ps3_solutions_template

Question 1

preliminaries

```
# clear work space
rm(list = ls())
#set random seed
set.seed(123)
```

build model objects

```
# generate mu and sigma grids
nGridPoints = 200
muGridMin = 0
muGridMax = 20
sigGridMin = 0
sigGridMax = 10
muGrid = seq(muGridMin, muGridMax,length.out = nGridPoints)
sigGrid = seq(sigGridMin, sigGridMax,length.out = nGridPoints)
muGridSize = (muGridMax - muGridMin) / nGridPoints
sigGridSize = (sigGridMax - sigGridMin) / nGridPoints

prior = (1/20) * (1/10)
```

step 1: compute join posterior

```
# compute posterior
computePost = function(data){
  #initialize posterior matrix
  postM = matrix(rep(-1, nGridPoints ^ 2 ),
                 nrow = nGridPoints,
                 ncol = nGridPoints,
                 byrow = TRUE)
  #fill out the posterior matrix
  for (row in 1:nGridPoints) {
    for (col in 1:nGridPoints) {
      muVal = muGrid[row]
      sigVal = sigGrid[col]
      #compute data likelihood
      loglike = sum(log(dnorm(data, muVal, sigVal)))
      # update posterior matrix cell
      postM[row,col] =  exp(loglike) * prior
```

```
    }
  }
  # normalize the posterior & return
  postM = postM / sum(postM * muGridSize * sigGridSize)
  return(postM)
}

data = read.csv("~/Desktop/data_task_duration_difficulty.csv")
dataDur = data$duration[1:65]

postDur = computePost(dataDur)
```

step 2: compute marginal posterior distributions

```
#compute marginal distributions
muMarg = numeric(nGridPoints)
sigMarg = numeric(nGridPoints)
for (i in 1:nGridPoints){
  muMarg[i] = sum(postDur[i,] * sigGridSize)
  sigMarg[i] = sum(postDur[,i] * muGridSize)
}

#check normalization
muMargNorm = sum(muMarg) * muGridSize
print(muMargNorm)
```

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

```
sigMargNorm = sum(sigMarg) * sigGridSize
print(sigMargNorm)
```

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

step 3: compute summary statistics of marginal posteriors

```
muMean <- 0
sigMean <- 0
muXsqrd <- 0
sigXsqrd <- 0
for (i in 1:nGridPoints) {
  muMean <- muMean + muMarg[i] * i * (muGridSize)^2
  muXsqrd <- muXsqrd + muMarg[i] * i^2 * (muGridSize)^3
  sigMean <- sigMean + sigMarg[i] * i * (sigGridSize)^2
  sigXsqrd <- sigXsqrd + sigMarg[i] * i^2 * (sigGridSize)^3
}

muSD <- sqrt(muXsqrd - muMean^2)
sigSD <- sqrt(sigXsqrd - sigMean^2)
paste("mu mean: ", muMean)
```

```
## [1] "mu mean:  7.26430615384615"
```

```r
paste("sigma mean: ", sigMean)
```

```
## [1] "sigma mean:  3.36775920404954"
```

```r
paste("mu SD: ", muSD)
```

```
## [1] "mu SD:  0.413207313890835"
```

```r
paste("sigma SD: ", sigSD)
```

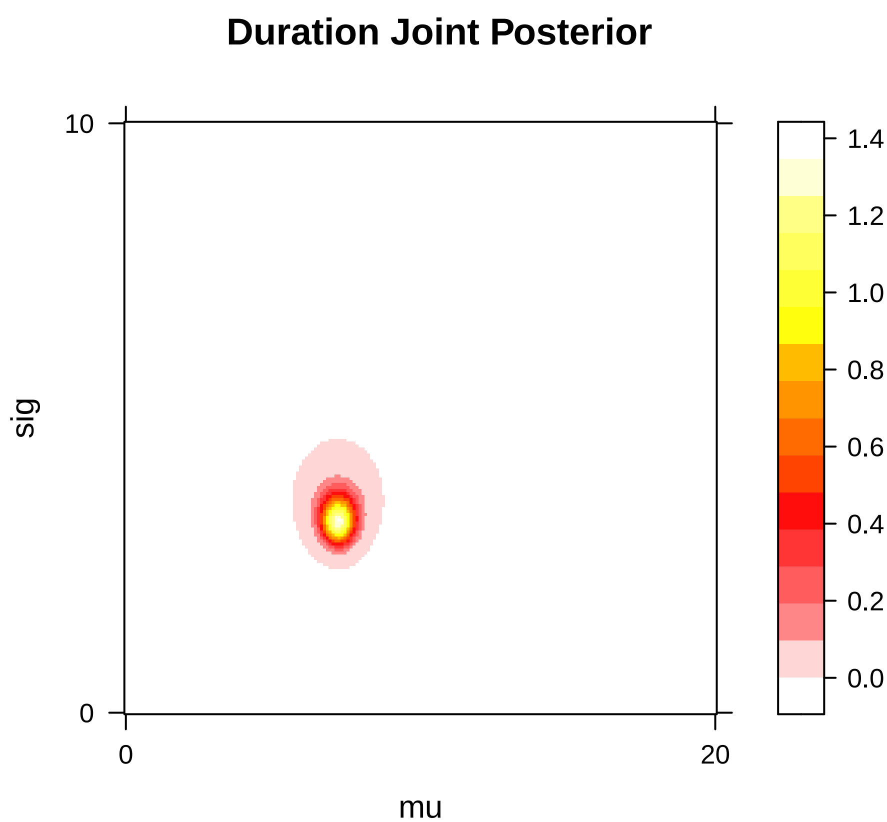
```
## [1] "sigma SD:  0.300985614904835"
```

```r
#compute covariance
cov = 0
for (i in 1:nGridPoints){
  for (j in 1: nGridPoints){
    cov = cov + postDur[i, j] * (muGrid[i] - muMean) * (sigGrid[j] - sigMean)
  }
}
paste("cov: ", cov)
```

```
## [1] "cov:  0.426586138184369"
```

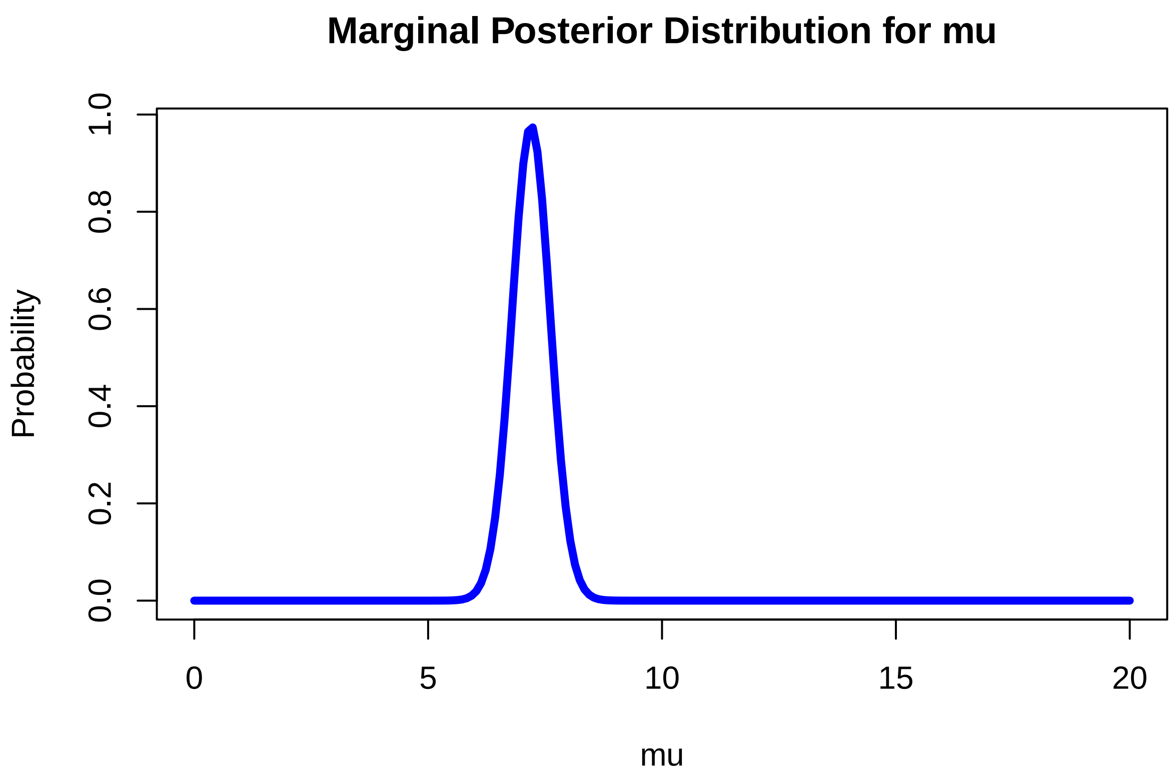
step 4: plot heat map of joint posterior

```r
# visualize joint posterior
library(lattice)
new.palette=colorRampPalette(c("white","red","yellow","white"),space="rgb")
levelplot(postDur, col.regions=new.palette(20),
          xlab = "mu", ylab = "sig",
          main = "Duration Joint Posterior",
          scales=list(x=list(at=c(1,nGridPoints), labels=c(muGridMin,muGridMax)),
                      y=list(at=c(1,nGridPoints), labels=c(sigGridMin,sigGridMax))))
```

