

Exercises 3

06/12/2018

Example 1. Two component normal mixture model

We consider that X is a random with *data generating process (DGP)* characterized by the *probability density function (PDF)*

$$f(x; \theta) = \pi_1 \phi(x; \mu_1, \sigma_1^2) + \pi_2 \phi(x; \mu_2, \sigma_2^2),$$

where

$$\phi(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(x - \mu)^2}{\sigma^2}\right)$$

is a normal density function with mean μ and variance σ^2 , and $\pi_1 > 0$, $\pi_2 > 0$, and

$$\pi_1 + \pi_2 = 1.$$

Here, we put all of the parameters into the vector $\theta^\top = (\pi_1, \pi_2, \mu_1, \mu_2, \sigma_1^2, \sigma_2^2)$.

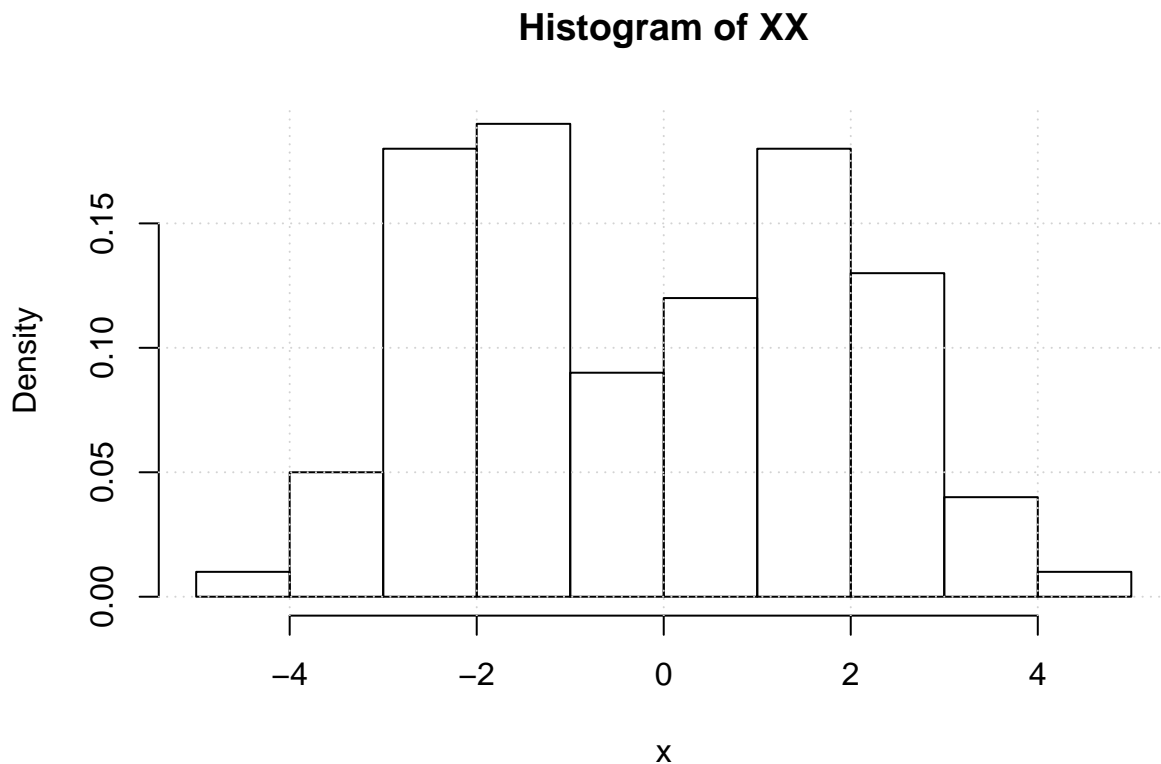
We suppose that we observe $n = 100$ *independent and identically distributed (IID)* samples X_1, \dots, X_n , from the scenario where

$$\theta^\top = (0.5, 0.5, -2, 2, 1, 1).$$

We shall firstly visualize this scenario via a histogram.

```
# Set a random seed
set.seed(123)
# Set pi
Pi <- c(0.5,0.5)
# Set mu
Mu <- c(-2,2)
# Set sigma^2
Sigma_s <- c(1,1)
# Set n
nn <- 100
# Define a function for sampling from a discrete distribution with the parameters Pi
sampler <- function(n) {
  # Get cutoff values
  cutoff <- cumsum(Pi)
  # Generate n uniform random variables
  rando <- runif(n)
  # Obtain the random sample using the cutoffs
  samp <- sapply(rando,
                 function(rr) {which(cutoff>rr)[1]}))
}
# Obtain a sample from the discrete distribution
ZZ <- sampler(nn)
# Initialize a vector to store the X values
XX <- c()
# Using the discrete sample, generate a sample from the desired mixture model
for (zz in 1:2) {
  XX <- c(XX,
          rnorm(sum(ZZ==zz),Mu[zz],sqrt(Sigma_s[zz])))
}
```

```
# Plot a histogram
hist(XX,xlab='x',probability = T)
# Add a grid
grid()
```



We will use the `mixtools` package of <https://www.jstatsoft.org/article/view/v032i06> in order to fit a normal mixture model to the data. If you have not already installed the package, run the command `install.packages('mixtools')`. Although `mixtools` is somewhat slower and older than other packages, we prefer it due to its simplicity of use and clear reporting style.

```
# Load the mixtools package
library(mixtools)
```

```
## mixtools package, version 1.1.0, Released 2017-03-10
## This package is based upon work supported by the National Science Foundation under Grant No. SES-051
```

```
# Set your initial values theta_0 at the true generative values
Pi0 <- Pi
Mu0 <- Mu
Sigma_s0 <- Sigma_s
# Use the normalmixEM function from mixtools to fit a mixture model (k is the number of components here)
mixfit <- normalmixEM(XX,
                      Pi0, Mu0, sqrt(Sigma_s0),
                      k=2)
```

```
## number of iterations= 27
```

```
# Display the parameter estimates, noting that normalmixEM uses the notation lambda for pi
summary(mixfit)
```

