A3Q1

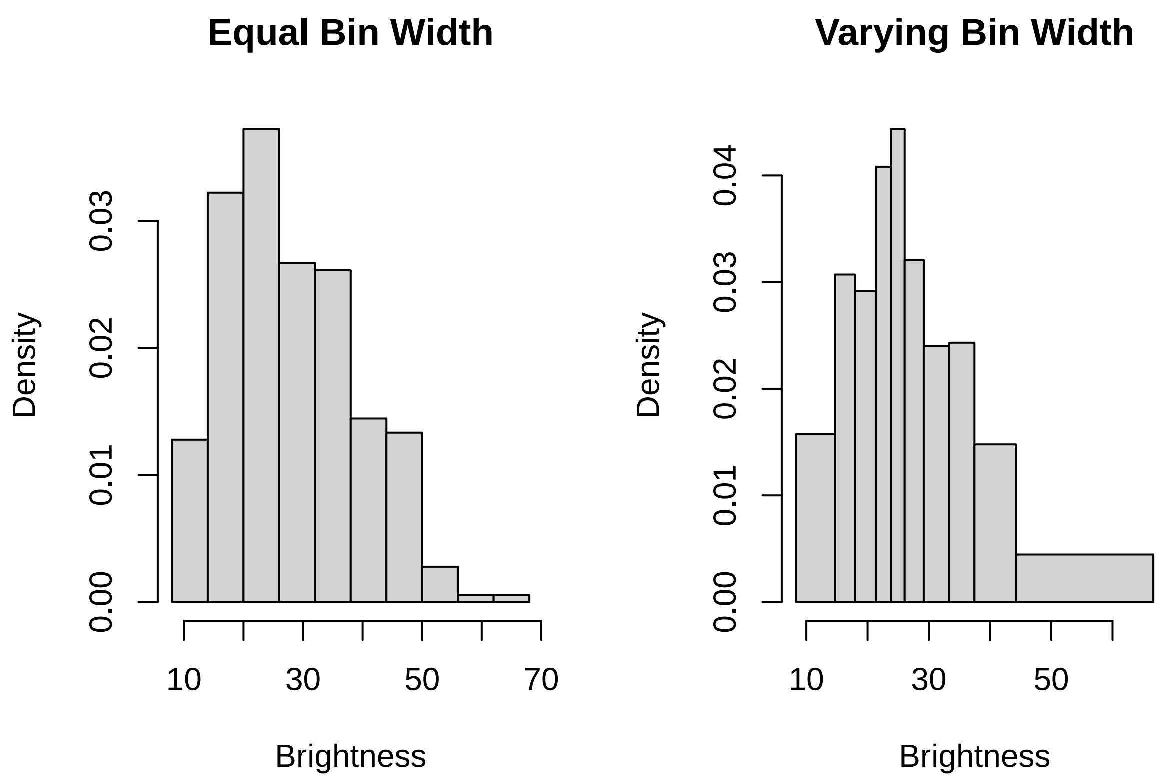
a)

```
one <- read.csv("one180.csv", header = T)
two <- read.csv("two120.csv", header = T)

Brightness <- c(rowMeans(one), rowMeans(two))
Digit1 <- c(rep(1, 180), rep(0, 120))

df <- data.frame('Brightness' = Brightness, 'Digit1' = Digit1)
head(df)
```

```
##   Brightness Digit1
## 1   14.11862      1
## 2   17.67985      1
## 3   12.10204      1
## 4   12.08929      1
## 5   19.60587      1
## 6   10.18750      1
```

b)

```
par(mfrow = c(1, 2))
hist(df$Brightness, breaks = seq(8, 68, 6), prob = TRUE,
     main = 'Equal Bin Width', xlab = 'Brightness')
quant.brightness <- quantile(df$Brightness, seq(0, 1, length.out = 11))
hist(df$Brightness, breaks = quant.brightness, main = 'Varying Bin Width',
          xlab = 'Brightness')
```