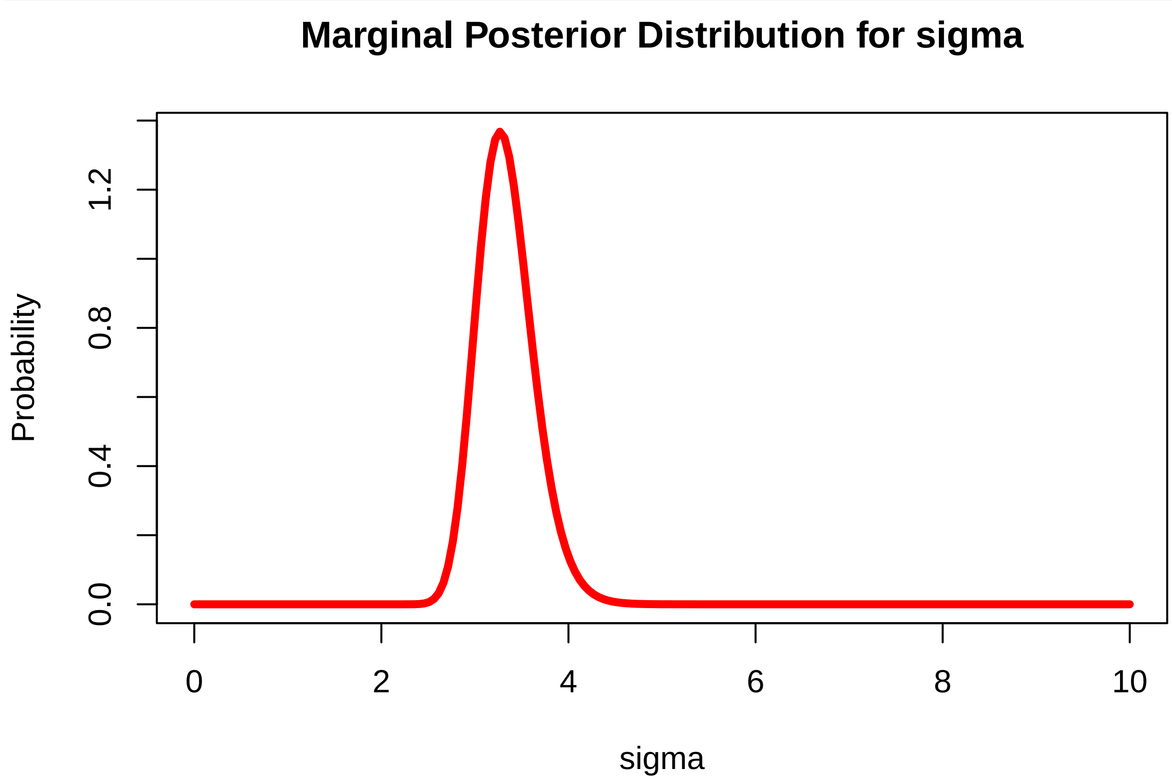


step 5: plot marginal posteriors

```
# visualize marginal posterior for mu
plot(muGrid, muMarg, type="l", lwd=4,
     xlab = "mu", ylab = "Probability",
      main = "Marginal Posterior Distribution for mu", col="blue")
```




```
# visualize marginal posterior for sigma
plot(sigGrid, sigMarg, type="l", lwd=4,
     xlab = "sigma", ylab = "Probability",
      main = "Marginal Posterior Distribution for sigma", col="red")
```



step 6: compute posterior prob mu < 5

```
# use the marginal distribution for mu to compute P(mu < 5)
prob = 0
counter = 1
while (muGrid[counter] <= 5){
  prob = prob + muMarg[counter] * muGridSize
  counter = counter + 1
}

paste(prob)
```

```
## [1] "4.3688905160503e-07"
```

Because the probability is « 1, we can conclude that Prof. Rangel's hypothesis was wrong.

Question 2

preliminaries

```
# clear work space
rm(list = ls())
#set random seed
set.seed(123)
```

build model objects

```
# define values
bGridPoints = 200
sGridPoints = 100
bGridMin = -10
bGridMax = 10
sGridMin = 0
sGridMax = 5
b0Grid = seq(bGridMin, bGridMax, length.out = bGridPoints)
b1Grid = seq(bGridMin, bGridMax, length.out = bGridPoints)
sGrid = seq(sGridMin, sGridMax, length.out = sGridPoints)
bGridSize = (bGridMax - bGridMin) / bGridPoints
sGridSize = (sGridMax - sGridMin) / sGridPoints

# import data
data = read.csv("~/Desktop/data_task_duration_difficulty_cleaned.csv")
dataDur = data$duration[1:63]
dataDif = data$difficulty[1:63]
```

Step 1: compute joint posterior

```
# compute posterior
computePost = function(x, y){
```

```r
  #initialize posterior array
  numPoints = bGridPoints ^ 2 * sGridPoints
  print(numPoints)
  postArr = array(rep(0, numPoints),
                  dim = c(sGridPoints, bGridPoints, bGridPoints))

  #fill out the posterior array
  for (i in 1:sGridPoints) {
    for (j in 1:bGridPoints) {
      for (k in 1:bGridPoints){
        sigVal = sGrid[i]
        #print(sigVal)
        b0Val = b0Grid[j]
        #print(b0Val)
        b1Val = b1Grid[k]
        #print(b1Val)
        #compute data likelihood
        #loglike = 0
        like = 1
        #print(like)
        for (l in 1:1){
          #loglike = loglike + log(dnorm(datadur[l], b0Val + b1Val * datadif[l], sigVal))
          #print(datadur[l])
          #print(datadif[l])
          #print(dnorm(datadur[l], b0Val + b1Val * datadif[l], sigVal))
          like = like * dnorm(y[l] - b0Val - (b1Val * x[l]), mean = 0, sd = sigVal)
        }
        #print(like)
        # update posterior matrix cell
        postArr[i, j, k] =  like
        #print(postArr[i,j,k])
      }
    }
  }
  # normalize the posterior & return
  postArr = postArr / sum(postArr * bGridSize^2 * sGridSize)
  return(postArr)
}

posterior = computePost(x=dataDif, y=dataDur)
```

```
## [1] 4e+06
```

```r
print(sum(posterior))
```

```
## [1] 2000
```

```r
# joint posterior: beta0-beta1
b0b1 = matrix(rep(0, bGridPoints^2),
              nrow = bGridPoints,
              ncol = bGridPoints,
              byrow=TRUE)
for (i in 1:bGridPoints){
  for (j in 1:bGridPoints){
    b0b1[i,j] = sum(posterior[ , i, j]) * sGridSize
```

```
  }
}

# joint posterior: beta1-sigma
b1s = matrix(rep(0, bGridPoints * sGridPoints),
              nrow = bGridPoints,
              ncol = sGridPoints,
              byrow=TRUE)
for (i in 1:bGridPoints){
  for (j in 1:sGridPoints){
    b1s[i,j] = sum(posterior[j, ,i]) * bGridSize
  }
}

# joint posterior: beta0-sigma
b0s = matrix(rep(0, bGridPoints * sGridPoints),
              nrow = bGridPoints,
              ncol = sGridPoints,
              byrow=TRUE)
for (i in 1:bGridPoints){
  for (j in 1:sGridPoints){
    b0s[i,j] = sum(posterior[j, i, ]) * bGridSize
  }
}
```

