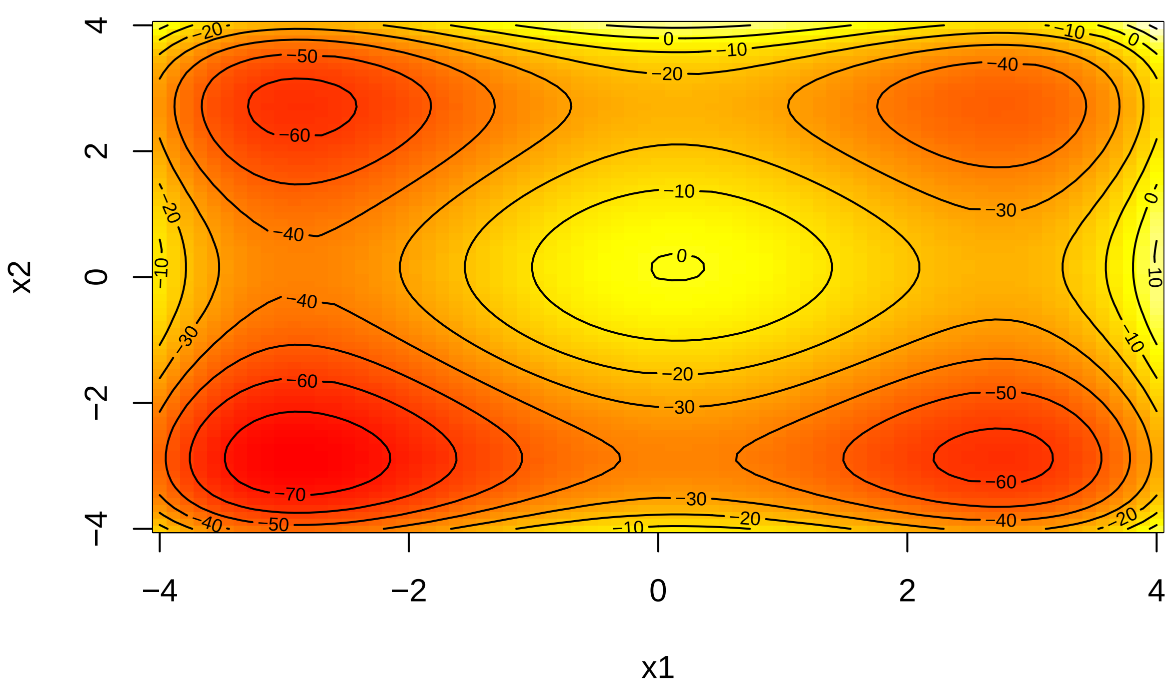
A2

Mushi Wang 20732874

2019/10/6

1 (a).

```
xdata = as.matrix(read.csv(file="fake_objective_fn_data.csv", header=FALSE))
dimnames(xdata)[[2]] = NULL

f <- function(x1, x2) {
  temp = 0
  for(i in 1 : nrow(xdata)) {
    temp = xdata[i,1] * x1^4 + xdata[i,2] * x1^2 + xdata[i,3] * x1 +
      xdata[i,4] * x2^4 + xdata[i,5] * x2^2 + xdata[i,6] * x2 + temp
  }
  temp = temp / nrow(xdata)
  return(temp)
}
x1 = x2 = seq(-4,4,length=75)
z = outer(x1, x2, FUN="f")

image(x1,x2,z,col = heat.colors(100))
contour(x1,x2,z,add=T)
```



There are four local mins, located around (-3, -3), (-3, 3), (3, -3) and (3 ,3) with value around -70, -60, -60 and -40 respectively. The gobal min is located around (-3 ,-3) with value around -70.

(b)

```
create.rho.fn <- function(pop) {
  n = nrow(pop)
  function(theta) {
    alpha <- theta[1]
    beta <- theta[2]
    sum(pop[,1] * alpha^4 + pop[,2] * alpha^2 + pop[,3] * alpha +
          pop[,4] * beta^4 + pop[,5] * beta^2 + pop[,6] * beta) / n
  }
}
rho <- create.rho.fn(xdata)
```

(c)

```
createLeastSquaresGradient <- function(pop) {
  n = nrow(pop)
  function(theta) {
    alpha <- theta[1]
    beta <- theta[2]
    1 / n * c(sum(pop[,1] * 4 * alpha^3 + pop[,2] * 2 * alpha + pop[,3]),
              sum(pop[,4] * 4 * beta^3 + pop[,5] * 2 * beta + pop[,6]))
  }
}
gradient <- createLeastSquaresGradient(xdata)
```

