Lesson 11H: Mixture models

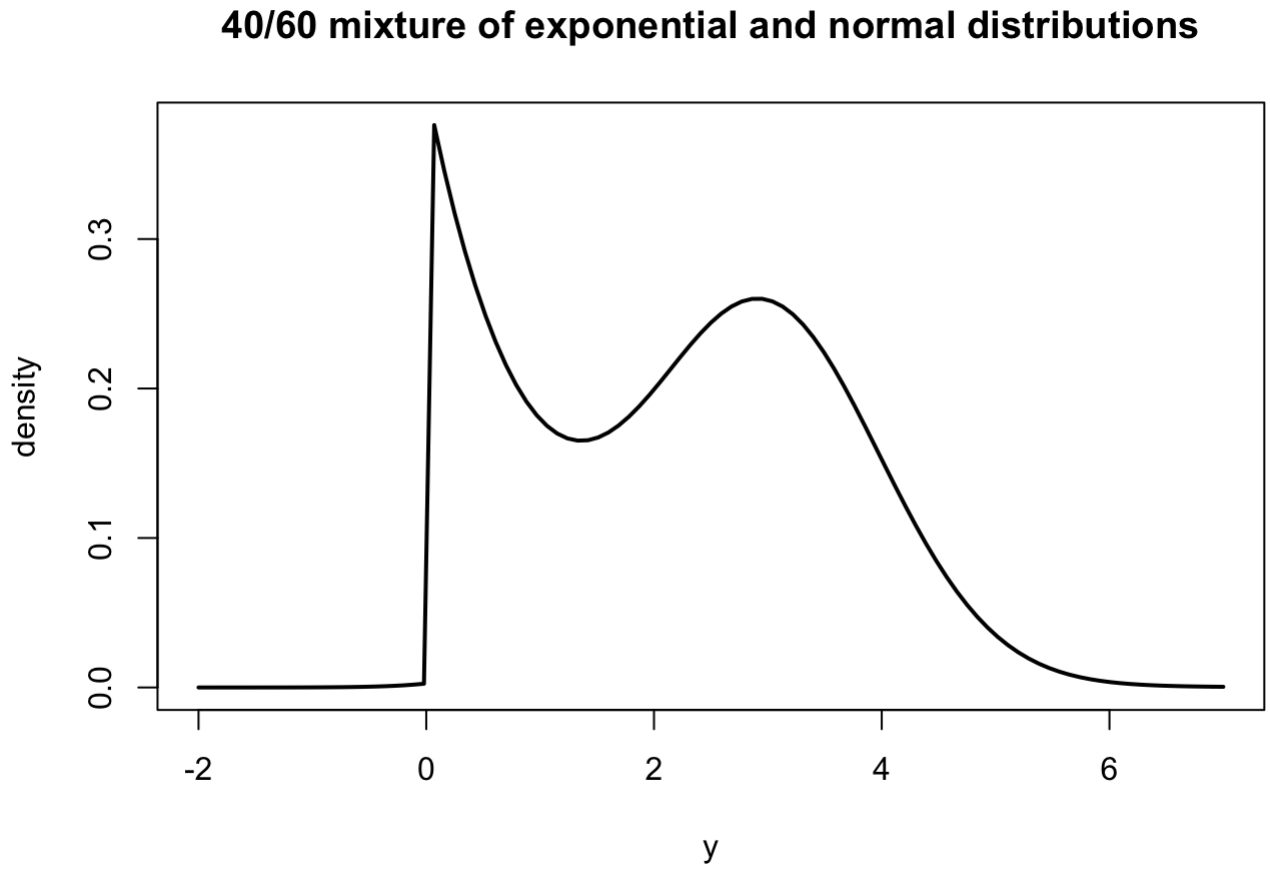
Histograms of data often reveal that they do not follow any standard probability distribution. Sometimes we have explanatory variables (or covariates) to account for the different values, and normally distributed errors are adequate, as in normal regression. However, if we only have the data values themselves and no covariates, we might have to fit a non-standard distribution to the data. One way to do this is by mixing standard distributions.