Step 2: compute marginal posteriors

```
# compute marginal posteriors by summing along an axis of the 2D joint posteriors
b0MargPost = rep(0, bGridPoints)
b1MargPost = rep(0, bGridPoints)
sigMargPost = rep(0, sGridPoints)

for (i in 1:bGridPoints){
  b0MargPost[i] = sum(b0b1[i,]) * bGridSize
  b1MargPost[i] = sum(b0b1[,i]) * bGridSize
}

for (i in 1:sGridPoints){
  sigMargPost[i] = sum(b1s[,i]) * bGridSize
}
```

Step 3: compute summary statistics of marginal posteriors

```
# compute summary statistics
b0Mean <- 0
b1Mean <- 0
sigMean <- 0
b0Xsqrd <- 0
b1Xsqrd <- 0
sigXsqrd <- 0
```

```r
for (i in 1:bGridPoints) {
  b0Mean <- b0Mean + b0MargPost[i] * b0Grid[i] * (bGridSize)
  b0Xsqrd <- b0Xsqrd + b0MargPost[i] * b0Grid[i]^2 * (bGridSize)
  b1Mean <- b1Mean + b1MargPost[i] * b1Grid[i] * (bGridSize)
  b1Xsqrd <- b1Xsqrd +b1MargPost[i] * b1Grid[i]^2 * (bGridSize)
}

for (i in 1:sGridPoints) {
  sigMean <- sigMean + sigMargPost[i] * sGrid[i] * (sGridSize)
  sigXsqrd <- sigXsqrd + sigMargPost[i] * sGrid[i]^2 * (sGridSize)
}

b0SD <- sqrt(b0Xsqrd - b0Mean^2)
b1SD <- sqrt(b1Xsqrd - b1Mean^2)
sigSD <- sqrt(sigXsqrd - sigMean^2)
paste("b0 mean: ", b0Mean)
```

```
## [1] "b0 mean:  0.00075110037486889"
```

```r
paste("b1 mean: ", b1Mean)
```

```
## [1] "b1 mean:  1.99978371471175"
```

```r
paste("sigma mean: ", sigMean)
```

```
## [1] "sigma mean:  2.52514364155158"
```

```r
paste("b0 SD: ", b0SD)
```

```
## [1] "b0 SD:  5.80270020537439"
```

```r
paste("b1 SD: ", b1SD)
```

```
## [1] "b1 SD:  1.8545197325985"
```

```r
paste("sigma SD: ", sigSD)
```

```
## [1] "sigma SD:  1.44336175676655"
```

```r
#compute covariance
cov = 0
for (i in 1:bGridPoints){
  for (j in 1:bGridPoints){
    cov = cov + b0b1[i, j] * (b0Grid[i] - b0Mean) * (b1Grid[j] - b1Mean) * bGridSize^2
  }
}
paste("cov: ", cov)
```

```
## [1] "cov:  -9.62037472577332"
```

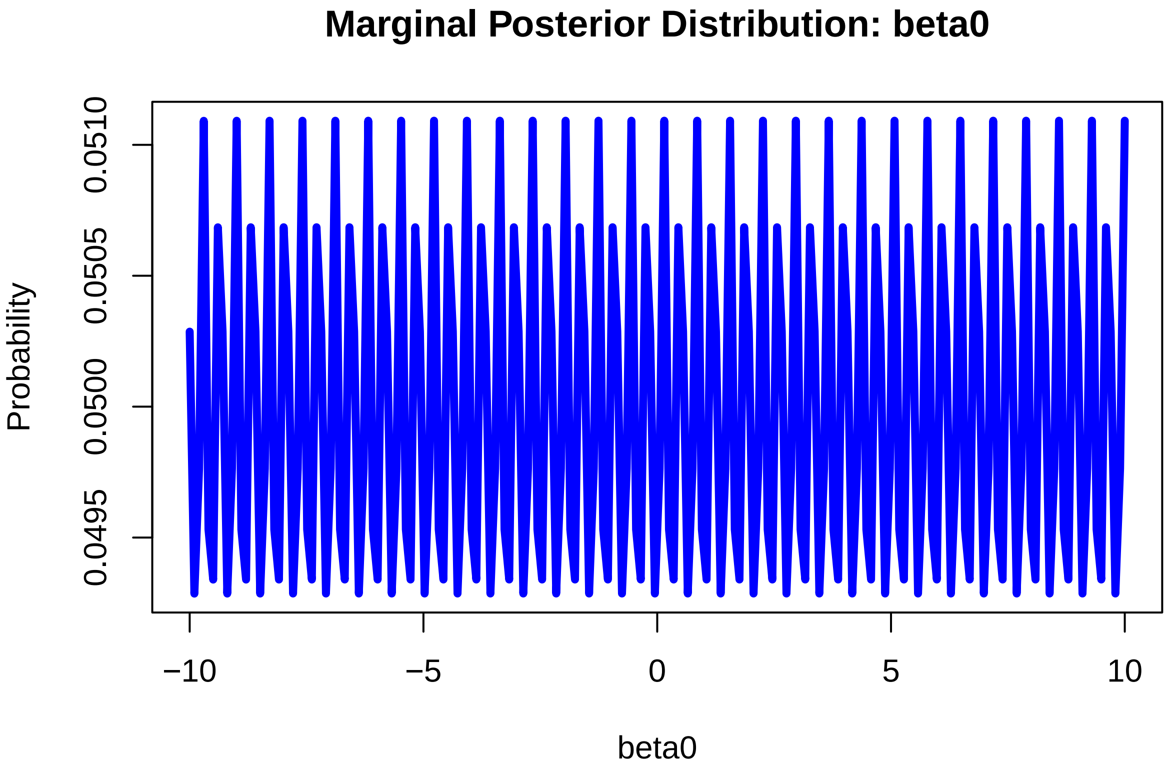
```r
print(sum(b1MargPost) * bGridSize)
```