(d)

```
gradientDescent <- function(theta = 0,
      rhoFn, gradientFn, lineSearchFn, testConvergenceFn,
      maxIterations = 100,
      tolerance = 1E-6, relative = FALSE,
      lambdaStepsize = 0.01, lambdaMax = 0.5 ) {

  converged <- FALSE
  i <- 0

  while (!converged & i <= maxIterations) {
    g <- gradientFn(theta) ## gradient
    glength <-  sqrt(sum(g^2)) ## gradient direction
    if (glength > 0) g <- g /glength

    lambda <- lineSearchFn(theta, rhoFn, g,
                lambdaStepsize = lambdaStepsize, lambdaMax = lambdaMax)

    thetaNew <- theta - lambda * g
    converged <- testConvergenceFn(thetaNew, theta,
                                   tolerance = tolerance,
                                   relative = relative)
    theta <- thetaNew
    i <- i + 1
  }
```

```r
  ## Return last value and whether converged or not
  list(theta = theta, converged = converged, iteration = i, fnValue = rhoFn(theta)
       )
}

### line searching could be done as a simple grid search
gridLineSearch <- function(theta, rhoFn, g,
                           lambdaStepsize = 0.01,
                           lambdaMax = 1) {
  ## grid of lambda values to search
  lambdas <- seq(from = 0, by = lambdaStepsize,  to = lambdaMax)

  ## line search
  rhoVals <- Map(function(lambda) {rhoFn(theta - lambda * g)}, lambdas)
  ## Return the lambda that gave the minimum
  lambdas[which.min(rhoVals)]
}

### Where testCovergence might be (relative or absolute)
testConvergence <- function(thetaNew, thetaOld, tolerance = 1E-10, relative=FALSE) {
   sum(abs(thetaNew - thetaOld)) < if (relative) tolerance * sum(abs(thetaOld)) else tolerance
}

paste('alpha = 0, beta = 0')
```

```
## [1] "alpha = 0, beta = 0"
```

```r
result1 <- gradientDescent(theta = c(0,0),
                           rhoFn = rho, gradientFn = gradient,
                           lineSearchFn = gridLineSearch,
                           testConvergenceFn = testConvergence)
Map(function(x){if (is.numeric(x)) round(x,3) else x}, result1)
```

```
## $theta
## [1] -2.891 -2.879
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 10
##
## $fnValue
## [1] -77.283
```

```r
paste('alpha = 1, beta = 1')
```

```
## [1] "alpha = 1, beta = 1"
```

```r
result2 <- gradientDescent(theta = c(1,1),
                           rhoFn = rho, gradientFn = gradient,
                           lineSearchFn = gridLineSearch,
                           testConvergenceFn = testConvergence)

Map(function(x){if (is.numeric(x)) round(x,3) else x}, result2)
```

```
## $theta
## [1] 2.732 2.719
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 6
##
## $fnValue
## [1] -49.282
```

```r
paste('alpha = 0, beta = 3')
```

```
## [1] "alpha = 0, beta = 3"
```

```r
result3 <- gradientDescent(theta = c(0,3),
                          rhoFn = rho, gradientFn = gradient,
                          lineSearchFn = gridLineSearch,
                          testConvergenceFn = testConvergence)

Map(function(x){if (is.numeric(x)) round(x,3) else x}, result3)
```

```
## $theta
## [1] -2.885  2.721
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 8
##
## $fnValue
## [1] -63.26
```

All of them converge. The local minimum points are (-2.891 , -2.879), (2.732, 2.719) and (-2.885, 2.721) with value -77.283, -49.282 and -63.26 respectively. We note that different initial value coverges to different point which suggests that there are multiple local minimums. Comparing the value of each local minima, we conclude that (-2.891 , -2.879) is the point of gobal minima.

(e)

```r
gradientDescentWithSolutionPath <- function(theta,
      rhoFn, gradientFn, lineSearchFn, testConvergenceFn,
      maxIterations = 100,
      tolerance = 1E-6, relative = FALSE,
      lambdaStepsize = 0.01, lambdaMax = 0.5) {

  SolutionPath = matrix(NA,nrow = maxIterations + 2,ncol = length(theta))
  SolutionPath[1,] = theta
  converged <- FALSE
  i <- 0

  while (!converged & i <= maxIterations) {
    g <- gradientFn(theta) ## gradient
    glength <-  sqrt(sum(g^2)) ## gradient direction
    if (glength > 0) g <- g /glength
```

```r
    lambda <- lineSearchFn(theta, rhoFn, g,
                lambdaStepsize = lambdaStepsize, lambdaMax = lambdaMax)

    thetaNew <- theta - lambda * g
    converged <- testConvergenceFn(thetaNew, theta,
                                   tolerance = tolerance,
                                   relative = relative)
    theta <- thetaNew
    i <- i + 1
    SolutionPath[(i+1),] = theta
  }
  SolutionPath = SolutionPath[1:(i+1),]
  ## Return last value and whether converged or not
  list(theta = theta, converged = converged, iteration = i, fnValue = rhoFn(theta) ,
       SolutionPath = SolutionPath
       )
}
```

```r
Optim1 = gradientDescentWithSolutionPath(rhoFn = rho, gradientFn = gradient, theta = c(0,0),
           lineSearchFn = gridLineSearch, testConvergenceFn = testConvergence)

Optim2 = gradientDescentWithSolutionPath(rhoFn = rho, gradientFn = gradient, theta = c(1,1),
           lineSearchFn = gridLineSearch,testConvergenceFn = testConvergence)

Optim3 = gradientDescentWithSolutionPath(rhoFn = rho, gradientFn = gradient, theta = c(0,3),
           lineSearchFn = gridLineSearch,testConvergenceFn = testConvergence)

image(x1,x2,z,col = heat.colors(100))
contour(x1,x2,z,add=T )

n.arrows = dim(Optim1$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim1$SolutionPath[i,1],Optim1$SolutionPath[i,2],
       Optim1$SolutionPath[(i+1),1],Optim1$SolutionPath[(i+1),2],
       length = 0.12,angle = 15)
}
```

```
## Warning in arrows(Optim1$SolutionPath[i, 1], Optim1$SolutionPath[i, 2], :
## zero-length arrow is of indeterminate angle and so skipped
```