Mixture distributions are just a weighted combination of probability distribtuions. For example, we could take an exponential distribution with mean 1 and normal distribution with mean 3 and variance 1 (although typically the two mixture components would have the same support; here the exponential component has to be non-negative and the normal component can be positive or negative). Suppose we give them weights: 0.4 for the exponential distribution and 0.6 for the normal distribution. We could write the PDF for this distribution as

$$p(y) = 0.4 \cdot \exp(-y) \cdot I_{(y \geq 0)} + 0.6 \cdot \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y-3)^2 \right).$$

The PDF of this mixture distribution would look like this:

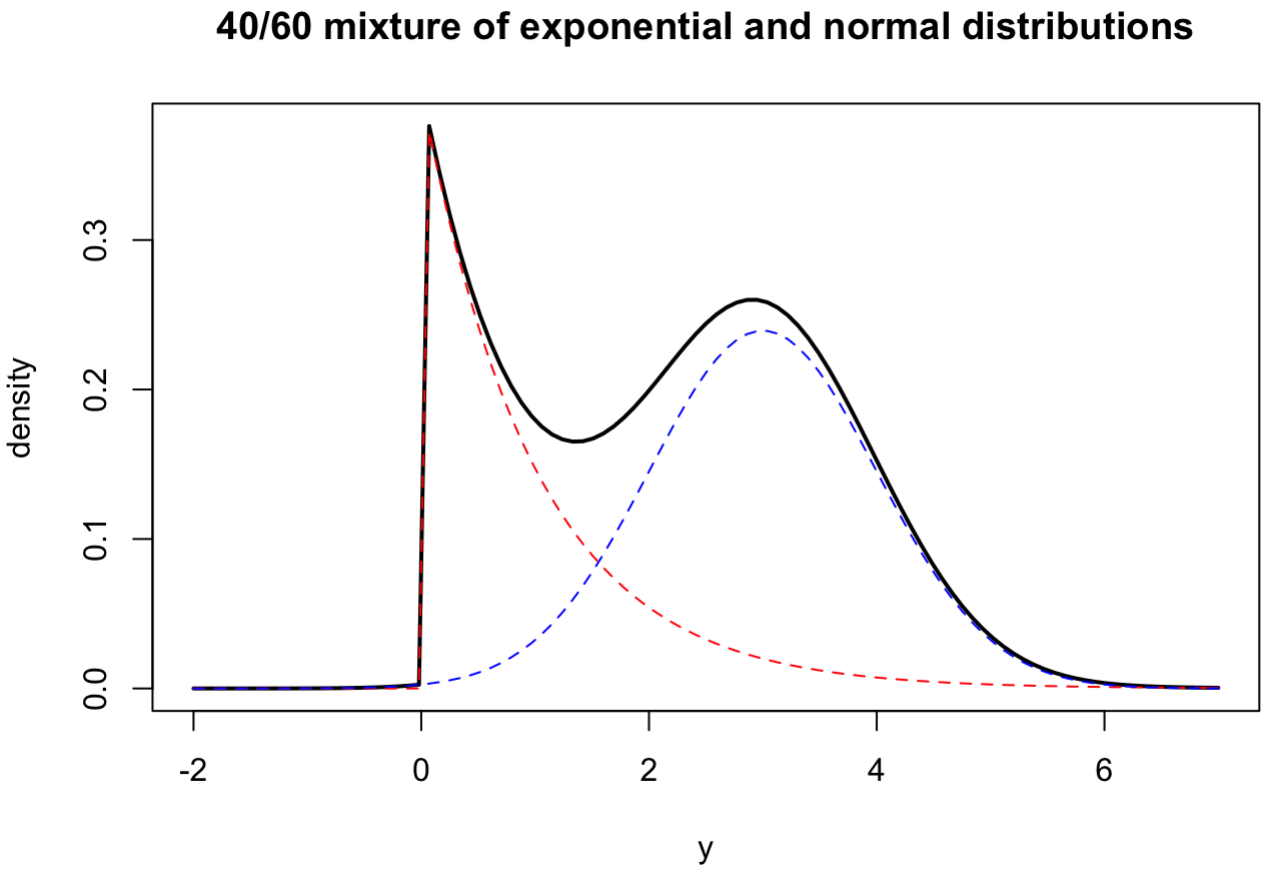
```
curve( 0.4*dexp(x, 1.0) + 0.6*dnorm(x, 3.0, 1.0), from=-2.0, to=7.0, ylab="density", xlab="y", m
ain="40/60 mixture of exponential and normal distributions", lwd=2)
```



We could think of these two distributions as governing two distinct populations, one following the exponential distribution and the other following the normal distribution.