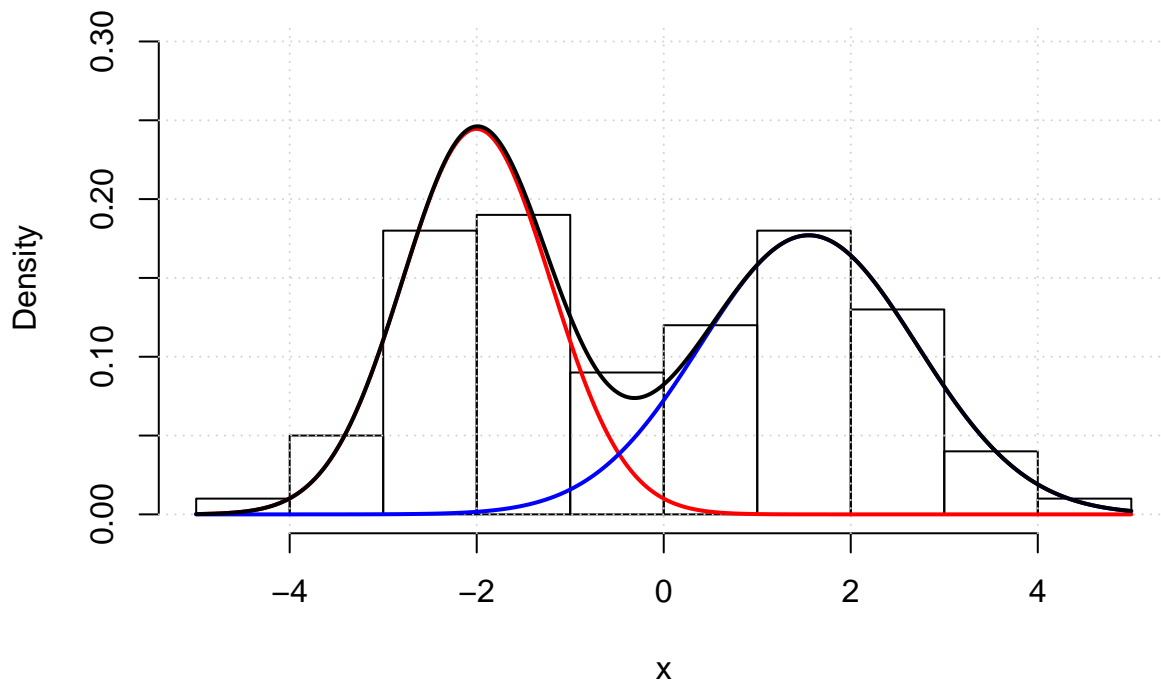
```
## summary of normalmixEM object:
##      comp 1      comp 2
```

```
## lambda  0.485348 0.514652
## mu      -2.002221 1.550621
## sigma   0.791839 1.159742
## loglik at estimate: -198.7077
```

We can also access the estimated parameters and utilize them to plot a graph over our histogram data.

```
# Access fitted values
Pi_hat <- mixfit$lambda
Mu_hat <- mixfit$mu
Sigma_s_hat <- mixfit$sigma^2
# Plot the histogram again
hist(XX,xlab='x',probability = T,ylim=c(0,0.3))
# Add a grid
grid()
# Make a dummy variable for the plot device
dum <- seq(-5,5,length.out = 1000)
# Plot the component densities in red and blue
lines(dum,
      Pi_hat[1]*dnorm(dum,Mu_hat[1],sqrt(Sigma_s_hat[1])),
      col='red',lwd=2)
lines(dum,
      Pi_hat[2]*dnorm(dum,Mu_hat[2],sqrt(Sigma_s_hat[2])),
      col='blue',lwd=2)
# Plot the overvall mixture model in black
lines(dum,
      Pi_hat[1]*dnorm(dum,Mu_hat[1],sqrt(Sigma_s_hat[1]))+
      Pi_hat[2]*dnorm(dum,Mu_hat[2],sqrt(Sigma_s_hat[2])),
      lwd=2)
```

Histogram of XX



Exercise 1. A three component mixture

Consider instead IID data X_1, \dots, X_n , $n = 100$ using `set.seed(321)`, from the normal mixture model

$$f(x; \theta) = \pi_1 \phi(x; \mu_1, \sigma_1^2) + \pi_2 \phi(x; \mu_2, \sigma_2^2) + \pi_3 \phi(x; \mu_3, \sigma_3^2),$$

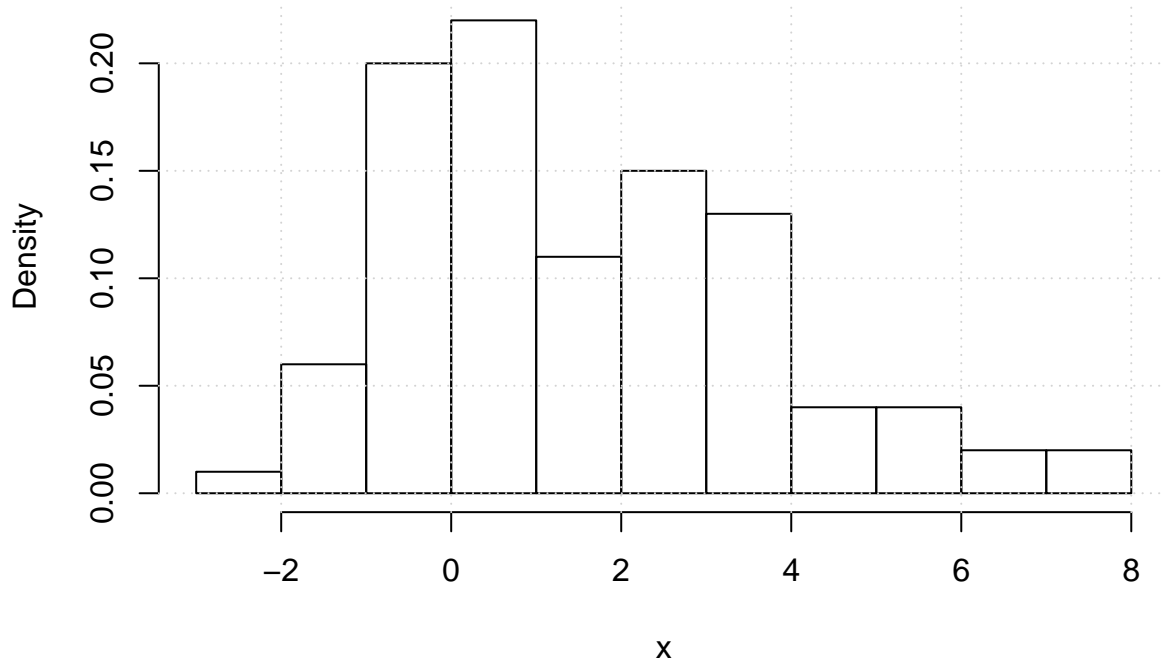
where

$$\theta^\top = (\pi_1, \pi_2, \pi_3, \mu_1, \mu_2, \mu_3, \sigma_1^2, \sigma_2^2, \sigma_3^2) = (0.5, 0.3, 0.2, 0, 2, 4, 1, 2, 3).$$