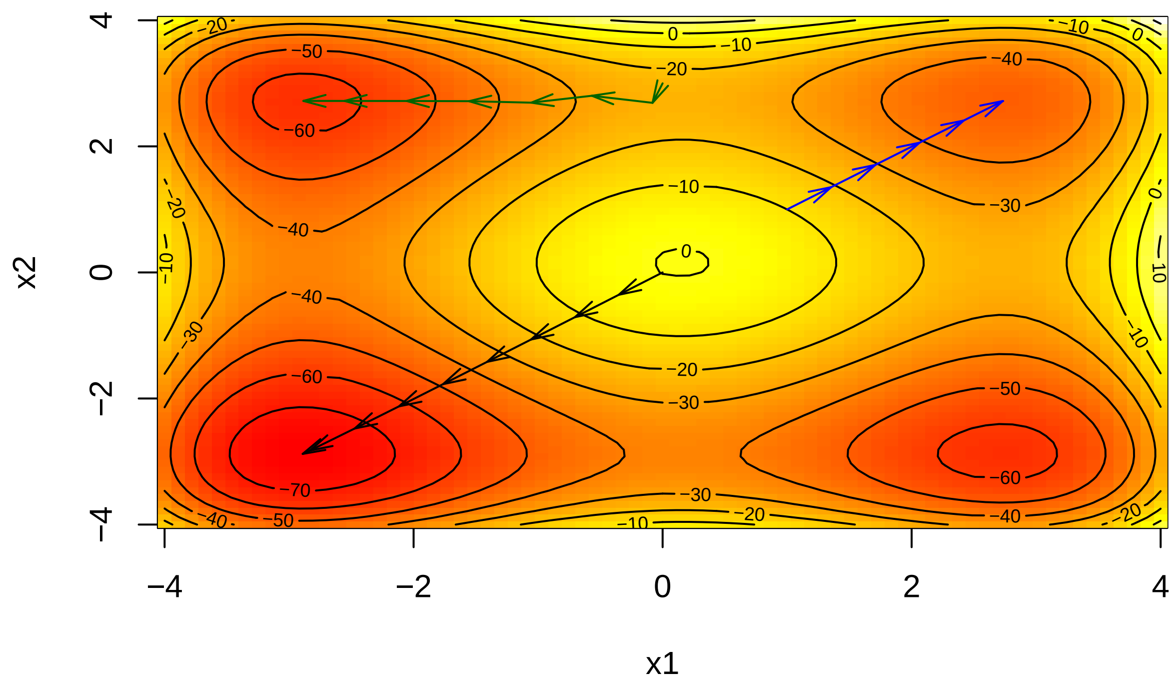
```r
n.arrows = dim(Optim2$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim2$SolutionPath[i,1],Optim2$SolutionPath[i,2],
       Optim2$SolutionPath[(i+1),1],Optim2$SolutionPath[(i+1),2],
       length = 0.12,angle = 15,col='blue')
}
```

```
## Warning in arrows(Optim2$SolutionPath[i, 1], Optim2$SolutionPath[i, 2], :
## zero-length arrow is of indeterminate angle and so skipped
```

```r
n.arrows = dim(Optim3$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim3$SolutionPath[i,1],Optim3$SolutionPath[i,2],
       Optim3$SolutionPath[(i+1),1],Optim3$SolutionPath[(i+1),2],
       length = 0.12,angle = 15,col='darkgreen')
```

```
}
```

```
## Warning in arrows(Optim3$SolutionPath[i, 1], Optim3$SolutionPath[i, 2], :
## zero-length arrow is of indeterminate angle and so skipped
```



(f)

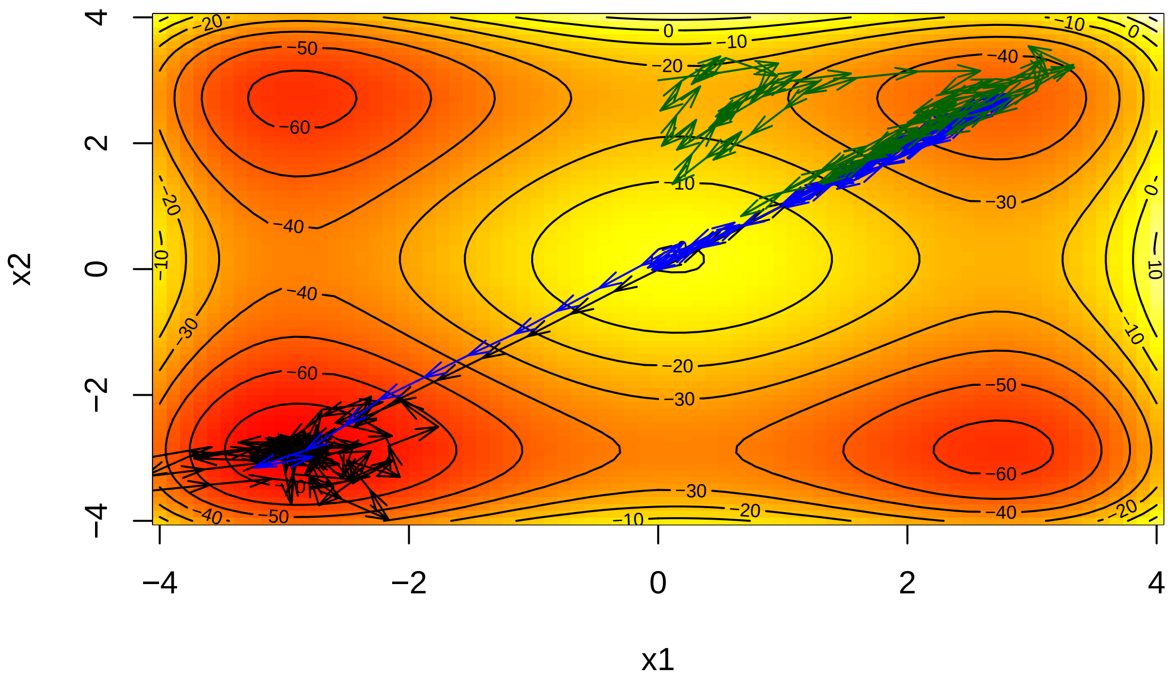
```
create.sgrad.fn <-function(pop, nsize) {
  function(theta) {
    alpha <- theta[1]
    beta <- theta[2]
    row = sample(1:nrow(pop), nsize, replace=FALSE)
    1 / nsize * c(sum(pop[row,1] * 4 * alpha^3 + pop[row,2] * 2 * alpha + pop[row,3]),
                  sum(pop[row,4] * 4 * beta^3 + pop[row,5] * 2 * beta + pop[row,6]))
  }
}
sample(xdata, 1, replace=FALSE)
```

```
## [1] -12.98826
```