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

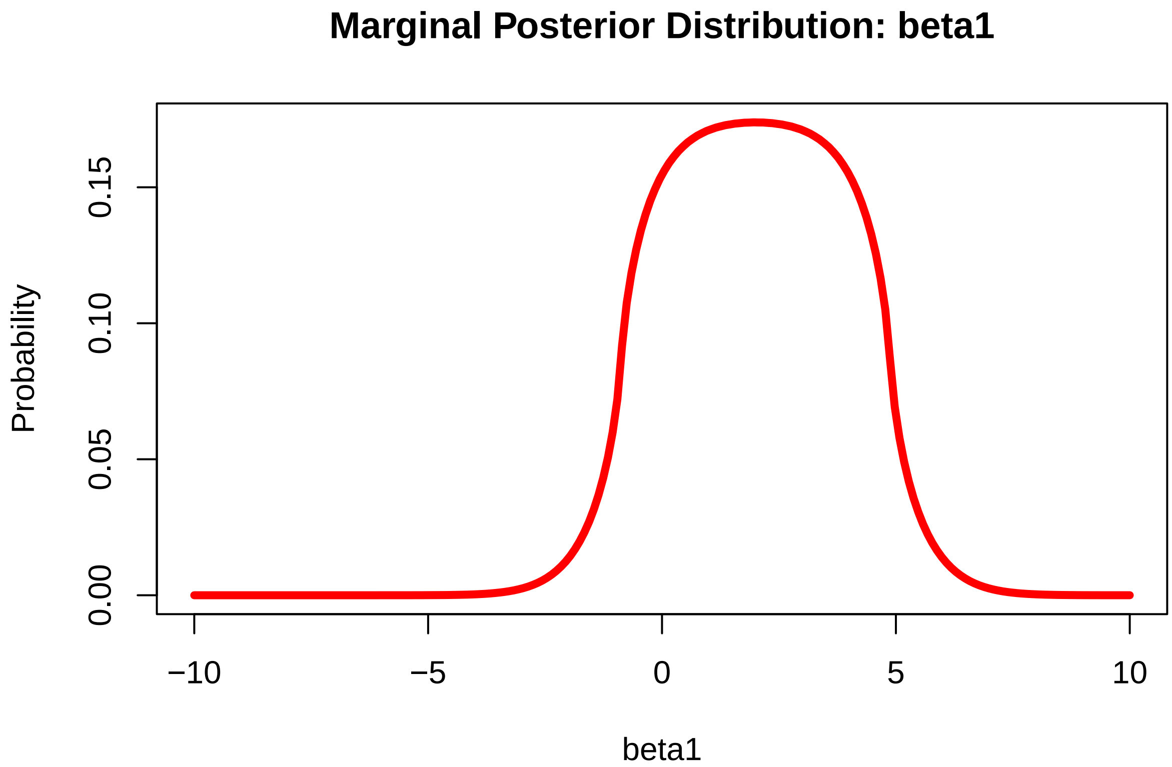
Step 4: plot marginal posterior densities

```r
# plot marginal posteriors
plot(b0Grid, b0MargPost, type="l", lwd=4,
```

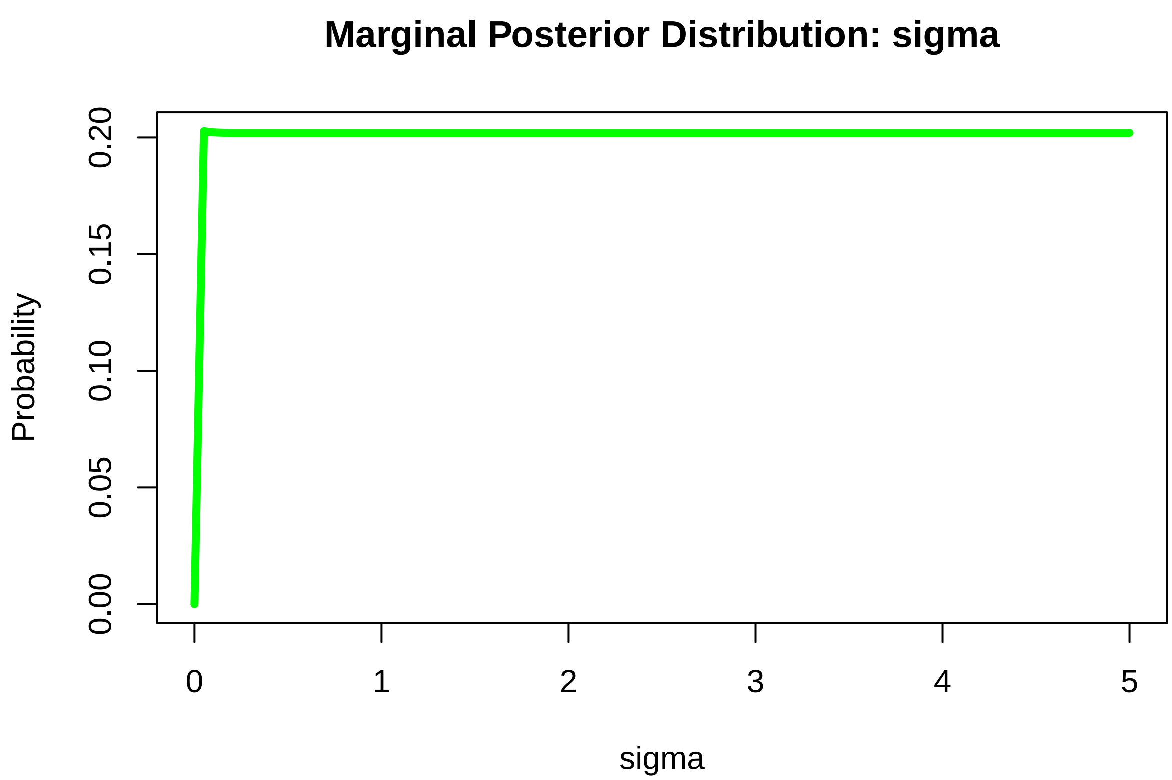
```
     xlab = "beta0", ylab = "Probability",
      main = "Marginal Posterior Distribution: beta0", col="blue")
```




```
plot(b1Grid, b1MargPost, type="l", lwd=4,
     xlab = "beta1", ylab = "Probability",
      main = "Marginal Posterior Distribution: beta1", col="red")
```

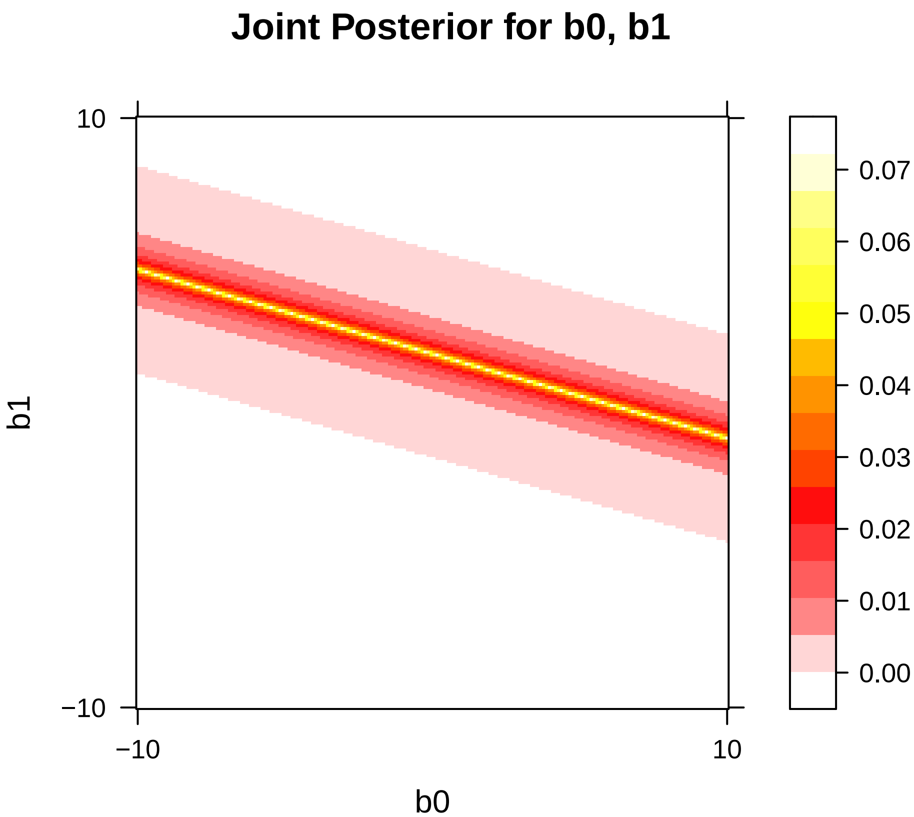



```
plot(sGrid, sigMargPost, type="l", lwd=4,
     xlab = "sigma", ylab = "Probability",
      main = "Marginal Posterior Distribution: sigma", col="green")
```