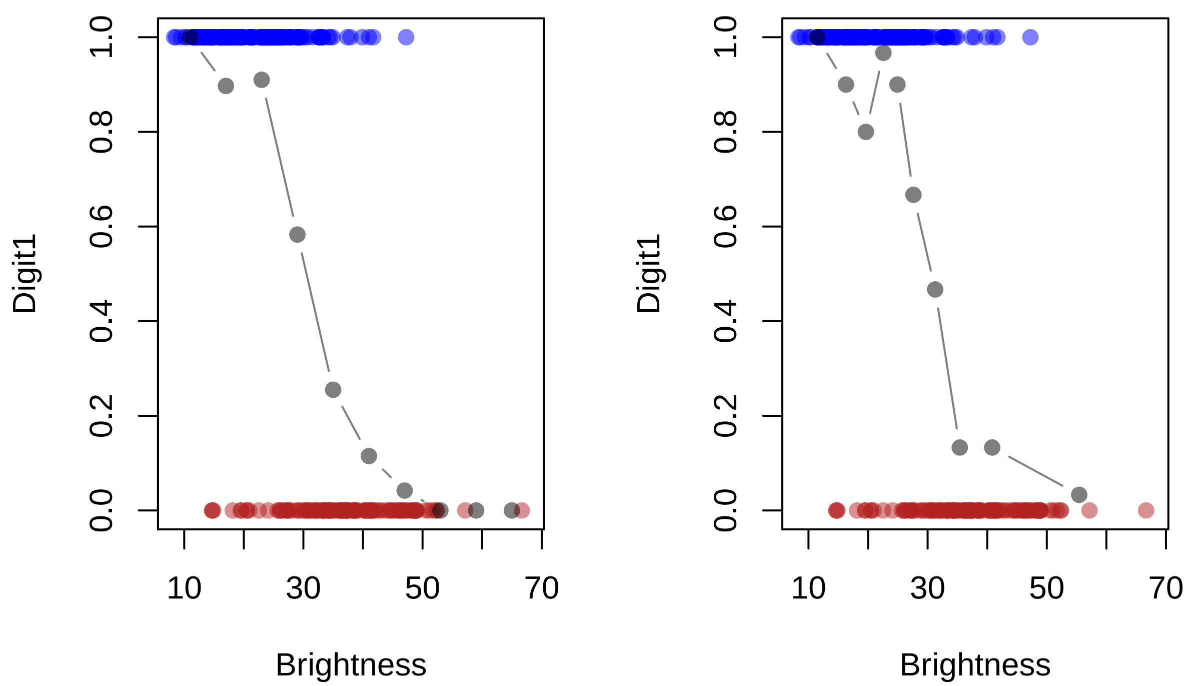
c)

```
plotting <- function(bright) {
  plot( df$Brightness, df$Digit1, pch=19,
        col=c(adjustcolor("firebrick",0.5),
              adjustcolor("blue", 0.5))[df$Digit1 +1],
        xlim=c(8,68), xlab="Brightness", ylab="Digit1" )

  propx = matrix(0, nrow=length(bright)-1, ncol=5)
  dimnames(propx)[[2]] = c("Lower", "Upper", "Total",
                           "Num.Digit1", "Prop.Digit1")
  for (i in 1:nrow(propx)) {
    propx[i,1:2] = c(bright[i], bright[i+1])
    propx[i,3] = sum(df$Brightness > bright[i] & df$Brightness <= bright[i+1])
    propx[i,4] = sum(df$Digit1[df$Brightness > bright[i] &
                               df$Brightness <= bright[i+1]] )
    propx[i,5] = round(propx[i,4]/propx[i,3],3)
  }

  points( bright[-length(bright)]+ diff(bright)/2,
          propx[,5], pch=19,  col=adjustcolor("black", 0.5), type='b' )
  propx
}
```

```r
bright1 = seq(8, 68, 6)
bright2 = quantile(df$Brightness, seq(0,1,length.out=11))
par(mfrow = c(1,2))
plotting(bright1)
```

```
##       Lower Upper Total Num.Digit1 Prop.Digit1
##   [1,]     8    14    23         23       1.000
##   [2,]    14    20    58         52       0.897
##   [3,]    20    26    67         61       0.910
##   [4,]    26    32    48         28       0.583
##   [5,]    32    38    47         12       0.255
##   [6,]    38    44    26          3       0.115
##   [7,]    44    50    24          1       0.042
##   [8,]    50    56     5          0       0.000
##   [9,]    56    62     1          0       0.000
## [10,]    62    68     1          0       0.000
```

```r
plotting(bright2)
```




```
##           Lower    Upper Total Num.Digit1 Prop.Digit1
##   [1,]  8.316327 14.66696    29         29       1.000
##   [2,] 14.666964 17.92296    30         27       0.900
##   [3,] 17.922959 21.35332    30         24       0.800
##   [4,] 21.353316 23.80306    30         29       0.967
```

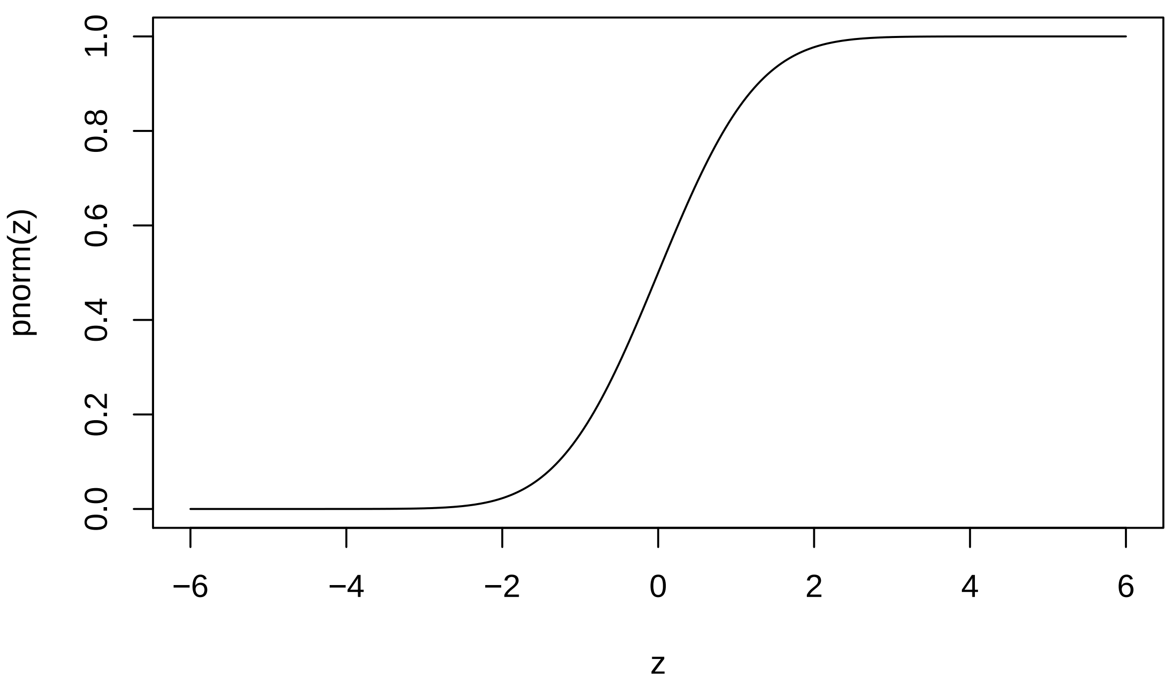
```
##  [5,] 23.803061 26.05804    30          27       0.900
##  [6,] 26.058036 29.17602    30          20       0.667
##  [7,] 29.176020 33.34145    30          14       0.467
##  [8,] 33.341454 37.45332    30           4       0.133
##  [9,] 37.453316 44.21786    30           4       0.133
## [10,] 44.217857 66.66709    30           1       0.033
```