(g)

```
fixedStep <- function(theta, rhoFn, g,
                      lambdaStepsize = 0.5,
                      lambdaMax = 1) {
  return(0.5)
}
```

```r
nostop <- function(thetaNew, thetaOld, tolerance = 1E-10, relative=FALSE) {
   FALSE
}

Optim4 <- gradientDescentWithSolutionPath(theta = c(0,0), rhoFn = rho,
          gradientFn = create.sgrad.fn(xdata, 1), lineSearchFn = fixedStep,
          testConvergenceFn = nostop, maxIterations=100, lambdaStepsize = 0.5, tolerance = 1)

Optim5 <- gradientDescentWithSolutionPath(theta = c(1,1), rhoFn = rho,
          gradientFn = create.sgrad.fn(xdata, 1), lineSearchFn = fixedStep,
          testConvergenceFn = nostop, maxIterations=100, lambdaStepsize = 0.5, tolerance = 1)

Optim6 <- gradientDescentWithSolutionPath(theta = c(0,3), rhoFn = rho,
          gradientFn = create.sgrad.fn(xdata, 1),
          lineSearchFn = fixedStep,
          testConvergenceFn = nostop, maxIterations=100, lambdaStepsize = 0.5, tolerance = 1)

image(x1,x2,z,col = heat.colors(100))
contour(x1,x2,z,add=T )

n.arrows = dim(Optim4$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim4$SolutionPath[i,1],Optim4$SolutionPath[i,2],
       Optim4$SolutionPath[(i+1),1],Optim4$SolutionPath[(i+1),2],
       length = 0.12,angle = 15)
}

n.arrows = dim(Optim5$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim5$SolutionPath[i,1],Optim5$SolutionPath[i,2],
       Optim5$SolutionPath[(i+1),1],Optim5$SolutionPath[(i+1),2],
       length = 0.12, angle = 15, col='blue')
}

n.arrows = dim(Optim6$SolutionPath)[1]
for(i in 1:(n.arrows-1)){
  arrows(Optim6$SolutionPath[i,1],Optim6$SolutionPath[i,2],
       Optim6$SolutionPath[(i+1),1],Optim6$SolutionPath[(i+1),2],
       length = 0.12, angle = 15,col='darkgreen')
}
```



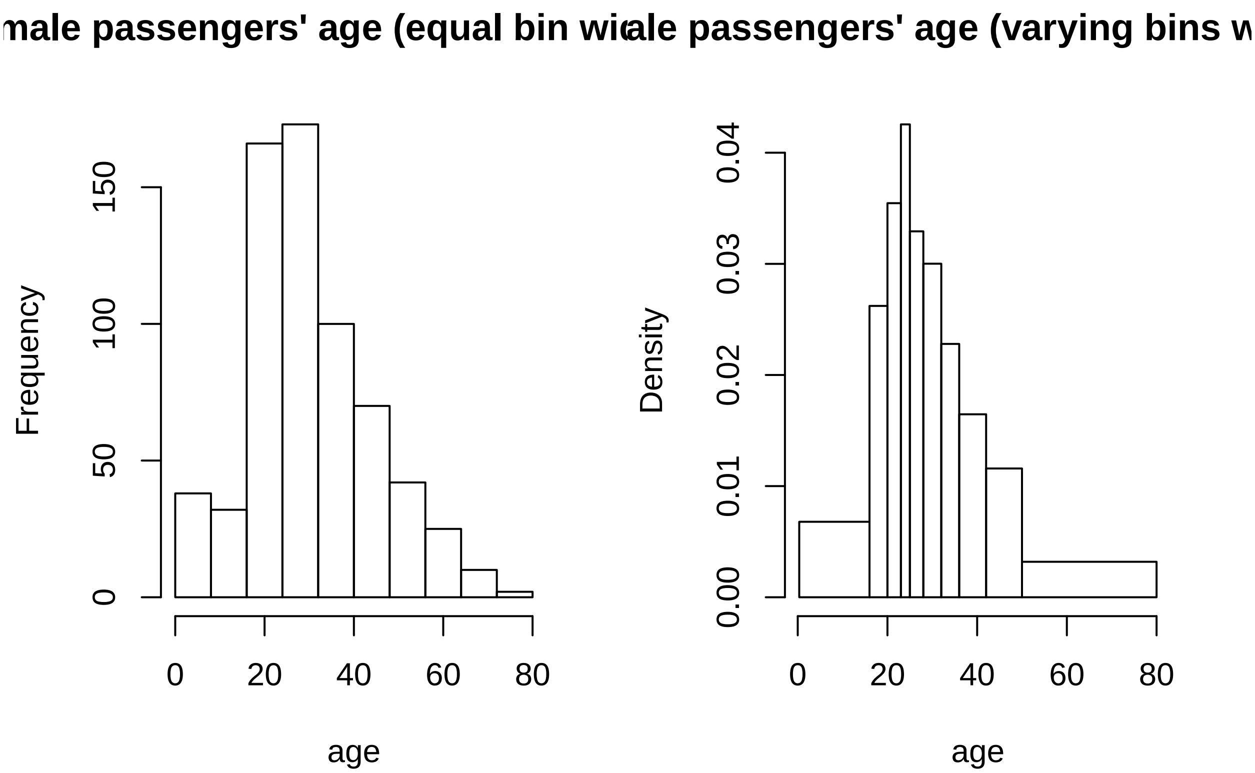
All three converges to the local minimum points with some detour. Then stay around the local minimum points