We can visualize this sample via a histogram:

```
# Set a random seed
set.seed(321)
# Set pi
Pi <- c(0.5,0.3,0.2)
# Set mu
Mu <- c(0,2,4)
# Set sigma^2
Sigma_s <- c(1,2,3)
# Set n
nn <- 100
# Define a function for sampling from a discrete distribution with the parameters Pi
sampler <- function(n) {
  # Get cutoff values
  cutoff <- cumsum(Pi)
  # Generate n uniform random variables
  rando <- runif(n)
  # Obtain the random sample using the cutoffs
  samp <- sapply(rando,
                 function(rr) {which(cutoff>rr)[1]}))
}
# Obtain a sample from the discrete distribution
ZZ <- sampler(nn)
# Initialize a vector to store the X values
XX <- c()
# Using the discrete sample, generate a sample from the desired mixture model
for (zz in 1:3) {
  XX <- c(XX,
          rnorm(sum(ZZ==zz),Mu[zz],sqrt(Sigma_s[zz])))
}
# Plot a histogram
hist(XX,xlab='x',probability = T)
# Add a grid
grid()
```

Histogram of XX



Modify the script from **Example 1** in order to fit the data using a 3-component normal mixture model, using the generative parameter vector

$$\boldsymbol{\theta}^\top = (\pi_1, \pi_2, \pi_3, \mu_1, \mu_2, \mu_3, \sigma_1^2, \sigma_2^2, \sigma_3^2)$$

as the initialization $\boldsymbol{\theta}^{(0)}$.

Use alternative initialization values in your code, and observe what you obtain. When you specify no initial values, the function will randomize the initialization for you. What can be said about the initialization's influence on the obtained solution? Do you think that the maximum likelihood problem for the mixture model has a unique global optimum?

Exercise 2. Kernel density estimation

Suppose that we observe a sample X_1, \dots, X_n from a DGP with unknown density function f . One method to estimate this density function is via a *kernel density estimator*

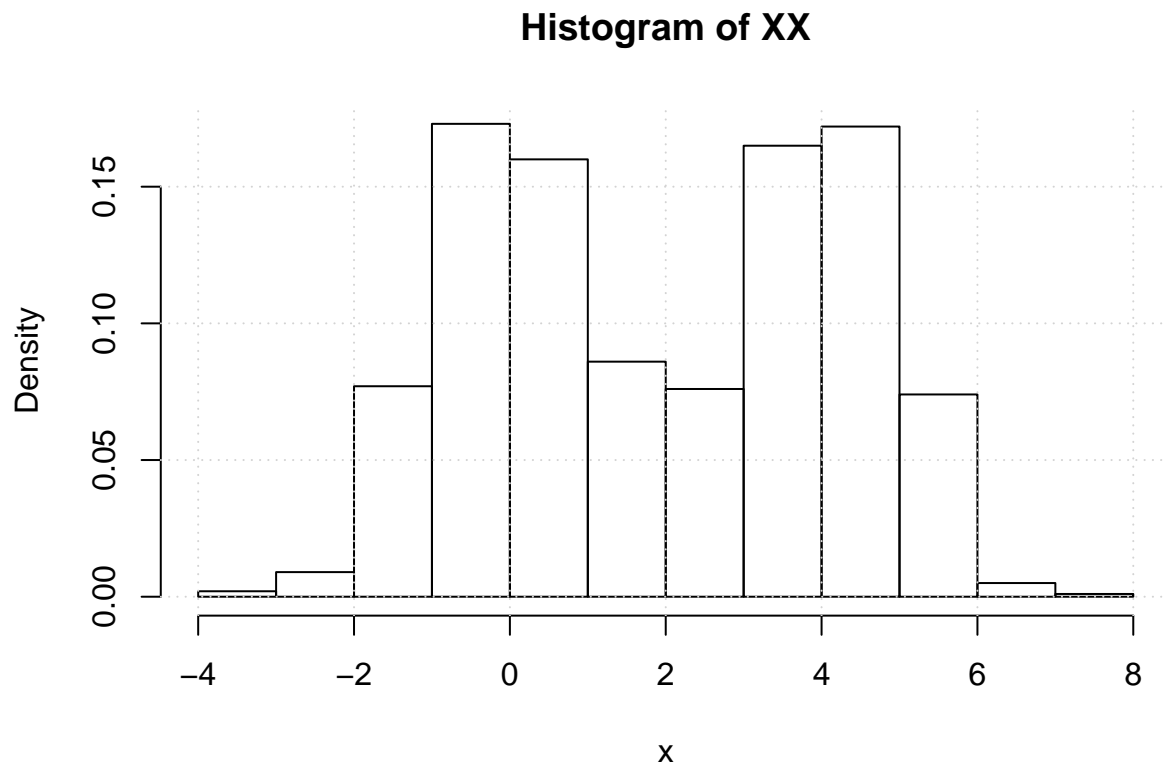
$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - X_i}{h}\right),$$

where $h > 0$ is the *bandwidth* and $K(x)$ is a so-called *kernel* function, which can be taken as a PDF for a random variable with mean *zero*.

Consider the following data:

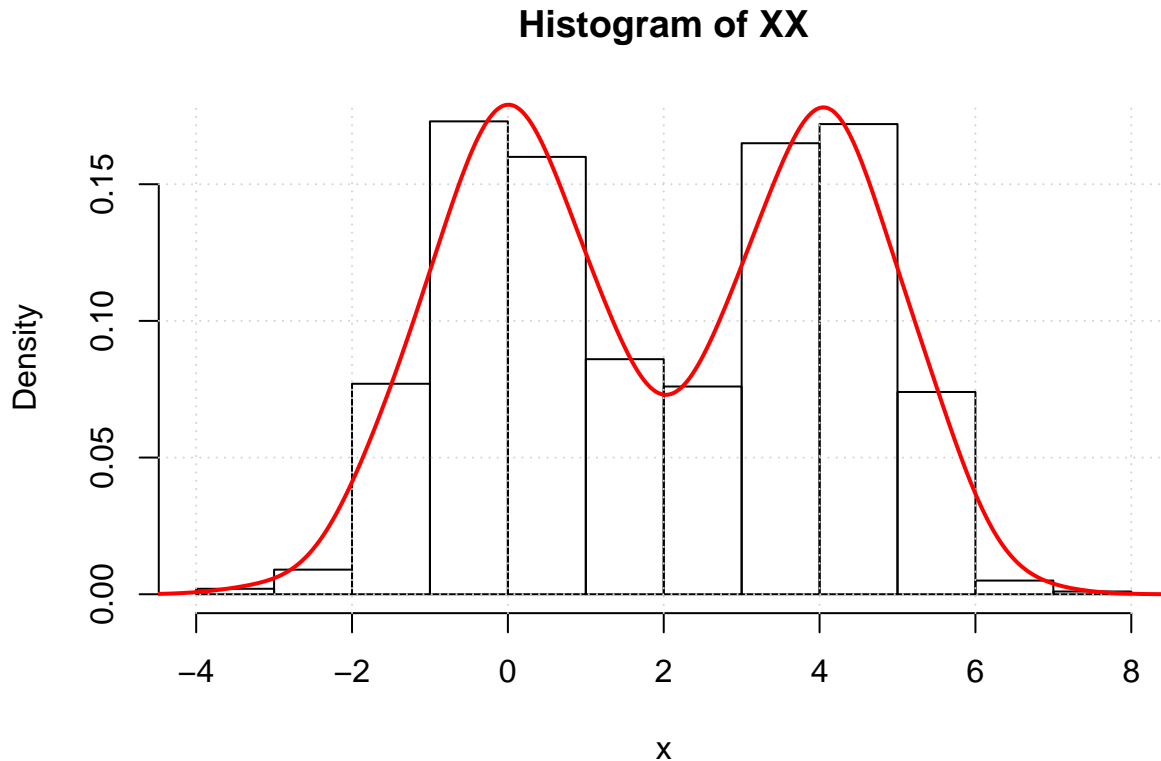
```
# Set random seed
set.seed(123)
# Set n
nn <- 1000
# Generate random data
XX <- c(rt(nn/2, df=100), 4 + rt(nn/2, df=30))
```

```
# Visualize the data via a histogram
hist(X, xlab='x', probability = T)
# Add a grid
grid()
```



Using the `density` function, we can construct kernel density estimator for the data.

```
# Construct a kernel density estimator
kde <- density(X)
# Visualize the data via a histogram
hist(X, xlab='x', probability = T)
# Add a grid
grid()
# Plot the kernel density estimator on top
lines(kde, lwd=2, col='red')
```



We notice that the kernel density estimator provides a good fit to the data.

The default settings for `density` utilizes the kernel $K(x) = \phi(x; 0, 1)$. With this in mind, what can be said regarding the relationship between kernel density estimators and normal mixture models. The `density` function allows for modification of the kernel. Try different kernels and observe any differences in the outcome.

The kernel density estimator is often used as an all-purpose density estimation tool. When can it fail?

Example 2. Multivariate mixture model

We consider the classic **iris** data, which can be loaded by the script `data(iris)`. The data contains $n = 150$ observations of $d = 4$ different measurements of the iris flower arising from $g = 3$ species of irises.

The measurements are sepal length, sepal width, petal length, and petal width. The species are *setosa*, *versicolor*, and *virginica*.

We can compute the summary statistics of the measurements of each species as follows.

```
# Load the iris data
data(iris)
# Obtain the setosa summary
summary(iris[which(iris$Species=='setosa'),1:4])
```

##	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
## Min.	:4.300	Min. :2.300	Min. :1.000	Min. :0.100
## 1st Qu.	:4.800	1st Qu.:3.200	1st Qu.:1.400	1st Qu.:0.200
## Median	:5.000	Median :3.400	Median :1.500	Median :0.200
## Mean	:5.006	Mean :3.428	Mean :1.462	Mean :0.246
## 3rd Qu.	:5.200	3rd Qu.:3.675	3rd Qu.:1.575	3rd Qu.:0.300
## Max.	:5.800	Max. :4.400	Max. :1.900	Max. :0.600

```
# Obtain the versicolor summary
summary(iris[which(iris$Species=='versicolor'),1:4])
```

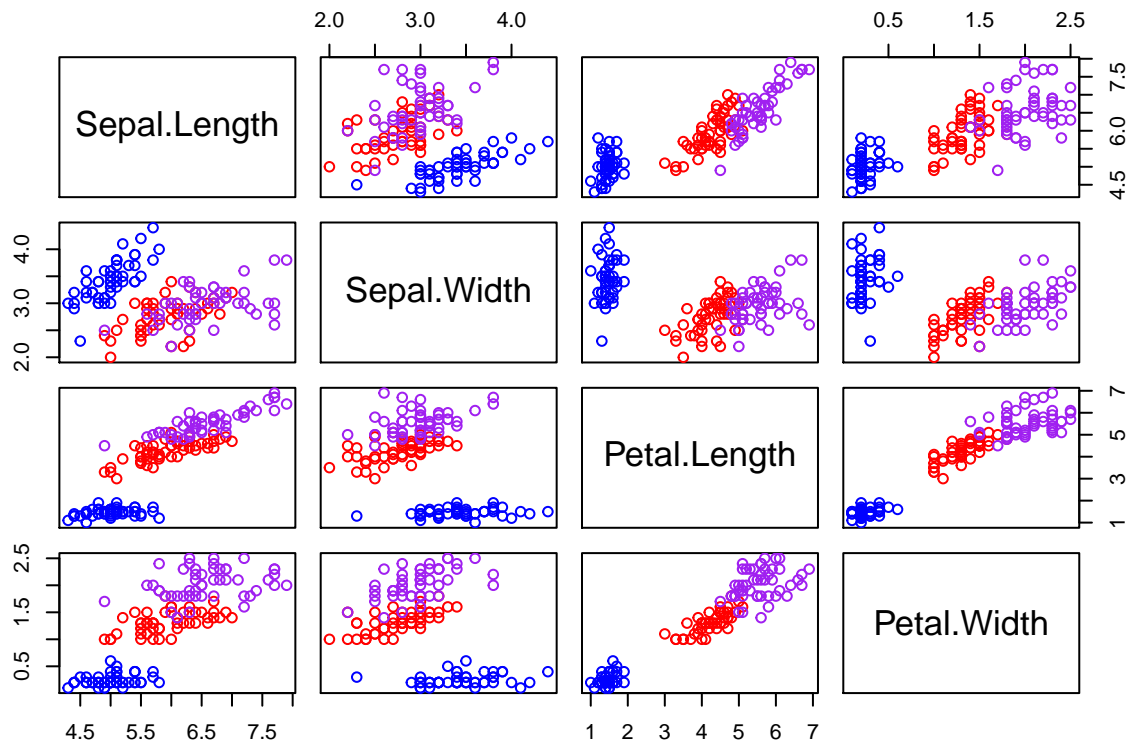
```
## Sepal.Length Sepal.Width Petal.Length Petal.Width
## Min. :4.900 Min. :2.000 Min. :3.00 Min. :1.000
## 1st Qu.:5.600 1st Qu.:2.525 1st Qu.:4.00 1st Qu.:1.200
## Median :5.900 Median :2.800 Median :4.35 Median :1.300
## Mean :5.936 Mean :2.770 Mean :4.26 Mean :1.326
## 3rd Qu.:6.300 3rd Qu.:3.000 3rd Qu.:4.60 3rd Qu.:1.500
## Max. :7.000 Max. :3.400 Max. :5.10 Max. :1.800
```

```
# Obtain the virginica summary
summary(iris[which(iris$Species=='virginica'),1:4])
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width
## Min. :4.900 Min. :2.200 Min. :4.500 Min. :1.400
## 1st Qu.:6.225 1st Qu.:2.800 1st Qu.:5.100 1st Qu.:1.800
## Median :6.500 Median :3.000 Median :5.550 Median :2.000
## Mean :6.588 Mean :2.974 Mean :5.552 Mean :2.026
## 3rd Qu.:6.900 3rd Qu.:3.175 3rd Qu.:5.875 3rd Qu.:2.300
## Max. :7.900 Max. :3.800 Max. :6.900 Max. :2.500
```