comment: From two tables, we can see that bright 2 has nearly equal number of 1s within each group, while bright 1 has the same interval width. Also, we see that Digit 2 are generally brighter than Digit 1 as it has more brightness in the middle. Digit 1's prob line is smoother than the digit 2. We can also see that there are only a few 1's for the last three to four bins for two tables.

d)

(i)

```r
z = seq(-6, 6, .01)
plot(z, pnorm(z), type='l')
```

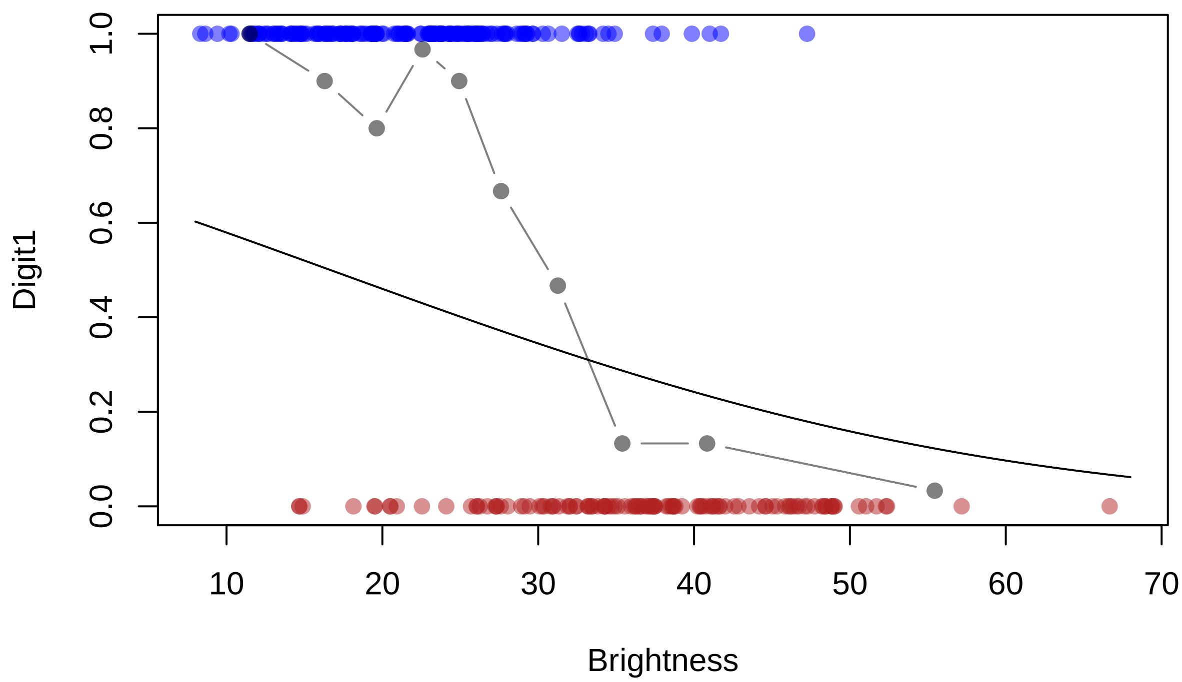


(ii)

```r
plotting(bright2)
```

```
##           Lower    Upper Total Num.Digit1 Prop.Digit1
##  [1,]  8.316327 14.66696    29         29       1.000
##  [2,] 14.666964 17.92296    30         27       0.900
##  [3,] 17.922959 21.35332    30         24       0.800
##  [4,] 21.353316 23.80306    30         29       0.967
##  [5,] 23.803061 26.05804    30         27       0.900
##  [6,] 26.058036 29.17602    30         20       0.667
##  [7,] 29.176020 33.34145    30         14       0.467
##  [8,] 33.341454 37.45332    30          4       0.133
##  [9,] 37.453316 44.21786    30          4       0.133
## [10,] 44.217857 66.66709    30          1       0.033
```

```r
z = seq(8, 68, .01)
lines(z, pnorm(1/2-0.03*z))
```



e)

$$Formula:$$

$$l(\theta) = l(\alpha, \beta) = \frac{1}{N}\sum_{i=1}^{N}[y_i log\frac{p_i}{1-p_i} + log(i-p_i)]$$

$$p_i = \Phi(\bar{y}) = \Phi(\alpha + \beta[x_i - \bar{x}])$$

$$Derivation:$$

$$\frac{\partial l}{\partial \alpha} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial p_i}\frac{\partial p_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \alpha} \qquad AND \qquad \frac{\partial l}{\partial \beta} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial p_i}\frac{\partial p_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \beta}$$

$$\frac{\partial l_i}{\partial p_i} = \frac{\partial}{\partial p_i}[y_i log\frac{p_i}{1-p_i} - log(1-p_i)] = \frac{y_i - p_i}{p_i(1-p_i)}$$