Let's draw the weighted PDFs for each population.

```
curve( 0.4*dexp(x, 1.0) + 0.6*dnorm(x, 3.0, 1.0), from=-2.0, to=7.0, ylab="density", xlab="y", m
ain="40/60 mixture of exponential and normal distributions", lwd=2)
curve( 0.4*dexp(x, 1.0), from=-2.0, to=7.0, col="red", lty=2, add=TRUE)
curve( 0.6*dnorm(x, 3.0, 1.0), from=-2.0, to=7.0, col="blue", lty=2, add=TRUE)
```



The general form for a discrete mixture of distributions is as follows:

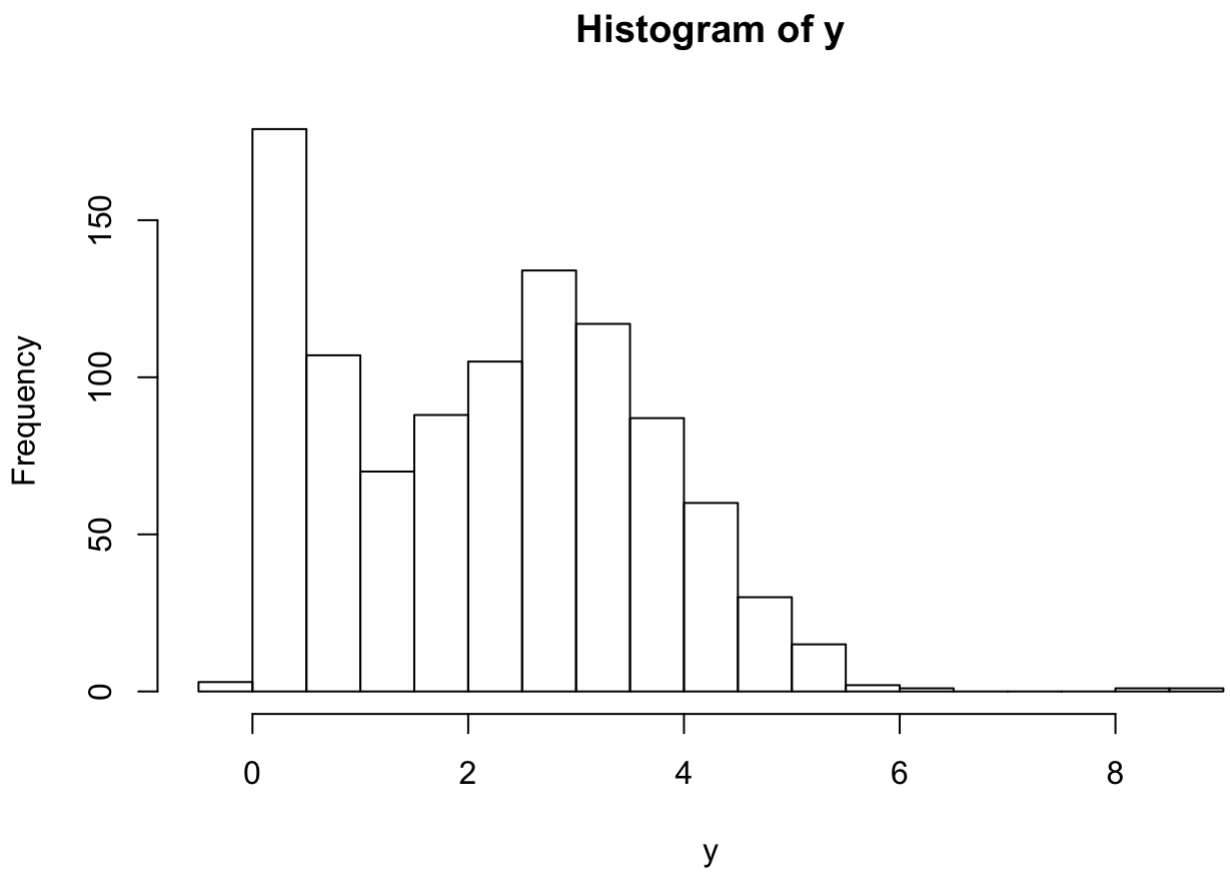
$$p(y) = \sum_{j=1}^{J} \omega_j \cdot f_j(y)$$