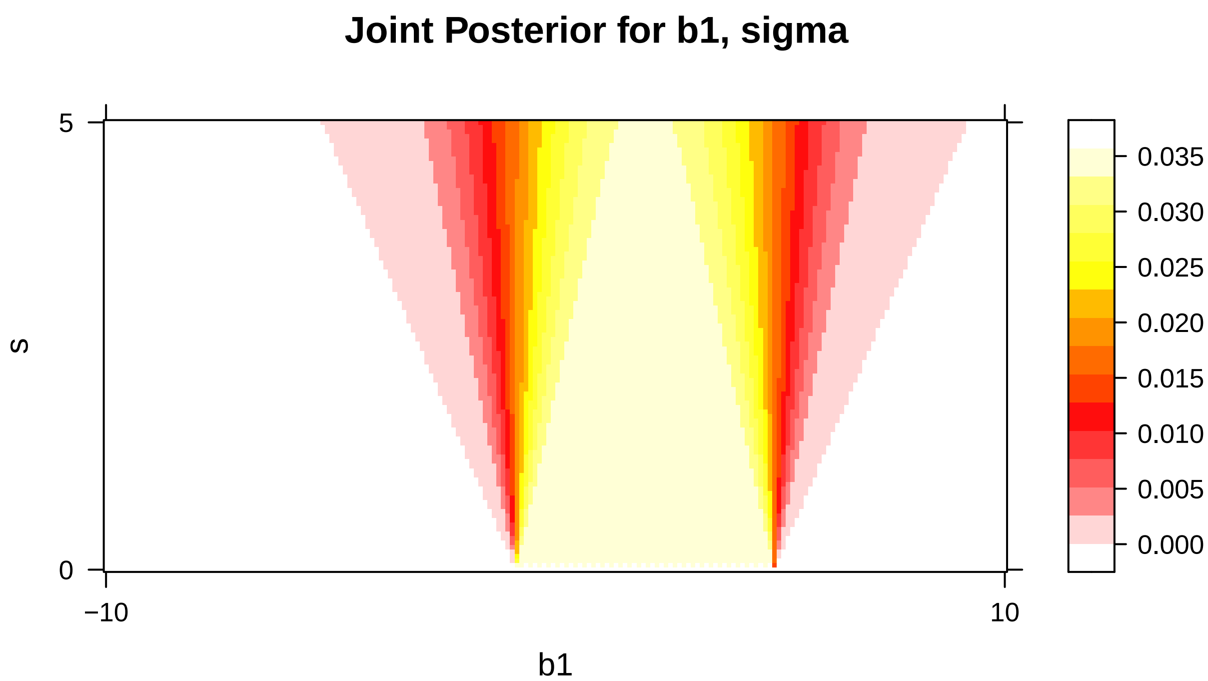


Step 5: Heat maps of joint posterior distributions

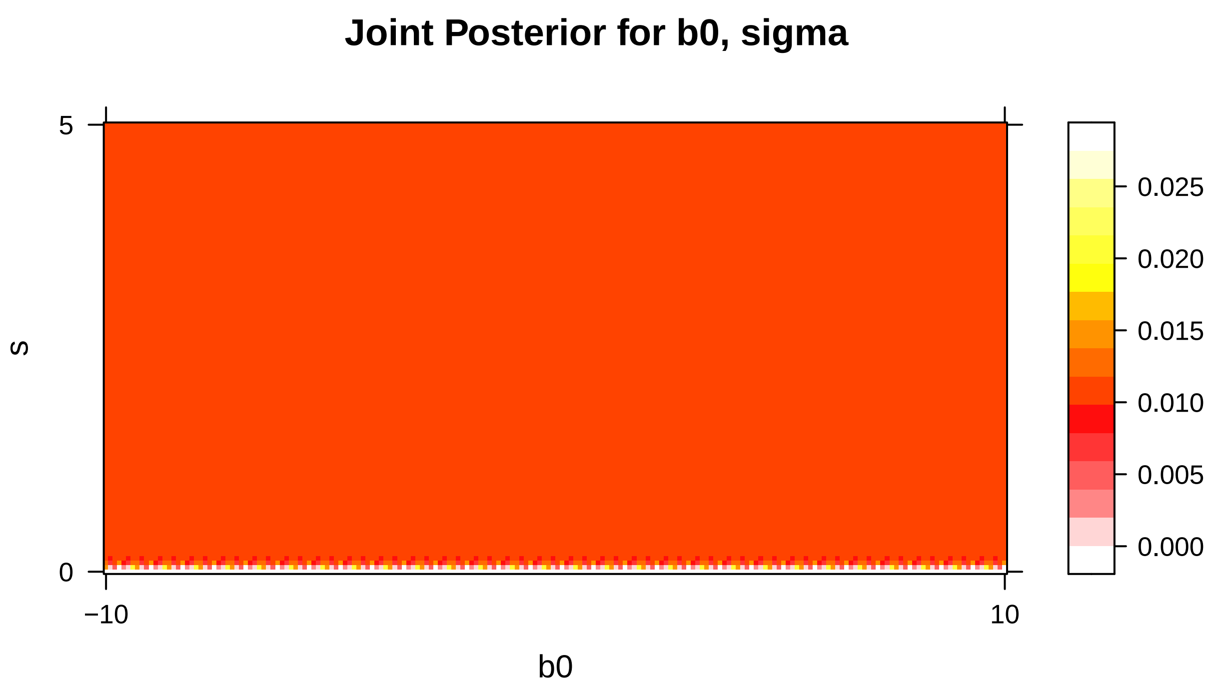
```
library(lattice)
new.palette=colorRampPalette(c("white","red","yellow","white"),space="rgb")
levelplot(b0b1, col.regions=new.palette(20),
          xlab = "b0", ylab = "b1",
          main = "Joint Posterior for b0, b1",
          scales=list(x=list(at=c(1,bGridPoints), labels=c(bGridMin,bGridMax)),
                      y=list(at=c(1,bGridPoints), labels=c(bGridMin,bGridMax))))
```




```
library(lattice)
new.palette=colorRampPalette(c("white","red","yellow","white"),space="rgb")
levelplot(b1s, col.regions=new.palette(20),
          xlab = "b1", ylab = "s",
          main = "Joint Posterior for b1, sigma",
          scales=list(x=list(at=c(1,bGridPoints), labels=c(bGridMin,bGridMax)),
                      y=list(at=c(1,sGridPoints), labels=c(sGridMin,sGridMax))))
```