2.

```
library(carData)
data(TitanicSurvival)
Titanic = na.omit(TitanicSurvival)
Titanic = Titanic[Titanic$sex == "male",]
Titanic$survived1 = as.numeric(Titanic$survived == "yes")
```

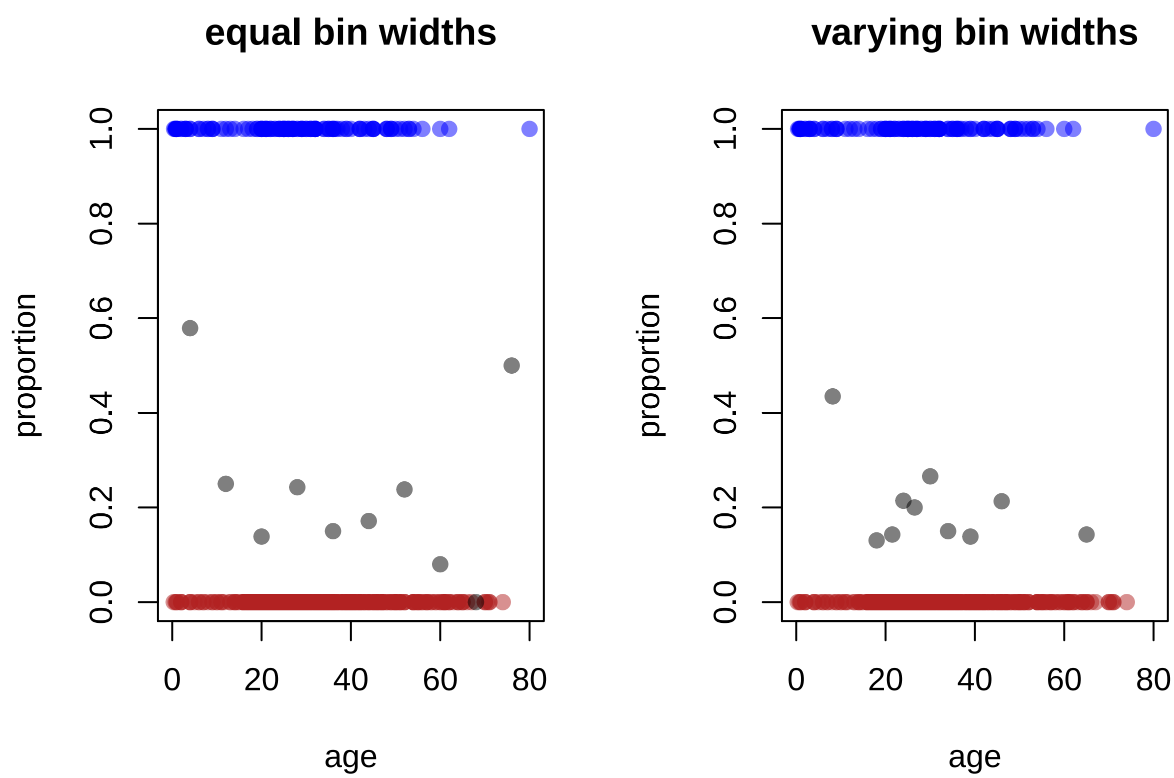
(a)

```
par(mfrow = c(1, 2))
hist(Titanic$age, breaks=seq(0, max(Titanic$age), by = 8),
     xlab="age", main = "male passengers' age (equal bin widths)")
hist(Titanic$age, breaks=quantile(Titanic$age, p=seq(0, 1, length.out=11)),
     xlab="age", main = "male passengers' age (varying bins widths)")
```



(b)

```
plot1a <- function(x, title) {
plot( Titanic$age, Titanic$survived1, pch=19,
      col=c(adjustcolor("firebrick",0.5), adjustcolor("blue", 0.5))[Titanic$survived1+1],
      xlim=c(0,80), xlab="age", ylab="proportion", main = title)

  propx = numeric(10)
  y = as.numeric(11)
  for (i in 2:length(x)) {
    y[i - 1] = (x[i] + x[i - 1]) / 2
    propx[i-1] = mean(Titanic$survived1[Titanic$age > x[i-1] & Titanic$age <= x[i]])
  }

  points( y, propx, pch=19,  col=adjustcolor("black", 0.5) ) }

par(mfrow=c(1,2))
plot1a(seq(0, max(Titanic$age), by = 8), "equal bin widths")
plot1a(quantile(Titanic$age, p=seq(0, 1, length.out=11)), "varying bin widths")
```




```
searchnum <- function(lw, hi) {
  survived = length(which(lw < Titanic$age & Titanic$age <= hi & Titanic$survived1 == 1))
  total = length(which(lw < Titanic$age & Titanic$age <= hi))
  return(c(survived, total))
}

table1 = matrix(nrow = 10, ncol = 5)
table2 = matrix(nrow = 10, ncol = 5)
age1 = seq(0, max(Titanic$age), by = 8)
age2 = quantile(Titanic$age, p=seq(0, 1, length.out = 11))

for(i in 1 : 10) {
  val1 = searchnum(age1[i], age1[i + 1])
  val2 = searchnum(age2[i], age2[i + 1])
  table1[i, 1] = age1[i]
  table1[i, 2] = age1[i + 1]
  table1[i, 3] = val1[1]
  table1[i, 4] = val1[2]
  table1[i, 5] = val1[1] / val1[2]
  table2[i, 1] = age2[i]
  table2[i, 2] = age2[i + 1]
  table2[i, 3] = val2[1]
  table2[i, 4] = val2[2]
  table2[i, 5] = val2[1] / val2[2]
}
par(mfrow = c(1, 2))
```

```r
paste("equal bin widths")
```

```
## [1] "equal bin widths"
```

```r
table1
```

```
##       [,1] [,2] [,3] [,4]      [,5]
##  [1,]    0    8   22   38 0.5789474
##  [2,]    8   16    8   32 0.2500000
##  [3,]   16   24   23  166 0.1385542
##  [4,]   24   32   42  173 0.2427746
##  [5,]   32   40   15  100 0.1500000
##  [6,]   40   48   12   70 0.1714286
##  [7,]   48   56   10   42 0.2380952
##  [8,]   56   64    2   25 0.0800000
##  [9,]   64   72    0   10 0.0000000
## [10,]   72   80    1    2 0.5000000
```

```r
paste("varying bin widths")
```

```
## [1] "varying bin widths"
```

```r
table2
```