where the ω's are positive weights that add up to 1 (they are probabilities) and each of the J $f_j(y)$ functions is a PDF for some distribution. In the example above, the weights were 0.4 and 0.6, f_1 was an exponential PDF and f_2 was a normal PDF.

One way to simulate from a mixture distribution is with a hierarchical model. We first simulate an indicator for which "population" the next observation will come from using the weights ω. Let's call this z_i. In the example above, z_i would take the value 1 (indicating the exponential distribution) with probability 0.4 and 2 (indicating the normal distribution) with probability 0.6. Next, simulate the observation y_i from the distribution corresponding to z_i.

Let's simulate from our example mixture distribution.

```
set.seed(117)
n = 1000
z = numeric(n)
y = numeric(n)
for (i in 1:n) {
  z[i] = sample.int(2, 1, prob=c(0.4, 0.6)) # returns a 1 with probability 0.4, or a 2 with prob
ability 0.6
  if (z[i] == 1) {
    y[i] = rexp(1, rate=1.0)
  } else if (z[i] == 2) {
    y[i] = rnorm(1, mean=3.0, sd=1.0)
  }
}
hist(y, breaks=30)
```



If we keep only the y values and throw away the z values, we have a sample from the mixture model above. To see that they are equivalent, we can marginalize the joint distribution of y and z:

$$p(y) = \sum_{j=1}^{2} p(y, z = j) = \sum_{j=1}^{2} p(z = j) \cdot p(y \mid z = j) = \sum_{j=1}^{2} \omega_j \cdot f_j(y) \, .$$

Bayesian inference for mixture models

When we fit a mixture model to data, we usually only have the y values and do not know which "population" they belong to. Because the z variables are unobserved, they are called *latent* variables. We can treat them as parameters in a hierarchical model and perform Bayesian inference for them. The hierarchial model might look like this:

$$\begin{aligned} y_i \mid z_i, \theta &\overset{\text{ind}}{\sim} f_{z_i}(y \mid \theta), \quad i = 1, \ldots, n \\ \Pr(z_i = j \mid \omega) &= \omega_j, \quad j = 1, \ldots, J \\ \omega &\sim p(\omega) \\ \theta &\sim p(\theta) \end{aligned}$$

where we might use a Dirichlet prior (see the review of distributions in the supplementary material) for the weight vector ω and conjugate priors for the population-specific parameters in θ. With this model, we could obtain posterior distributions for z (population membership of the observations), ω (population weights), and θ (population-specific parameters in f_j). Next, we will look at how to fit a mixture of two normal distributions in `JAGS`.