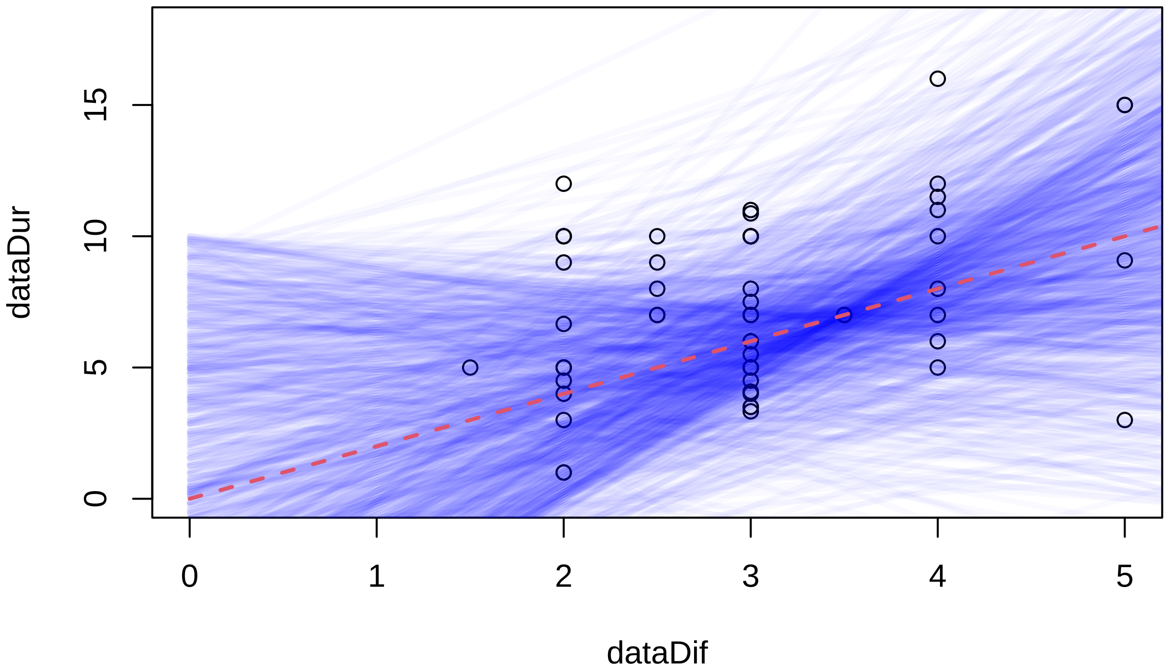



```
library(lattice)
new.palette=colorRampPalette(c("white","red","yellow","white"),space="rgb")
levelplot(b0s, col.regions=new.palette(20),
          xlab = "b0", ylab = "s",
          main = "Joint Posterior for b0, sigma",
          scales=list(x=list(at=c(1,bGridPoints), labels=c(bGridMin,bGridMax)),
                      y=list(at=c(1,sGridPoints), labels=c(sGridMin,sGridMax))))
```



Step 6: Visualize uncertainty in posterior regression lines

```
xGrid = seq(0, 18, length.out = 100)
# initialize plot
plot(dataDif,dataDur,xlim=c(0,5),ylim=c(0,18))
#plot posterior reg lines
for (sim in 1:1000) {
    b0Index = sample(1:bGridPoints, 1, prob=b0MargPost)
    b1Index = sample(1:bGridPoints, 1, prob=b0b1[b0Index,])
    b0Sample = b0Grid[b0Index]
    b1Sample = b1Grid[b1Index]
    points(xGrid, b0Sample + b1Sample*xGrid, type="l",lwd=3,
           col=rgb(red=0.0, green=0.0, blue=1.0, alpha=0.025))
}
points(xGrid, b0Mean + b1Mean*xGrid, lwd=2, col=2, lty=2, type="l")
```



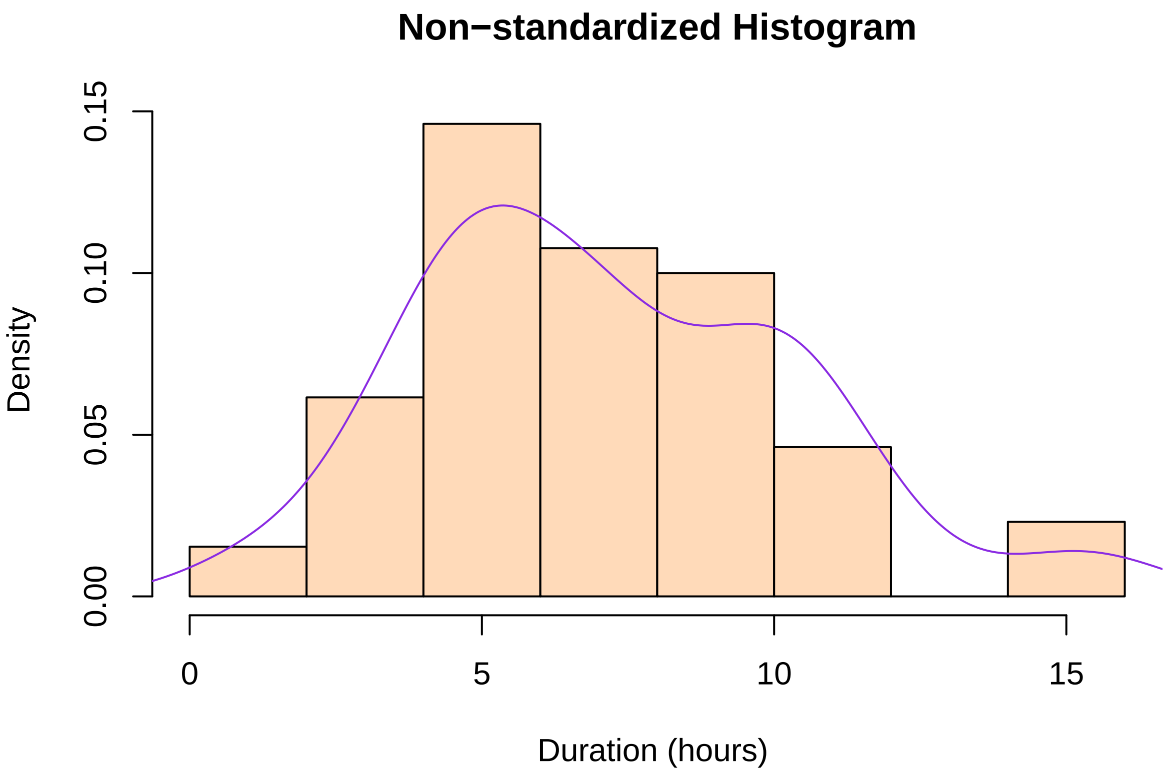
Question 3

preliminaries

```
# clear work space
rm(list = ls())
#set random seed
set.seed(123)
```

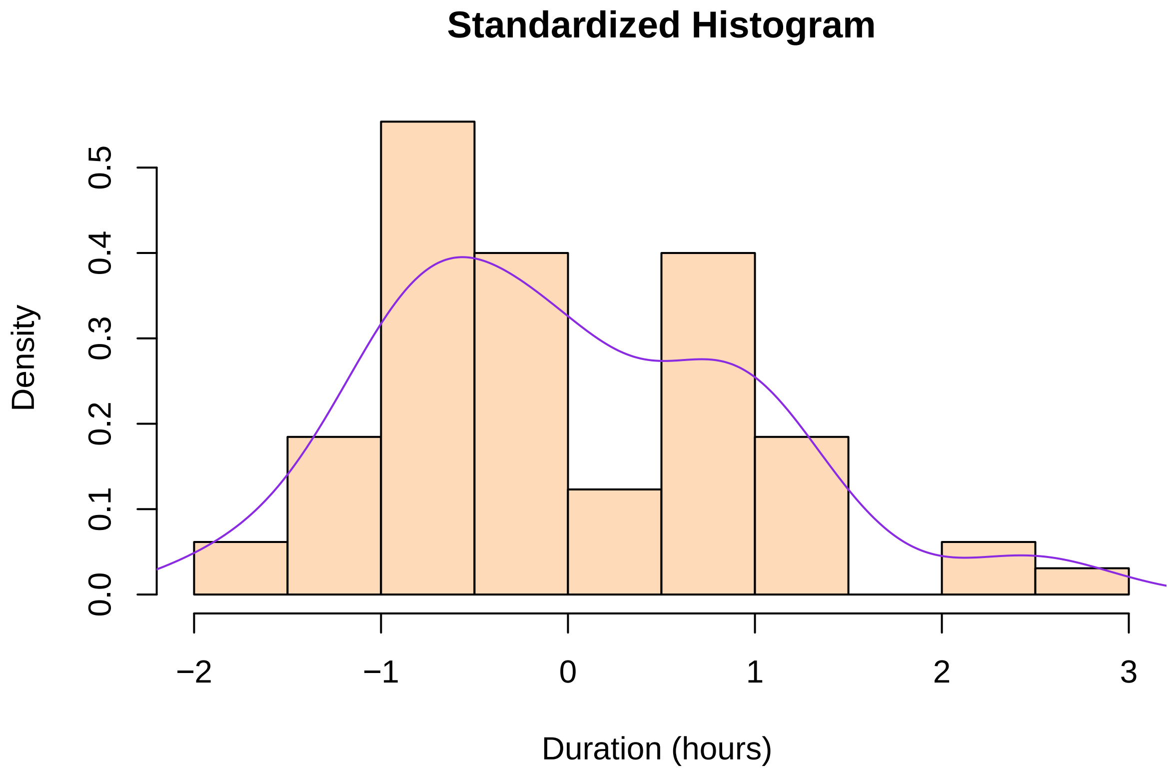
Step 1: About normality of duration data

```
# import data
data = read.csv("~/Desktop/data_task_duration_difficulty.csv")
dataDur = data$duration[1:65]

# make histogram and density line
hist(dataDur,
     breaks = 10,
     col="peachpuff",
     border="black",
     prob = TRUE, # show densities instead of frequencies
     xlab = "Duration (hours) ",
     main = "Non-standardized Histogram")
points(density(dataDur),type="l",col="blueviolet")
```