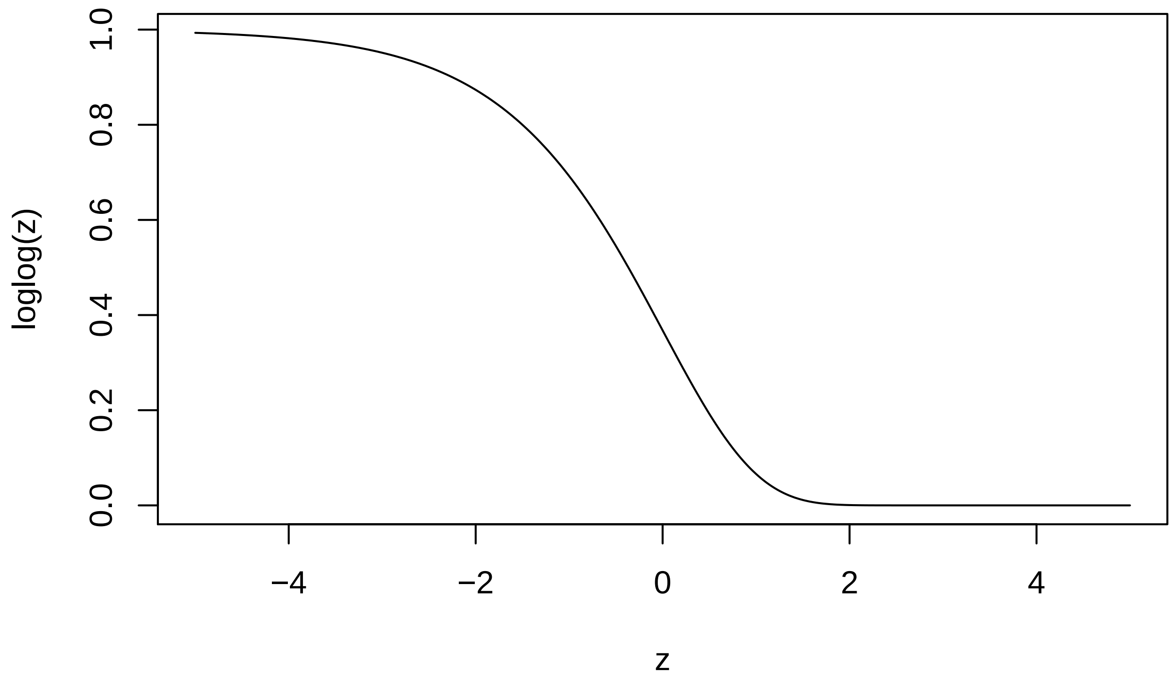
```
##          [,1] [,2] [,3] [,4]      [,5]
##  [1,]  0.3333   16   30   69 0.4347826
##  [2,] 16.0000   20    9   69 0.1304348
##  [3,] 20.0000   23   10   70 0.1428571
##  [4,] 23.0000   25   12   56 0.2142857
##  [5,] 25.0000   28   13   65 0.2000000
##  [6,] 28.0000   32   21   79 0.2658228
##  [7,] 32.0000   36    9   60 0.1500000
##  [8,] 36.0000   42    9   65 0.1384615
##  [9,] 42.0000   50   13   61 0.2131148
## [10,] 50.0000   80    9   63 0.1428571
```

The numbers of total passengers in each partition of equal bin with varys a lot. There are some intervals that has less that 10 passengers which can lead to extreme values and some intervals has more than a hundurd passengers. For example, there were only one male passenger whose age was between 72.03333 and 80, so the proportation can only be either 1 or 0, which makes it hard to find a relationship. However, the unequal bin partiton does not have that kind of problem. Each interval has relative resaonable number of people. In addition, the points in varying bin widths are more concentrated. On the other hand, the points in equal bin widths are more separated.

(c)