$$\frac{\partial p_i}{\partial \hat{y}_i} = \frac{\partial p_i}{\partial \hat{y}_i}\Phi(\hat{y}_i) = \phi(\hat{y}_i)$$

$$\frac{\partial \hat{y}_i}{\partial \alpha} = \frac{\partial}{\partial \alpha}(\alpha + \beta[x_i - \bar{x}]) = 1 \qquad AND \qquad \frac{\partial \hat{y}_i}{\partial \beta} = \frac{\partial}{\partial \beta}(\alpha + \beta[x_i - \bar{x}]) = x_i - \bar{x}$$

$$\frac{\partial l}{\partial \alpha} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial p_i}\frac{\partial p_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \alpha} = \frac{1}{N}\sum_{i=1}^{N}\frac{y_i - p_i}{p_i(1-p_i)} \times \phi(\hat{y}_i) \times 1$$

$$\frac{\partial l}{\partial \beta} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l_i}{\partial p_i}\frac{\partial p_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \beta} = \frac{1}{N}\sum_{i=1}^{N}\frac{y_i - p_i}{p_i(1-p_i)} \times \phi(\hat{y}_i) \times (x_i - \bar{x})$$

$$Therefore,$$

$$\frac{\partial l}{\partial(\alpha, \beta)} = \frac{1}{N}\sum_{i=1}^{N}\frac{y_i - p_i}{p_i(1-p_i)} \times \phi(\hat{y}) \times \begin{bmatrix} 1 \\ x_i - \bar{x} \end{bmatrix}$$

f)

(i)

```r
createObjProbit <- 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 = pnorm(y.hat)

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

(ii)

```r
createGradientProbit <- 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 = pnorm(y.hat)
    resids = y - pi
    wt = dnorm(y.hat)/(pi*(1-pi))

    -1*c(mean(resids*wt), mean(wt*(x - xbar) * resids))
  }
}
```

(iii)

```r
gradient <- createGradientProbit(df$Brightness, df$Digit1)
rho <- createObjProbit(df$Brightness, df$Digit1)


result <- gradientDescent(theta = c(0,0),
                          rhoFn = rho, gradientFn = gradient,
                          lineSearchFn = gridLineSearch,
                          testConvergenceFn = testConvergence,
                          lambdaStepsize = 0.001,
                          lambdaMax = 1)

Map(function(x){if (is.numeric(x)) round(x,4) else x}, result)
```

```
## $theta
## [1]  0.3284 -0.1181
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 97
##
## $fnValue
## [1] 0.3929
```

(iv)

If brightness has no effect on Digit1 then $\beta = 0$.

If the proportion of one is 180/300=0.6 then we need to solve $\phi(\alpha) = 0.6$ for α.

```r
qnorm(0.6)
```

```
## [1] 0.2533471
```

We get α is 0.2533471.

```r
result <- gradientDescent(theta = c(qnorm(0.6),0),
                          rhoFn = rho, gradientFn = gradient,
                          lineSearchFn = gridLineSearch,
                          testConvergenceFn = testConvergence,
                          lambdaStepsize = 0.001,
                          lambdaMax = 1)

Map(function(x){if (is.numeric(x)) round(x,4) else x}, result)
```

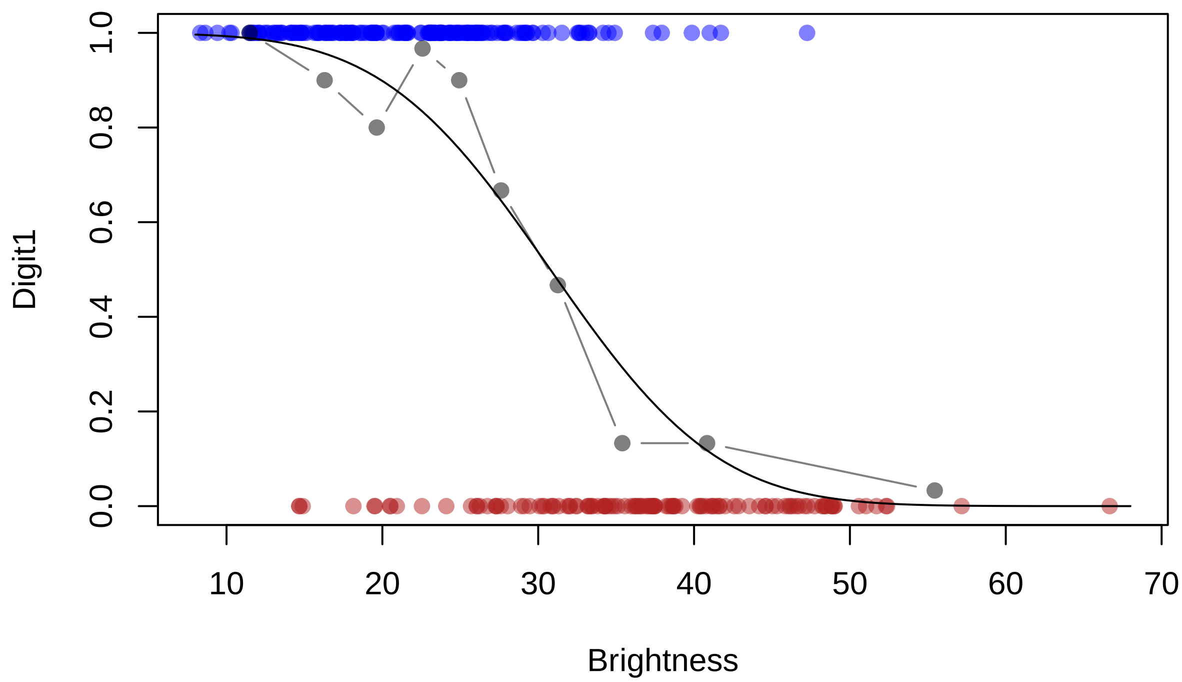
```
## $theta
## [1]  0.3283 -0.1181
##
## $converged
## [1] TRUE
##
## $iteration
## [1] 93
##
## $fnValue
## [1] 0.3929
```

We can see that there is an improvement on the number of iterations as the number of iterations is reduced.

g)

(i)