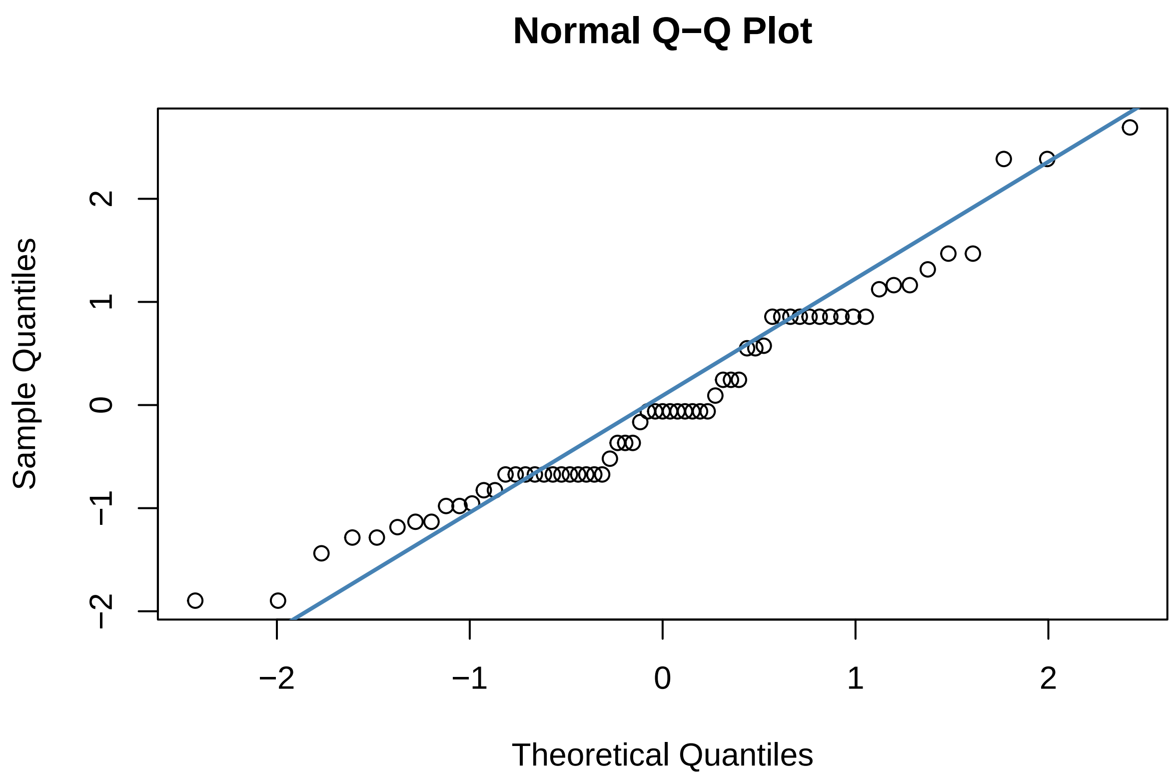



```
#standardize duration data
durMean = mean(dataDur)
durSD = sd(dataDur)
stdDur = rep(0, 65)
for (i in 1:65){
  stdDur[i] = (dataDur[i] - durMean) / durSD
}

hist(stdDur,
     breaks = 10,
     col="peachpuff",
     border="black",
     prob = TRUE, # show densities instead of frequencies
```

```
     xlab = "Duration (hours) ",
     main = "Standardized Histogram")
points(density(stdDur),type="l",col="blueviolet")
```



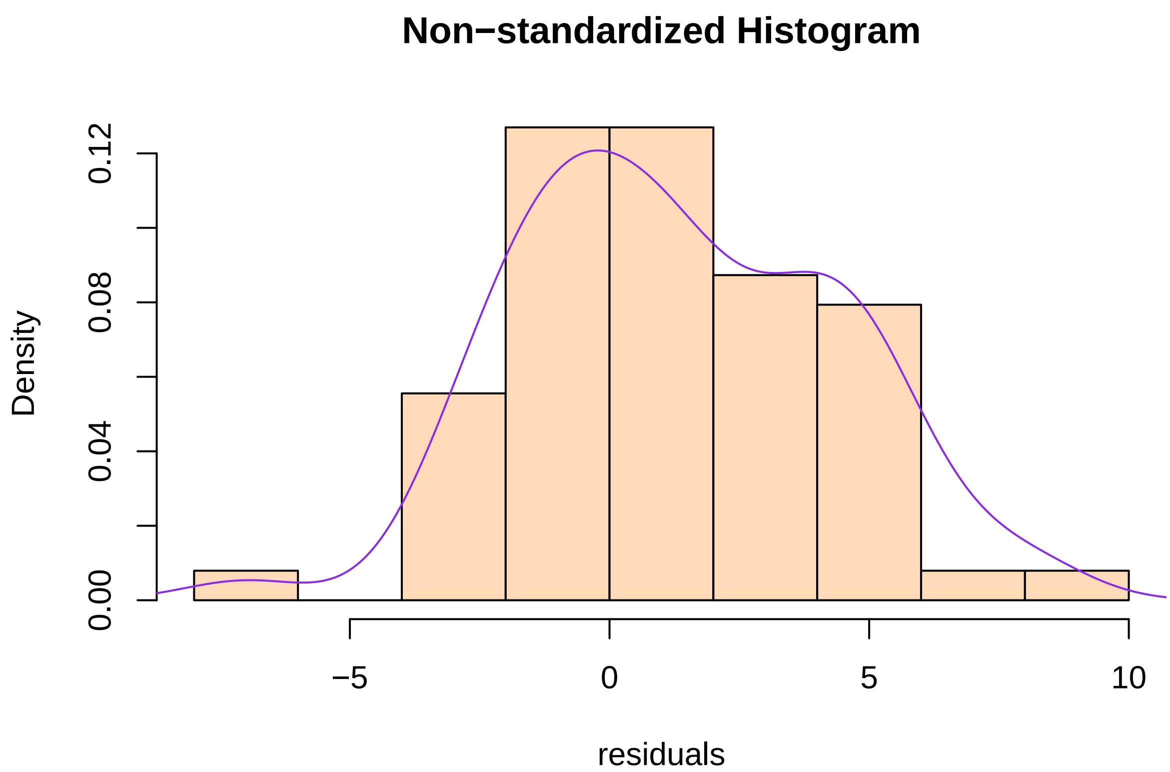

```
#make qqplot
qqnorm(stdDur, pch = 1)
qqline(stdDur, col = "steelblue", lwd = 2)
```