We observe that the three flower species have quite different distribution in measurements. This can further be visualized if we plot the data in a *matrix plot*.

```
# Plot the data in a matrix plot
plot(iris[,1:4],
     col=c('blue','red','purple')[as.numeric(iris[,5])])
```

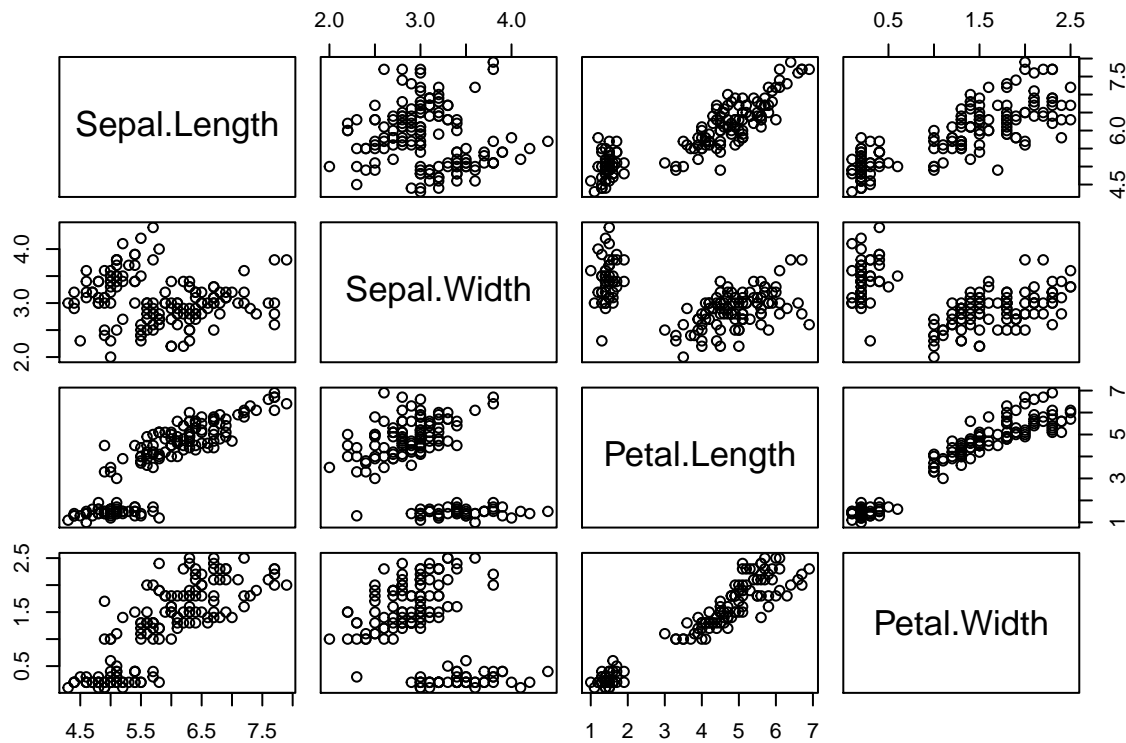


Clearly the data arises from three different populations, as we know from the data generating process that has been observed thus far.

Suppose however, that we do not know the species of each observation, but only observe the $d = 4$

measurements. Then, we would only be observing the following picture.

```
# Plot the data in a matrix plot
plot(iris[,1:4])
```



Due to its hierarchical form, we know that it may be possible to retrieve the $g = 3$ population structure from the data. We can do this by using *maximum likelihood estimation* to fit the mixture model

$$f(x; \theta) = \pi_1 \phi(x; \mu_1, \Sigma_1) + \pi_2 \phi(x; \mu_2, \Sigma_2) + \pi_3 \phi(x; \mu_3, \Sigma_3),$$

where

$$\phi(x; \mu, \Sigma) = |2\pi\Sigma|^{-1/2} \exp \left[-\frac{1}{2} (x - \mu)^\top \Sigma^{-1} (x - \mu) \right]$$

is the multivariate normal PDF with mean vector μ and covariance matrix Σ . Here $\pi_1 + \pi_2 + \pi_3 = 1$ and we put all of the elements of π_z , μ_z , and Σ_z ($z = 1, 2, 3$) into θ .