```r
temp = plotting(bright2)
z = seq(8, 68, length.out = 100)
lines(z, pnorm(0.3284 - 0.1181*(z- mean(df$Brightness))))
```



(ii)

```
x = apply(temp[,1:2],1, mean)
probit.prop = pnorm(0.3284 - 0.1181*(x- mean(df$Brightness)))
propx2 = cbind(temp, probit.prop)
round(propx2,3)
```

```
##        Lower  Upper Total Num.Digit1 Prop.Digit1 probit.prop
##  [1,]  8.316 14.667    29         29       1.000       0.989
##  [2,] 14.667 17.923    30         27       0.900       0.956
##  [3,] 17.923 21.353    30         24       0.800       0.906
##  [4,] 21.353 23.803    30         29       0.967       0.834
##  [5,] 23.803 26.058    30         27       0.900       0.755
##  [6,] 26.058 29.176    30         20       0.667       0.646
##  [7,] 29.176 33.341    30         14       0.467       0.477
##  [8,] 33.341 37.453    30          4       0.133       0.293
##  [9,] 37.453 44.218    30          4       0.133       0.117
## [10,] 44.218 66.667    30          1       0.033       0.002
```

comments: We can see that the probit.prop and prop.Digit1 is really similar, nearly the same in bin 7. However, in bin 5, the number is slightly off as there is a big gap between Prop.Digit1 and Probit.Prop.

(iii)

- The proportion of 1s is monotonic increasing or decreasing under the parametric model assumption.
- The proportions of 1s over each interval is constant under the non-parametric model assumption.

(iv)

$$\phi(0.3284 - 0.1181 \times (x - \bar{x})) = 0.5$$

```
(qnorm(0.5)-0.3284)/(-0.1181)+mean(df$Brightness)
```

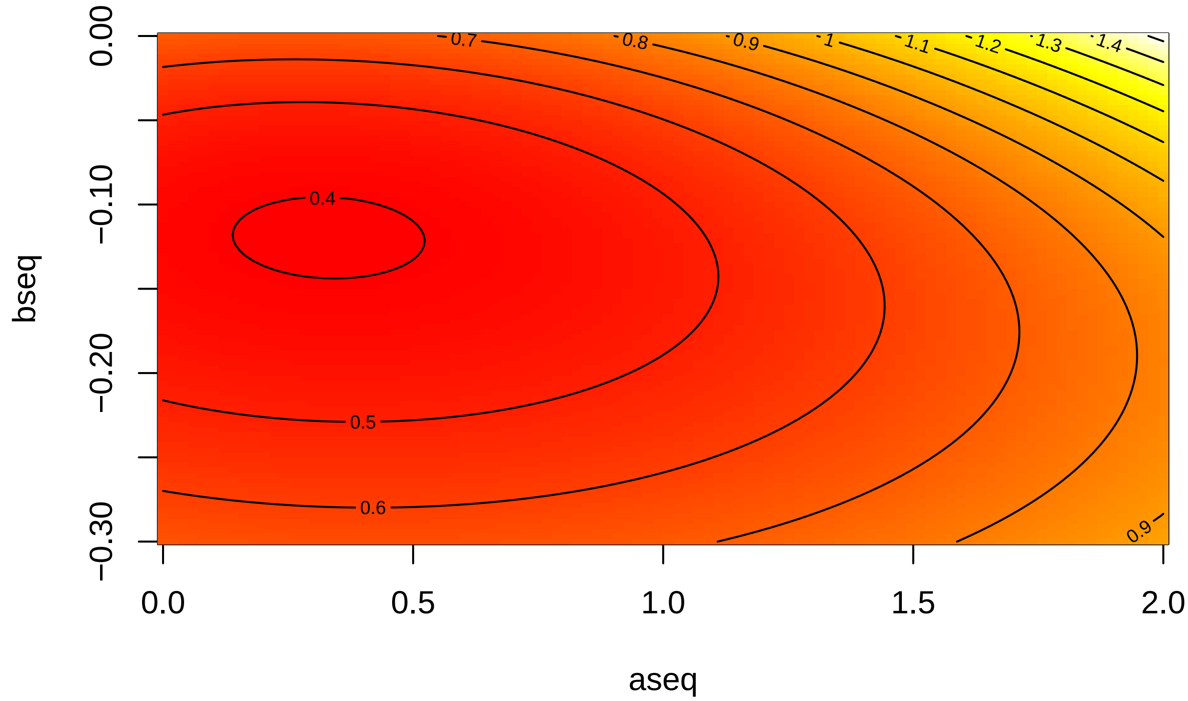
```
## [1] 30.77774
```

The average pixel brightness would be: 30.77774.

h)

(i)