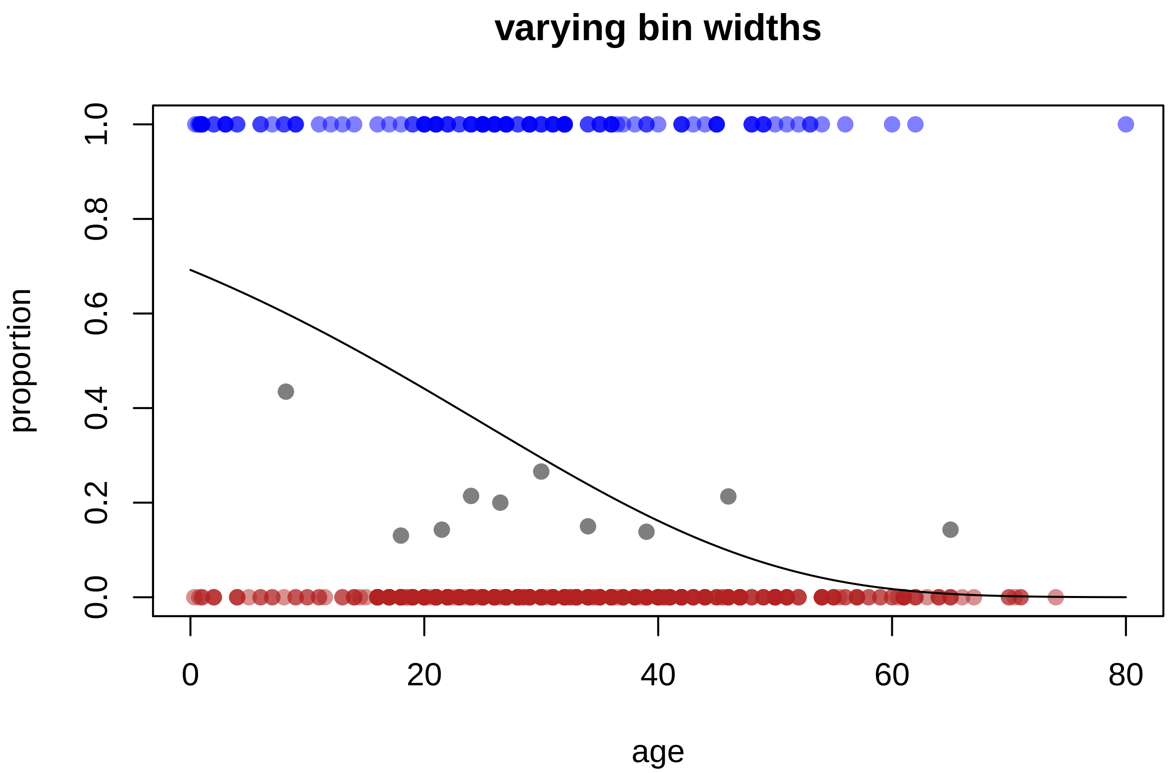
i.

```r
loglog <- function(z) {
  return(exp(-exp(z)))
}
z = seq(-5,5,.01)
plot(z,loglog(z), type='l')
```



ii.

```
plot1a(quantile(Titanic$age, p=seq(0, 1, length.out=11)), "varying bin widths")

z = seq(0, 80, .1)
lines( z, loglog( -1 + 0.04*z  ) )
```



(d) We define $g(x_i) = \frac{\partial p_i}{\partial(\alpha,\beta)}$

$$
\begin{aligned}
\frac{\partial p_i}{\partial \alpha} &= -e^{\alpha+\beta(x_i-\bar{x})}p_i \\
&= \log(p_i)p_i \\
\frac{\partial p_i}{\partial \beta} &= -(x_i-\bar{x})e^{\alpha+\beta(x_i-\bar{x})}p_i \\
&= (x_i-\bar{x})\log(p_i)p_i
\end{aligned}
$$

Thus $\frac{\partial l}{\partial(\alpha,\beta)} = -\log(p_i) \times p_i \times \begin{bmatrix} 1 \\ x_i - \bar{x} \end{bmatrix}$

$$\begin{aligned}
\frac{\partial l}{\partial(\alpha,\beta)} &= \sum_{i=1}^{N} y_i \frac{1-p_i}{p_i} \frac{\partial l}{\partial(\alpha,\beta)} \frac{p_i}{1-p_i} + \frac{1}{1-p_i}(-g(x_i)) \\
&= \sum_{i=1}^{N} y_i \frac{1-p_i}{p_i} \frac{g(x_i)(1-p_i) + g(x_i)p_i}{(1-p_i)^2} - \frac{g(x_i)}{1-p_i} \\
&= \sum_{i=1}^{N} \frac{g(x_i)y_i}{p_i(1-p_i)} - \frac{g(x_i)p_i}{p_i(1-p_i)} \\
&= \sum_{i=1}^{N} \frac{y_i - p_i}{p_i(1-p_i)} g(x_i) \\
&= \sum_{i=1}^{N} \frac{y_i - p_i}{p_i(1-p_i)} \times \log(p_i) \times p_i \times \begin{bmatrix} 1 \\ x_i - \bar{x} \end{bmatrix} \\
&= \sum_{i=1}^{N} \frac{y_i - p_i}{1-p_i} \times \log(p_i) \times \begin{bmatrix} 1 \\ x_i - \bar{x} \end{bmatrix}
\end{aligned}$$

(e)

i.

```
createObjBinary <- function(x,y) {
  ## local variable
  xbar <- mean(x)
  ## Return this function
  function(theta) {
    alpha <- theta[1]
    beta <- theta[2]
    y.hat = alpha + beta * (x - xbar)
    pi = loglog(y.hat)

    -1*sum( y*log(pi/(1-pi)) + log(1-pi)  )
  }
}
```

ii.

```
createBinaryLogisticGradient <- function(x,y) {
  ## local variables
  xbar <- mean(x)
  ybar <- mean(y)
  N <- length(x)

  function(theta) {
    alpha <- theta[1]
    beta <- theta[2]
    y.hat = alpha + beta * (x - xbar)
    pi = loglog(y.hat)
    resids = y - pi

    -1*c( sum(resids / (1 - pi) * log(pi)),  sum( (x - xbar) * resids / (1 - pi) * log(pi))  )
```

```
  }
}
```

iii.

```
gradient <- createBinaryLogisticGradient(Titanic$age, Titanic$survived1)
rho <- createObjBinary(Titanic$age, Titanic$survived1)

result <- gradientDescent(theta = c(0, 0),
                          rhoFn = rho, gradientFn = gradient,
                          lineSearchFn = gridLineSearch,
                          testConvergenceFn = testConvergence,
                          lambdaStepsize = 0.0001,
                          lambdaMax = 0.01,
                          maxIterations = 10^5)
### Print the results
Map(function(x){if (is.numeric(x)) round(x,3) else x}, result)
```

```
## $theta
## [1] 0.464 0.010
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 281
##
## $fnValue
## [1] 328.892
```