The three component mixture model above can be fitted using the `mvnormalmixEM` function from `mixtools` as follows.

```
# Load the mixtools package
library(mixtools)
# Set random seed
set.seed(111)
# Fit the mixture model using mixtools (recall that k is the number of components)
mixfit <- mvnormalmixEM(iris[, -5],
                        k=3)
```

```
## number of iterations= 200
```

```
# Obtain the estimated mixing proportions
mixfit$lambda
```

```
## [1] 0.3332880 0.2293453 0.4373667
```

```
# Obtain the estimated means
```

```
mixmapfit$mu
```

```
## [[1]]
## [1] 5.0060685 3.4281527 1.4620219 0.2459925
##
## [[2]]
## [1] 6.383977 2.992939 5.343599 2.108472
##
## [[3]]
## [1] 6.197856 2.808523 4.676159 1.449079
```

```
# Obtain the estimated covariances
```

```
mixmapfit$sigma
```

```
## [[1]]
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.12174586 0.09716792 0.01601907 0.01012907
## [2,] 0.09716792 0.14066284 0.01144081 0.00912148
## [3,] 0.01601907 0.01144081 0.02955644 0.00594996
## [4,] 0.01012907 0.00912148 0.00594996 0.01088503
##
## [[2]]
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.27404566 0.07716854 0.16163458 0.06973453
## [2,] 0.07716854 0.07340221 0.06664668 0.04269455
## [3,] 0.16163458 0.06664668 0.16793756 0.07376820
## [4,] 0.06973453 0.04269455 0.07376820 0.05847177
##
## [[3]]
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.5076934 0.13217070 0.5573041 0.17371525
## [2,] 0.1321707 0.11692922 0.1384065 0.05662739
## [3,] 0.5573041 0.13840648 0.7885673 0.24614171
## [4,] 0.1737152 0.05662739 0.2461417 0.09223767
```

Comparing the estimated mean values, does it appear as if the three different subspecies of irises have been identified? What about the estimated mixing proportions?

Using the notion of *maximum a posteriori* clustering, we can separate the n observations into *clusters*, based on which of the $g = 3$ fitted normal mixture components they are closest to. That is, for which $z = 1, 2, 3$ is the value

$$\pi_z \phi(\mathbf{x}; \boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z),$$

the largest, when we substitute the maximum likelihood estimate in for the parameter vector.

We can perform the *clustering* as follows.

```
# Perform the clustering
```

```
clustering <- apply(mixmapfit$posterior, 1, which.max)
```

```
# Print the first 6 outputs
```

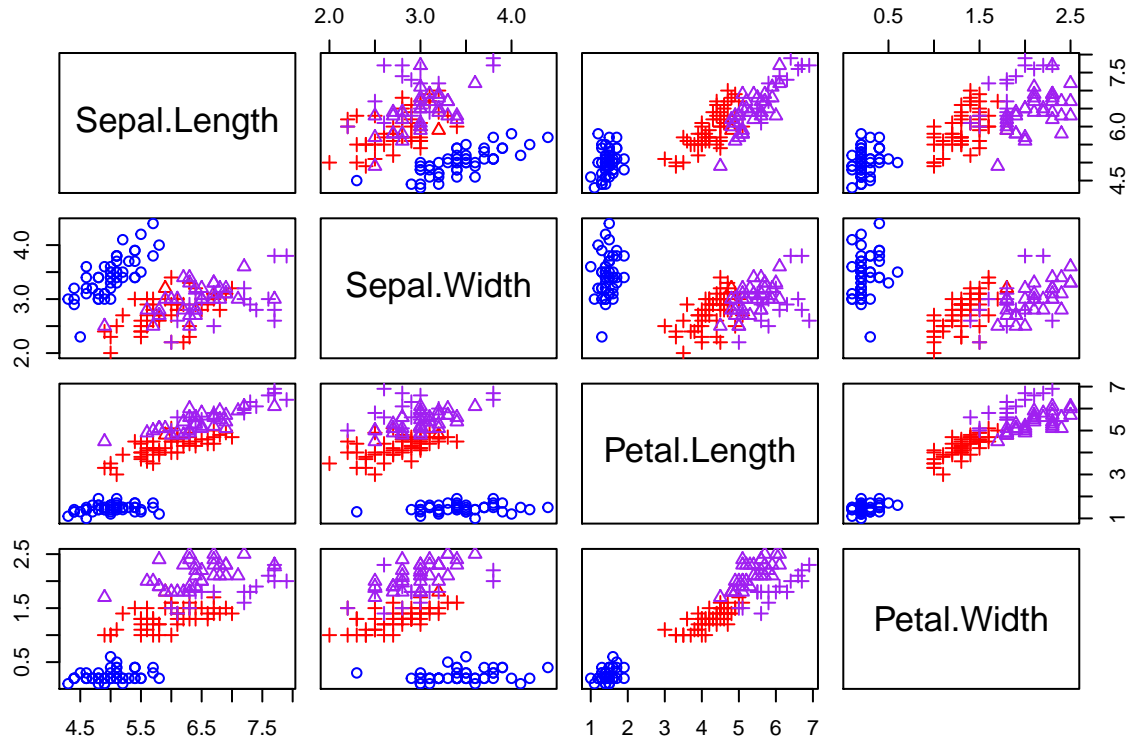
```
head(clustering)
```

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

We see that the first six outputs are all in the same cluster. Is this a promising result?

We now plot the clustering of each observation together in a different shape, along with the true species of each observation, in a different color.

```
# Plot the data in a matrix plot
plot(iris[,1:4],
     col=c('blue','red','purple')[as.numeric(iris[,5])],
     pch=clustering)
```



How successful have we been in capturing the subpopulations from this data? Given that the algorithm is initialized different each time it is run, can we conclude that there is a unique global optimal solution to the maximum likelihood estimat problem here?

Example 3. Repeated measurements

Suppose that we observe $m = 10$ measurements, at time points $k = 1, \dots, m$, each from $n = 10$ different individuals indexed by $i = 1, \dots, n$. Thus for each individual we observe m measurements

$$\mathbf{Y}_i^\top = (Y_{i1}, \dots, Y_{im}),$$

at m covariate vectors

$$\mathbf{x}_1^\top = (1, 1), \dots, \mathbf{x}_m^\top = (1, m).$$

Suppose that there is a fixed effect $\boldsymbol{\beta}^\top = (1, 1)$ that determines the overall effect of all of the individuals to the covariate vectors $\mathbf{x}_k = (1, k)$ ($k = 1, \dots, m$).

However, each individual i also has a *random effect* that is *idiosyncratic* or *specific* to that individual that makes their response different to $\boldsymbol{\beta}^\top = (\beta_1, \beta_2)$. Let that random effect be given by \mathbf{B}_i where

$$\mathbf{B}_i = (B_{i1}, B_{i2}) \sim \mathcal{N}(\mathbf{0}, \mathbf{V}).$$

Furthermore, suppose that we observe all of the observations with some additional *random noise* E_{ik} , where

$$E_{ik} \sim \mathcal{N}(0, \sigma^2).$$

Here, we let $\mathbf{V} = \mathbf{I}$ and $\sigma^2 = 1/2$.