```
create.rho.contour.fn <- function(x, y) {
  xbar <- mean(x)
  ## Return this function
  function(alpha, beta) {
    y.hat = alpha + beta * (x - xbar)
    pi = pnorm(y.hat)
    -1*mean(y*log(pi/(1-pi)) + log(1-pi))
  }
}

rho.to.plot = Vectorize(create.rho.contour.fn(df$Brightness, df$Digit1))

aseq = seq(0, 2,length = 100)
bseq = seq(-0.3, 0, length = 100)
z = outer(aseq, bseq , FUN=rho.to.plot)

image(aseq, bseq, z, col = heat.colors(100))
contour(aseq, bseq, z, add=T)
```



(ii)

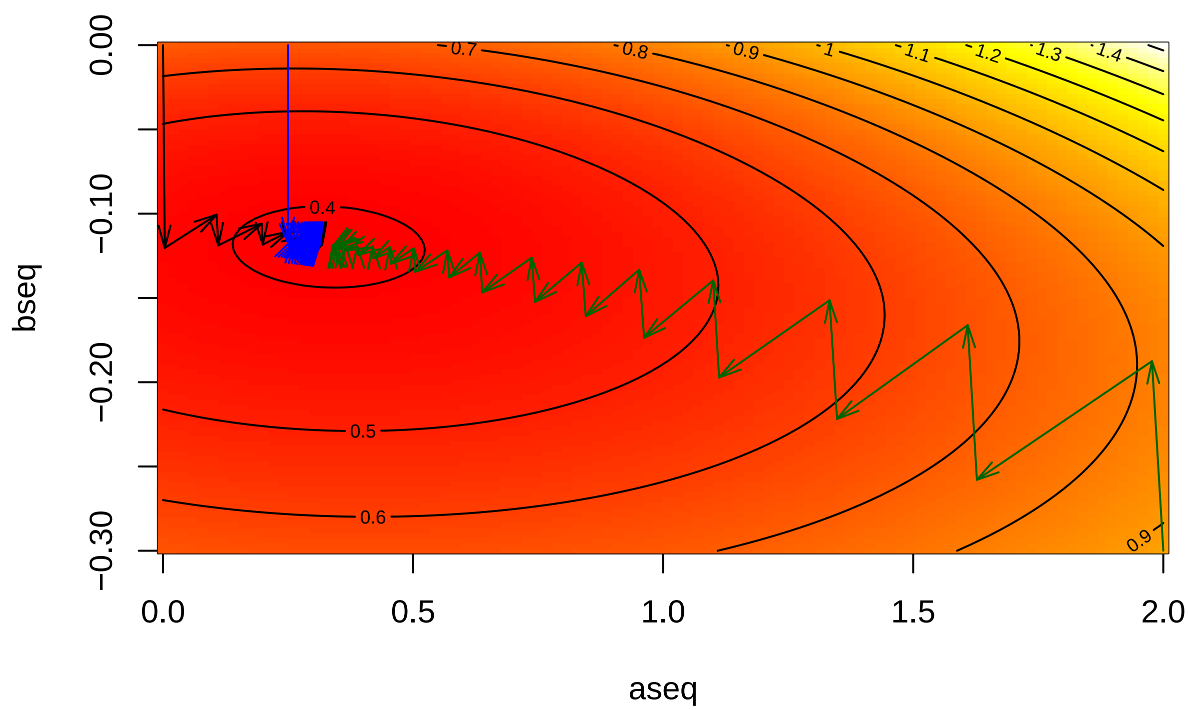
```
gradientDescentWithSolutionPath <- function(theta,
      rhoFn, gradientFn, lineSearchFn, testConvergenceFn,
      maxIterations = 100,
      tolerance = 1E-6, relative = FALSE,
      lambdaStepsize = 0.01, lambdaMax = 0.5) {

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

  while (!converged & i < (maxIterations-1) ) {
    g <- gradientFn(theta) ## gradient

    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)
}
```

(iii)

```
Optim1 = gradientDescentWithSolutionPath(rhoFn = rho,
                                         gradientFn = gradient,
                                         theta = c(0,0),
                                         lineSearchFn = gridLineSearch,
                                         testConvergenceFn = testConvergence,
                                         lambdaStepsize = 0.001,
                                         lambdaMax = 1,
                                         tolerance = 1e-3)
param1 = Optim1$theta

Optim2 = gradientDescentWithSolutionPath(rhoFn = rho,
                                         gradientFn = gradient,
                                         theta = c(0.25,0),
                                         lineSearchFn = gridLineSearch,
                                         testConvergenceFn = testConvergence,
                                         lambdaStepsize = 0.001,
                                         lambdaMax = 1,
                                         tolerance = 1e-3)
param2 = Optim2$theta

Optim3 = gradientDescentWithSolutionPath(rhoFn = rho,
                                         gradientFn = gradient,
                                         theta = c(2,-0.3),
                                         lineSearchFn = gridLineSearch,
                                         testConvergenceFn = testConvergence,
                                         lambdaStepsize = 0.001,
                                         lambdaMax = 1,
                                         tolerance = 1e-3)
param3 = Optim3$theta

image(aseq, bseq, z, col = heat.colors(100) )
contour(aseq, bseq, 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)
}
```