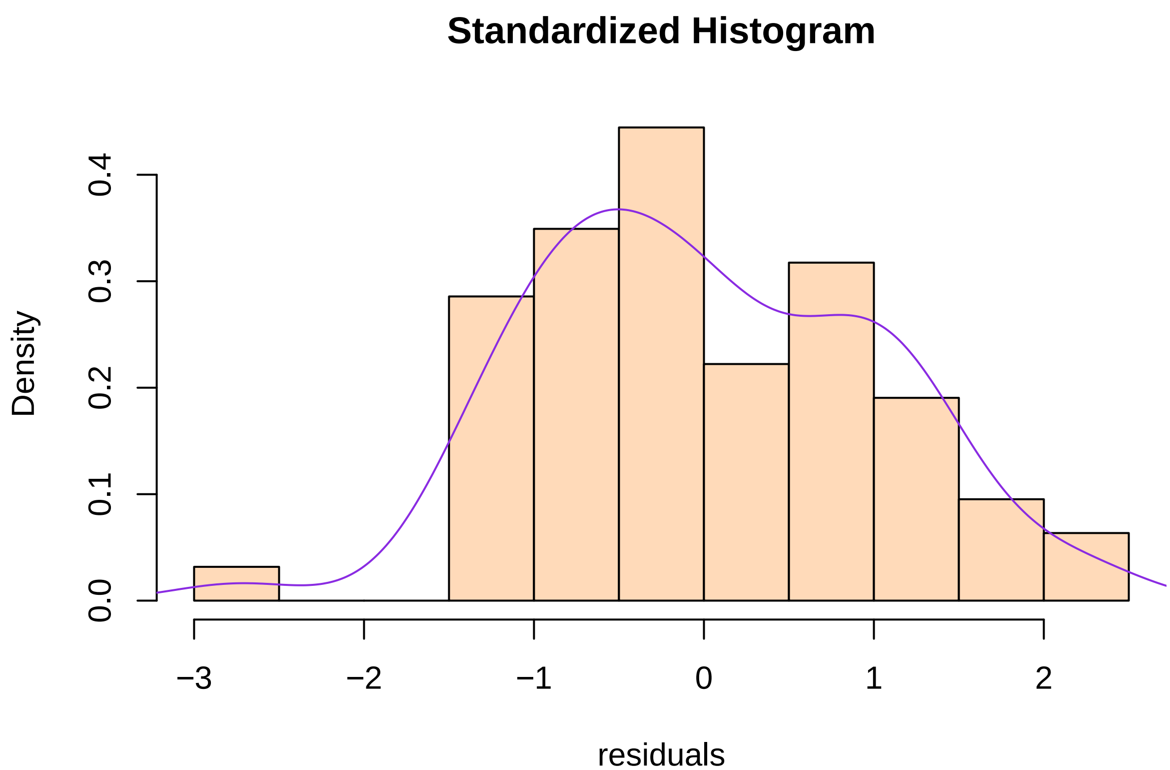
The data points lie primarily along the line in the qqplot, so we can assume that the data is approximately normally distributed.

Step 2: About normality of errors in regression model

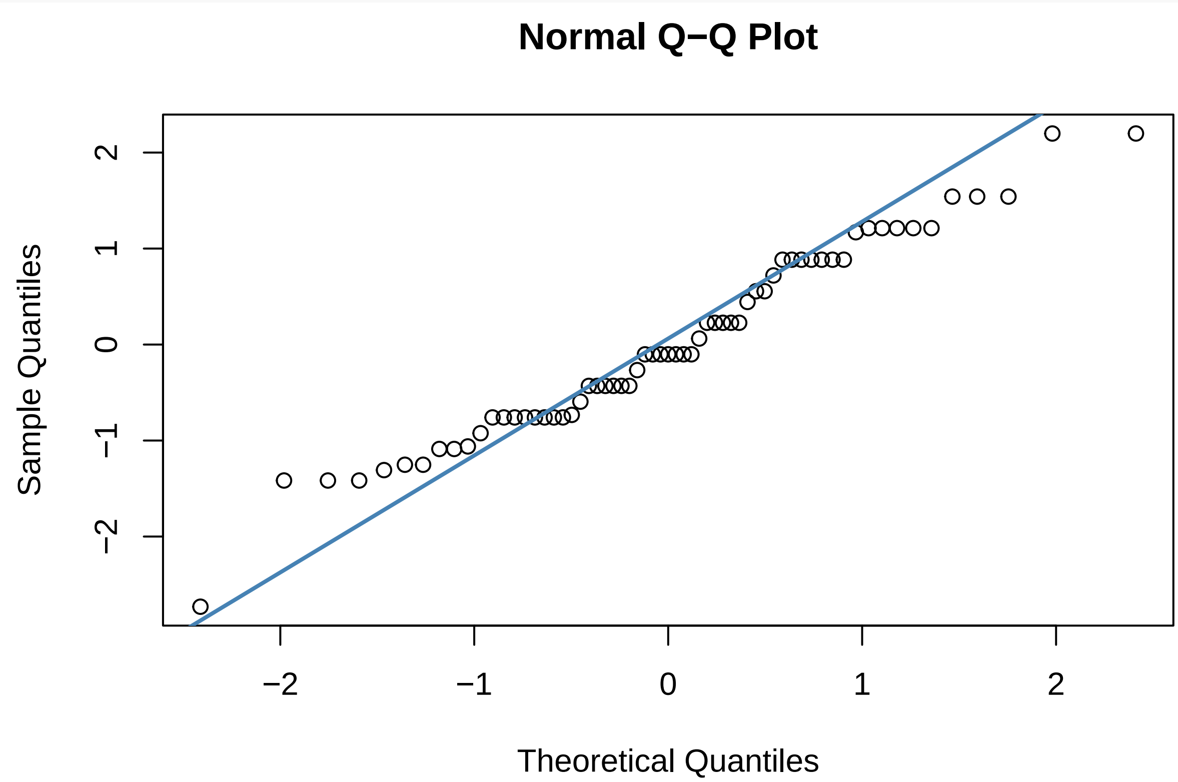
```
# import data
data = read.csv("~/Desktop/data_task_duration_difficulty_cleaned.csv")
dataDur = data$duration[1:63]
dataDif = data$difficulty[1:63]

# beta0 and beta1 means (from part 2)
b0Mean = 0.0007511
b1Mean = 1.9997837

# compute residuals
resd = rep(0, 63)
for (i in 1:63){
  resd[i] = dataDur[i] - (b0Mean + b1Mean * dataDif[i])
}

hist(resd,
     breaks = 10,
     col="peachpuff",
     border="black",
     prob = TRUE, # show densities instead of frequencies
     xlab = "residuals",
     main = "Non-standardized Histogram")
points(density(resd),type="l",col="blueviolet")
```




```r
#standardize duration data
resdMean = mean(resd)
resdSD = sd(resd)
stdResd = rep(0, 63)
for (i in 1:63){
  stdResd[i] = (resd[i] - resdMean) / resdSD
}

hist(stdResd,
     breaks = 10,
     col="peachpuff",
     border="black",
     prob = TRUE, # show densities instead of frequencies
     xlab = "residuals",
     main = "Standardized Histogram")
points(density(stdResd),type="l",col="blueviolet")
```




```
#make qqplot
qqnorm(stdResd, pch = 1)
qqline(stdResd, col = "steelblue", lwd = 2)
```



The tails of the sample quantile data pull away from the line, but, for the most part, the points lie on the line quite well. Therefore, it is safe to assume normality.