From the setup so far, we know that observation Y_{ik} from individual i is generated via the linear model

$$Y_{ik} = (\beta + \mathbf{B}_i)^\top \mathbf{x}_{ik} + E_{ik}.$$

With the notation that \mathbf{X}_i contains the rows \mathbf{x}_{ik}^\top and

$$\mathbf{E}_i^\top = (E_{i1}, \dots, E_{im}),$$

we can write the model as

$$\mathbf{Y}_i = \mathbf{X}_i (\beta + \mathbf{B}_i) + \mathbf{E}_i.$$

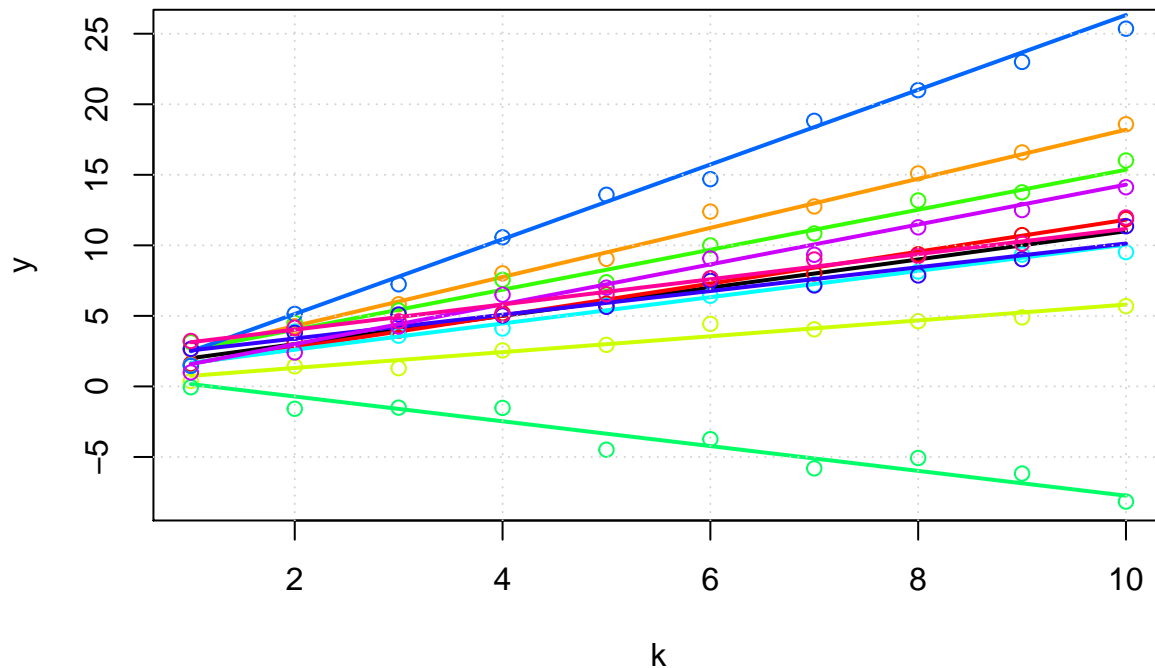
We plot the data (Y_{ik}, k) for each individual $i = 1, \dots, n$ along with the overall mean relationship

$$y = \beta_1 + \beta_2 k,$$

and each of the individual relationships, specific to individual i ,

$$y_i = \beta_1 + B_{i1} + (\beta_2 + B_{i2})k.$$

```
# Set random seed
set.seed(100)
# Set n
nn <- 10
# Set m
mm <- 10
# Create a list to store Y_i vectors
ylist <- list()
# Create a list to store B_i vectors
Blist <- list()
# Create a list to store X_i matrices
Xlist <- list()
# Use a loop to generate the data
for (ii in 1:nn) {
  # Make the X matrices
  Xlist[[ii]] <- cbind(1, 1:mm)
  # Make the B vectors
  Blist[[ii]] <- rnorm(2, 0, 1)
  # Generate the Y values
  ylist[[ii]] <- Xlist[[ii]] %*% (matrix(Blist[[ii]], 2, 1) + matrix(c(1, 1), 2, 1)) + rnorm(mm, 0, 0.5)
}
# Plot the mean function
plot(1:nn, 1+1:nn, type='l', lwd=2, xlab='k', ylab='y', ylim=c(min(unlist(ylist)), max(unlist(ylist))))
for (ii in 1:nn) {
  # Plot each set of points
  points(1:nn, ylist[[ii]], col=rainbow(nn)[ii])
  # Plot each mean line
  lines(1:nn, 1+Blist[[ii]][1] + (1+Blist[[ii]][2])*(1:nn),
        col=rainbow(nn)[ii], lwd=2, lty=1)
}
# Draw a grid
grid()
```



We can use the `lmer` function from the `lme4` library in order to estimate the parameters β , \mathbf{V} , and σ^2 , of a repeated measures model. We can install `lme4` via the command `install.packages('lme4')`.