```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')
}

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')
}
```




```r
knitr::kable(rbind('Optim1'=c(0,0,round(Optim1$theta[1],4),round(Optim1$theta[2],4),
                    Optim1[2:3],round(as.numeric(Optim1[4]),4)),
            'Optim2'=c(0.25,0,round(Optim2$theta[1],4),round(Optim2$theta[2],4),
                    Optim2[2:3],round(as.numeric(Optim2[4]),4)),
            'Optim3'=c(2,-0.3,round(Optim3$theta[1],4),round(Optim3$theta[2],4),
                    Optim3[2:3],round(as.numeric(Optim3[4]),4))),
      col.names = c('alpha0', 'beta0', 'alpha^*', 'beta^*', 'converged',
                   'iteration','fnValue'),
      align='c')
```

	alpha0	beta0	alpha^*	beta^*	converged	iteration	fnValue
Optim1	0	0	0.318	-0.1186	TRUE	19	0.3929
Optim2	0.25	0	0.3138	-0.1186	TRUE	34	0.3929
Optim3	2	-0.3	0.3395	-0.1187	TRUE	35	0.3929

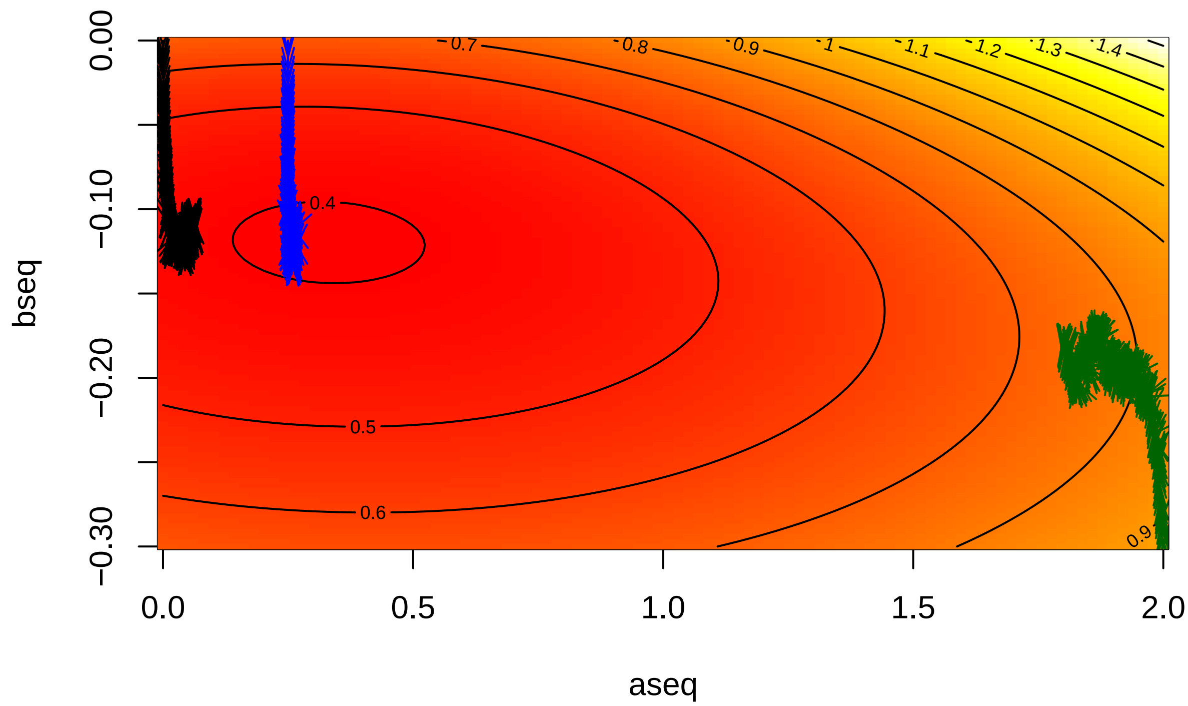
(iv)

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

  function(theta) {
    alpha <- theta[1]
    beta <- theta[2]

    subset = sample(N, nsize)
    x = x[subset]
    y = y[subset]

    y.hat = alpha + beta * (x - xbar)
    pi = pnorm(y.hat)
    resids = y - pi
    wt = dnorm(y.hat)/(pi*(1-pi))

    -1*c(mean(resids*wt), mean(wt*(x - xbar) * resids))
  }
}
```

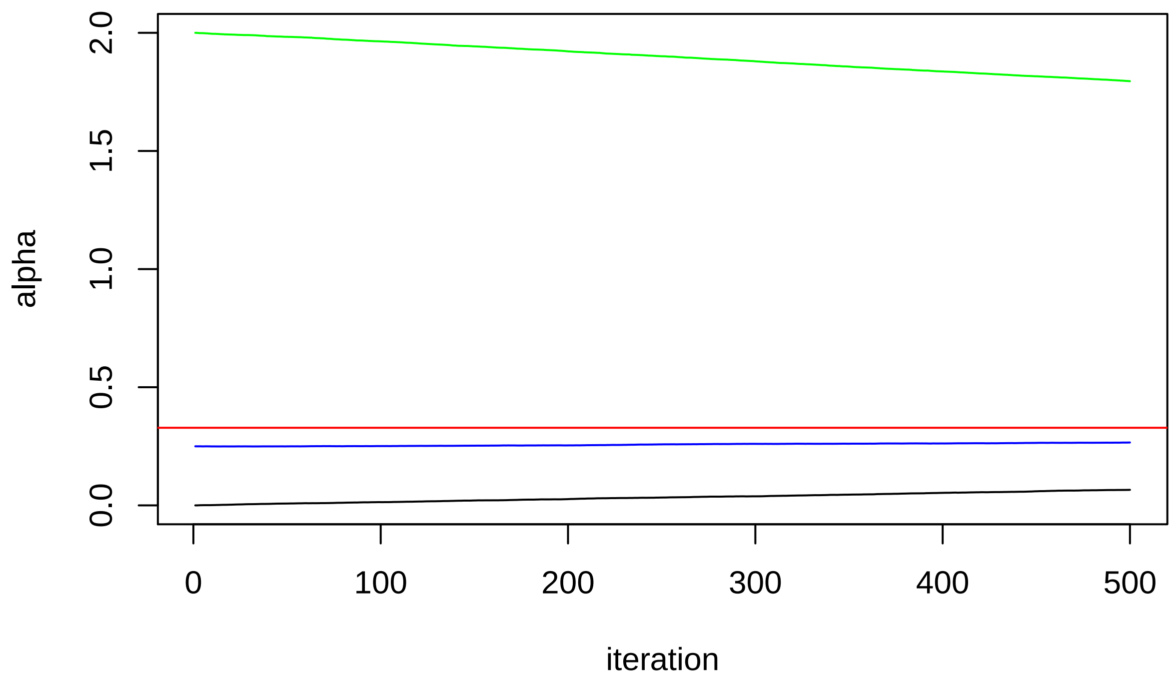
(v)

```r
sgrad.fn = createStochasticGrad(df$Brightness, df$Digit1, nsize = 25)

fixedLineSearch <- function(theta, rhoFn, g,
                            lambdaStepsize = 0.001,
                            lambdaMax = 1) {
  lambdaStepsize
}

Optim1 = gradientDescentWithSolutionPath(rhoFn = rho,
                                         gradientFn = sgrad.fn,
                                         theta = c(0,0),
                                         lineSearchFn = fixedLineSearch,
                                         testConvergenceFn = testConvergence,
                                         lambdaStepsize = 0.001,
                                         maxIterations = 500)
param1 = Optim1$theta
```