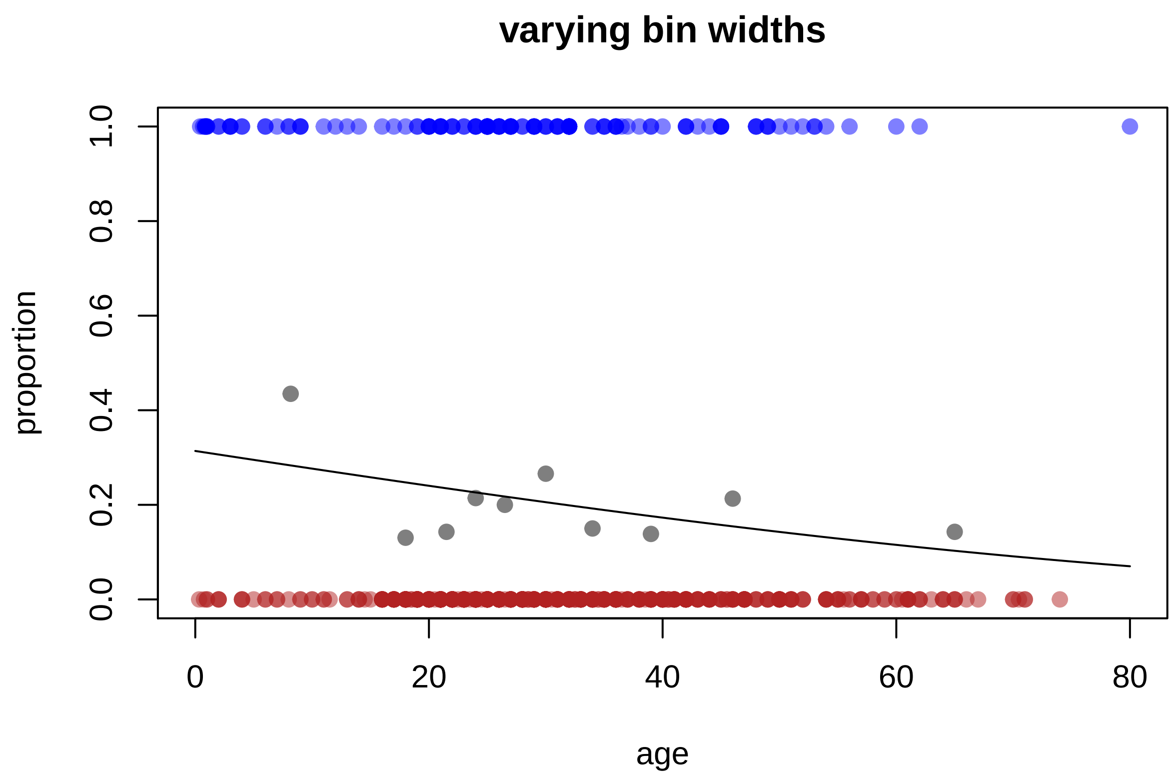
iv. If the age having no effect on survival, then $\beta = 0$, α should be the overall survival rate which is 0.2051672 so $\alpha = 0.2051672$, $\beta = 0$

```
result <- gradientDescent(theta = c(0.2051672, 0),
                          rhoFn = rho, gradientFn = gradient,
                          lineSearchFn = gridLineSearch,
                          testConvergenceFn = testConvergence,
                          lambdaStepsize = 0.0001,
                          lambdaMax = 0.01,
                          maxIterations = 10^5)
### Print the results
result
```

```
## $theta
## [1] 0.46472501 0.01037943
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 204
##
## $fnValue
## [1] 328.891
```

Yes, there is an improvement as the number of iterations is reduced. (g)

```r
plot1a(quantile(Titanic$age, p=seq(0, 1, length.out=11)), "varying bin widths")

z = seq(0, 80, .1)
lines( z, loglog(result[1]$theta[1] + result[1]$theta[2]*(z - mean(Titanic$age))) )
```



ii.

```r
x = quantile(Titanic$age, p=seq(0, 1, length.out=11))
propx1 = loglog(result[1]$theta[1] + result[1]$theta[2]*(x[-11] - mean(Titanic$age)))
propx1 = cbind(propx1[2], table2[,5])
propx1
```

```
##            [,1]      [,2]
##  [1,] 0.2546196 0.4347826
##  [2,] 0.2546196 0.1304348
##  [3,] 0.2546196 0.1428571
##  [4,] 0.2546196 0.2142857
##  [5,] 0.2546196 0.2000000
##  [6,] 0.2546196 0.2658228
##  [7,] 0.2546196 0.1500000
##  [8,] 0.2546196 0.1384615
##  [9,] 0.2546196 0.2131148
## [10,] 0.2546196 0.1428571
```

iii. We assume for both of models that whether a male passenger survived is independent from other male passengers. For the parametric model, we assume that there is log-log relationship between survival rate and male passengers' age. For the non-parametric model, we assume that there is a relationship between survival rate and male passengers' age.

iv. $p = f(0.46472501 + 0.01037943[x - \bar{x}]) = \frac{1}{2}$

$$
\begin{aligned}
-e^{\hat{\alpha}+\hat{\beta}(x-\bar{x})} &= \log(\frac{1}{2}) \\
\hat{\alpha} + \hat{\beta}(x - \bar{x}) &= \log(-\log(\frac{1}{2})) \\
(x - \bar{x}) &= \frac{\log(-\log\frac{1}{2}) - \hat{\alpha}}{\hat{\beta}}
\end{aligned}
$$

```
val = (log(-log(1 / 2)) - result[1]$theta[1]) / result[1]$theta[2]
val + mean(Titanic$age)
```

```
## [1] -49.49989
```

$x = -49.49989$ which concludes that no age has a 50-50 chance of survival.