In order to fit our model, we must first reformat our data into a form compatible with `lmer`. We can then fit the model as follows.

```
# Load lme4 package
library(lme4)

## Loading required package: Matrix

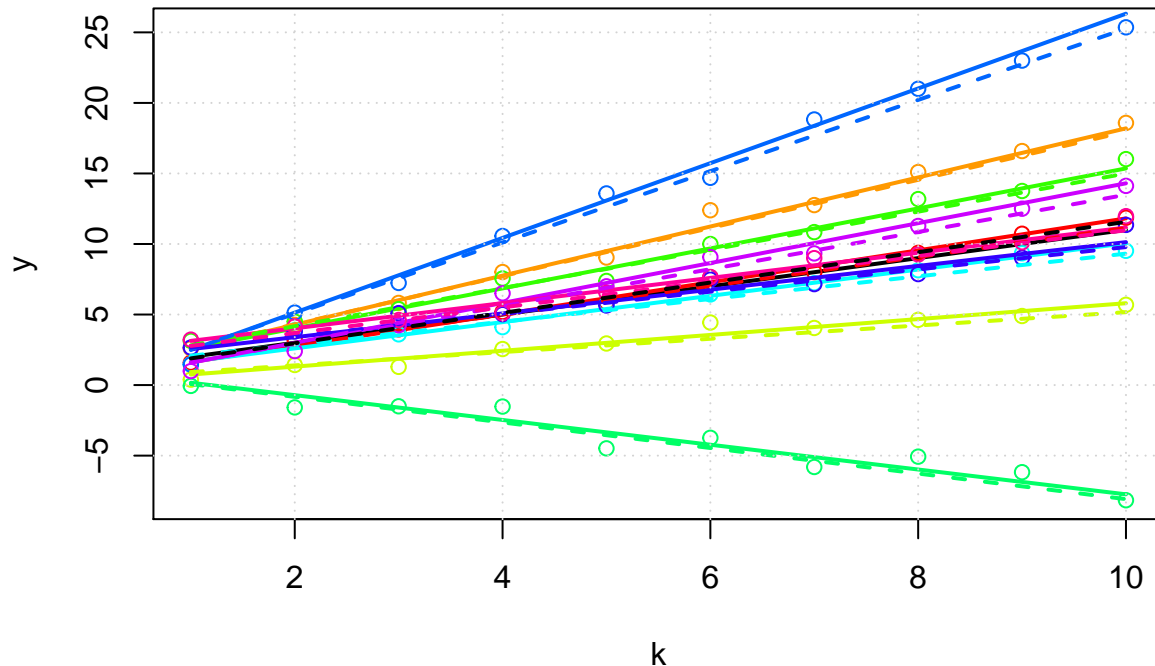
# Put data in long format with three columns for y, k, and i
data_forlmm <- as.data.frame(cbind(unlist(ylist),
                                   rep(1:10,10),
                                   unlist(lapply(1:10,rep,10))))

# Name the columns
names(data_forlmm) <- c('y','k','i')
# Fit the repeated measures model (REML=F turns on MLE)
lmm <- lmer(y~k+(1+k|i),data=data_forlmm,REML=F)
# Extract the coefficients B_i from the fitted model
coeffs <- ranef(lmm)$i
# Plot the mean function
plot(1:nn,1+1:nn,type='l',lwd=2,xlab='k',ylab='y',ylim=c(min(unlist(ylist)),max(unlist(ylist))))
for (ii in 1:nn) {
  # Plot each set of points
  points(1:nn,ylist[[ii]],col=rainbow(nn)[ii])
  # Plot each mean line
  lines(1:nn,1+Blist[[ii]][1]+(1+Blist[[ii]][2])*(1:nn),
        col=rainbow(nn)[ii],lwd=2,lty=1)
}
# Begin a loop to plot the individual lines
for (ii in 1:nn) {
  # Plot the individual lines in dashed lines
  lines(1:nn,1+coeffs[ii,1]+(1+coeffs[ii,2])*(1:nn),
```

```

        col=rainbow(nn)[ii],lwd=2,lty=2)
}
# Plot fitted mean line
lines(1:nn, fixef(lmm)[1] + fixef(lmm)[2]*(1:nn),
      lwd=2,lty=2)
# Add a grid
grid()

```



Exercise 3. Sleepstudy data

Consider the `sleepstudy` data from the `lme4` package. Load the data by using the `data(sleepstudy)` command.

The data measures the $n = 18$ subjects i that are measured at day t for their *reaction time* Y_{it} . Here, there are $t = 0, \dots, m$ where $m = 9$ is the total number of days.

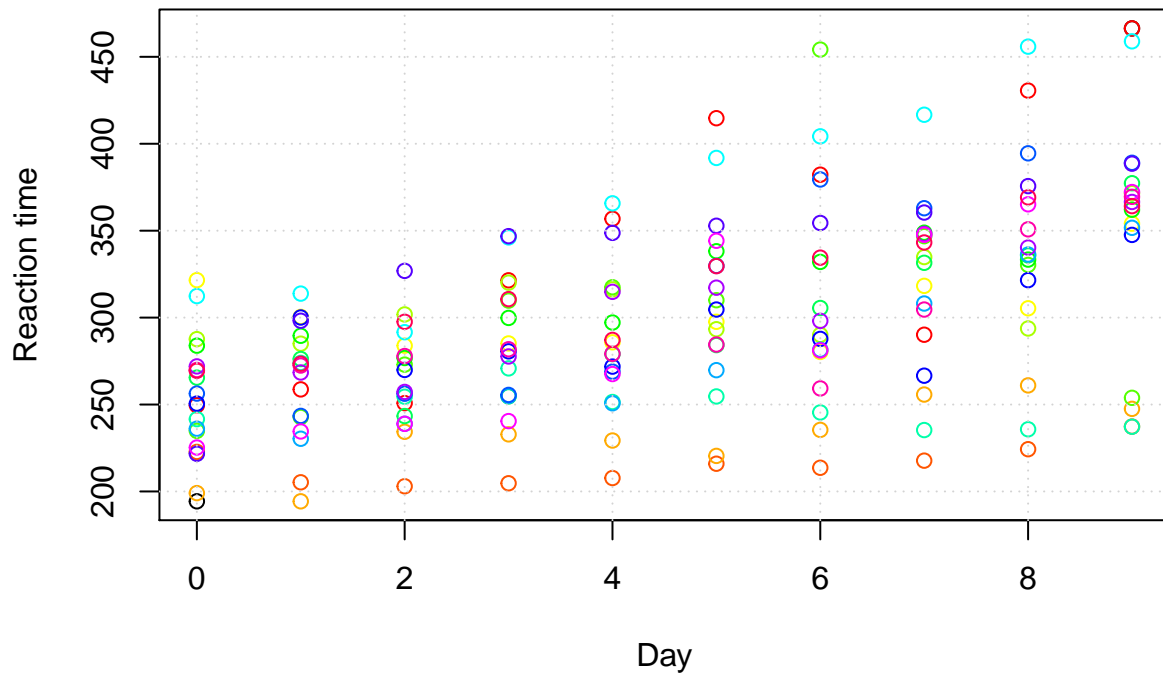
We begin by plotting the relationship between reaction time and days for each individual.

```

# Load sleepstudy data
data(sleepstudy)
# Get each of the different subject labels
unique_sub <- unique(sleepstudy$Subject)
# Construct an empty plot with appropriate size window
plot(c(0,9),c(min(sleepstudy$Reaction),max(sleepstudy$Reaction)),
     xlab='Day',ylab='Reaction time')
# For loop to plot data for each subject
for (ss in 1:18) {
  # Plot the data for individual subject
  points(0:9,
        sleepstudy$Reaction[sleepstudy$Subject==unique_sub[ss]],
        col=rainbow(18)[ss])
}

```

```
# Add a grid
grid()
```



From the data, does it appear as if all of the subjects follow the same mean relationship between reaction time and days into the experiment? Would a linear regression model be appropriate here or should we use a repeated measures model?

If you concluded that a repeated measurement model should be used, then utilize the code from **Example 3** to fit the same model. Plot the data with the fitted mean line and individual subject line obtained from the fitted object.