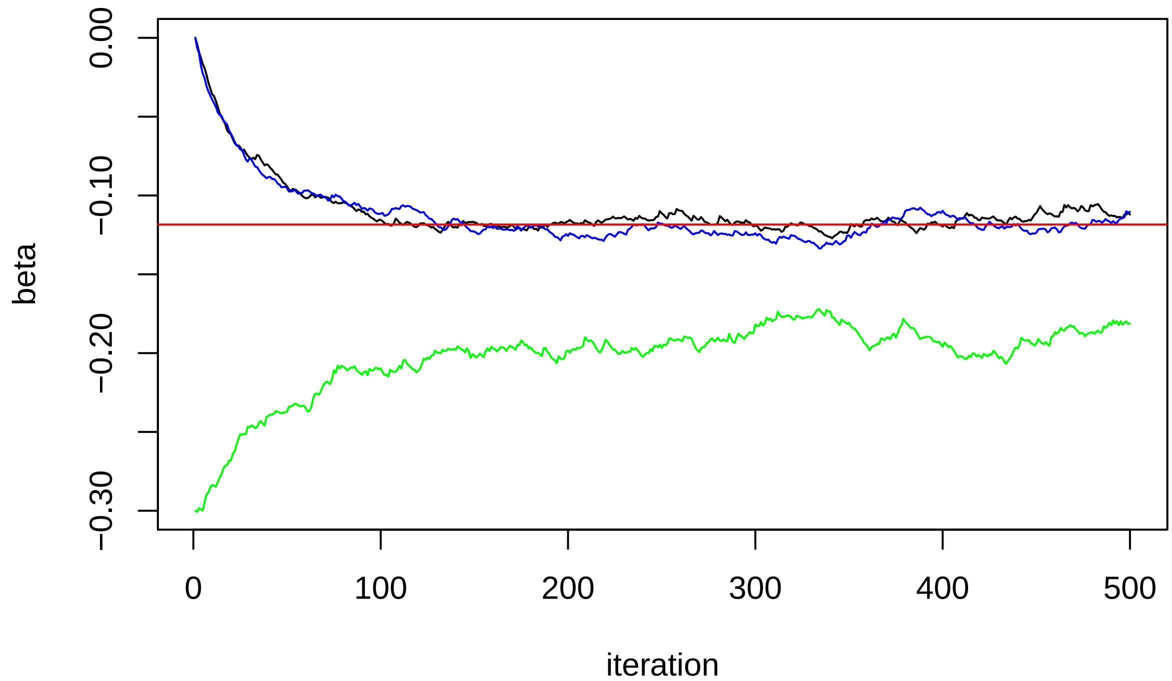
```r
Optim2 = gradientDescentWithSolutionPath(rhoFn = rho,
                                         gradientFn = sgrad.fn,
                                         theta = c(0.25,0),
                                         lineSearchFn = fixedLineSearch,
                                         testConvergenceFn = testConvergence,
                                         lambdaStepsize = 0.001,
                                         maxIterations = 500)
param2 = Optim2$theta

Optim3 = gradientDescentWithSolutionPath(rhoFn = rho,
                                         gradientFn = sgrad.fn,
                                         theta = c(2,-0.3),
                                         lineSearchFn = fixedLineSearch,
                                         testConvergenceFn = testConvergence,
                                         lambdaStepsize = 0.001,
                                         maxIterations = 500)
param3 = Optim3$theta

image(aseq, bseq, z, col = heat.colors(100) )
contour(aseq, bseq, 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)
}

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')
}

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')
}
```




```r
plot(Optim1$SolutionPath[, 1], type='l', ylim=c(0,2),
     xlab='iteration', ylab='alpha')
lines(Optim2$SolutionPath[,1], col='blue')
lines(Optim3$SolutionPath[,1], col='green')
abline(h=0.3284, col='red')
```




```r
plot(Optim1$SolutionPath[, 2], type='l', ylim=c(-0.3,0),
     xlab='iteration', ylab='beta')
lines(Optim2$SolutionPath[,2], col='blue')
lines(Optim3$SolutionPath[,2], col='green')
abline(h=-0.1184, col='red')
```


beta
0.00
−0.10
−0.20
−0.30
0
100
200
300
400
500
iteration