Example with `JAGS`

Data

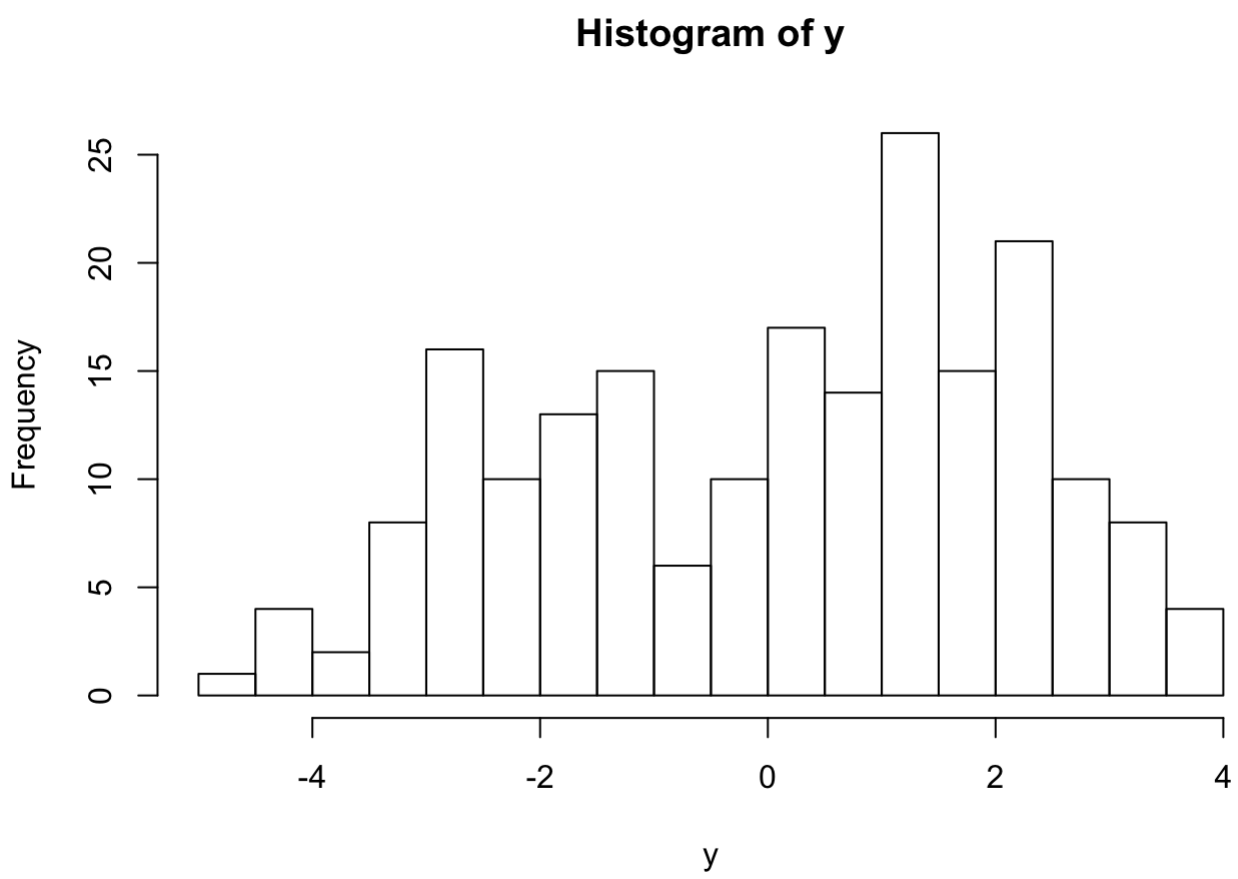
For this example, we will use the data in the attached file **`mixture.csv`**.

```
dat = read.csv("mixture.csv", header=FALSE)
y = dat$V1
(n = length(y))
```

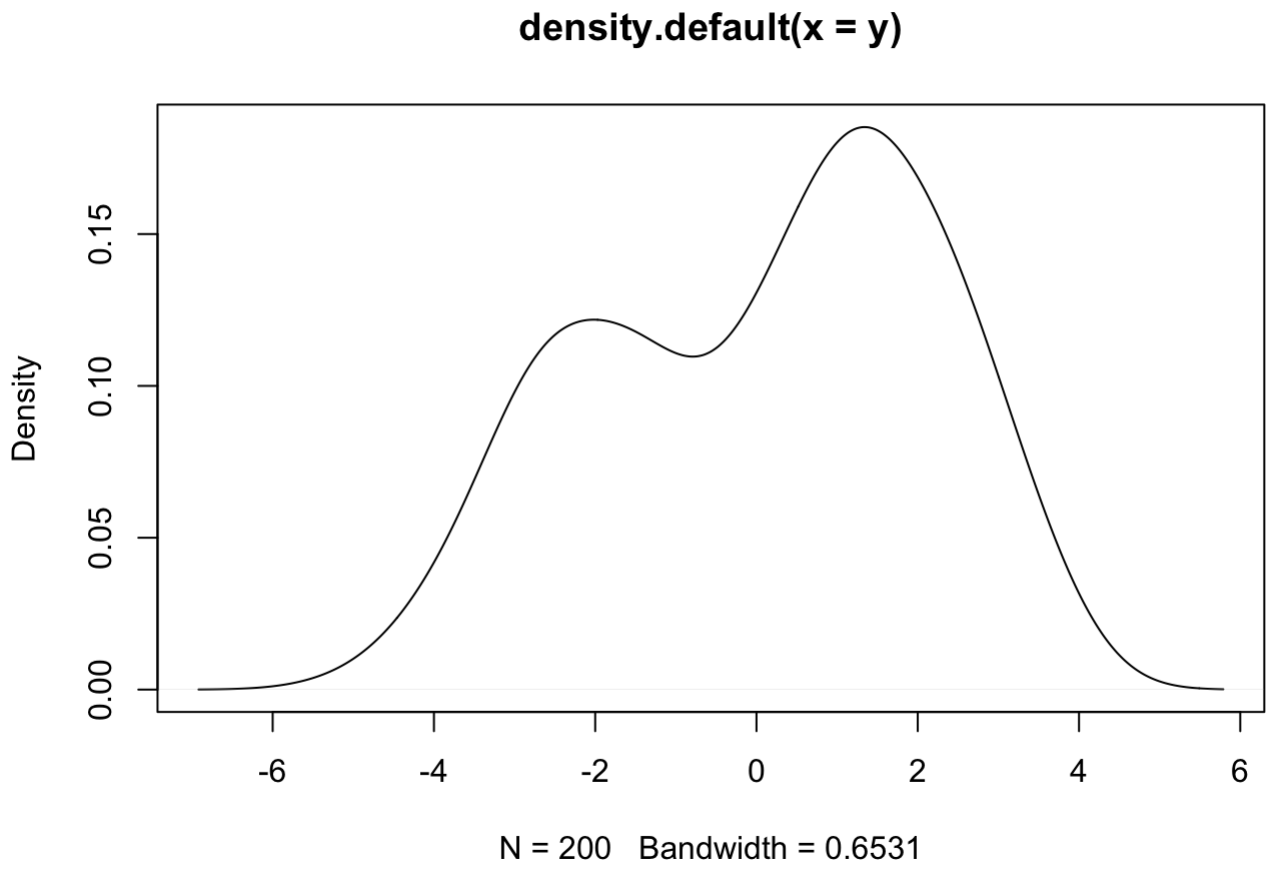
```
## [1] 200
```

Let's visualize these data.

```
hist(y, breaks=20)
```




```
plot(density(y))
```



It appears that we have two populations, but we do not know which population each observation belongs to. We can learn this, along with the mixture weights and population-specific parameters with a Bayesian hierarchical model.

We will use a mixture of two normal distributions with variance 1 and different (and unknown) means.

Model

```
library("rjags")
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.2.0
```

```
## Loaded modules: basemod,bugs
```

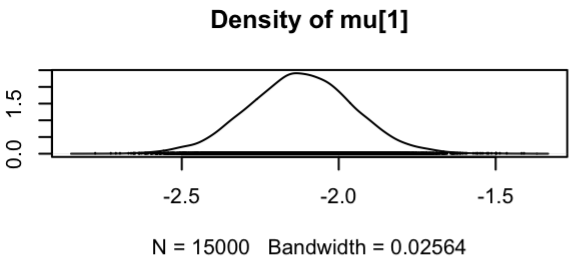
```
mod_string = " model {
    for (i in 1:length(y)) {
        y[i] ~ dnorm(mu[z[i]], prec)
      z[i] ~ dcat(omega)
    }

  mu[1] ~ dnorm(-1.0, 1.0/100.0)
    mu[2] ~ dnorm(1.0, 1.0/100.0) T(mu[1],) # ensures mu[1] < mu[2]

    prec ~ dgamma(1.0/2.0, 1.0*1.0/2.0)
  sig = sqrt(1.0/prec)

    omega ~ ddirich(c(1.0, 1.0))
} "

set.seed(11)

data_jags = list(y=y)

params = c("mu", "sig", "omega", "z[1]", "z[31]", "z[49]", "z[6]") # Select some z's to monitor

mod = jags.model(textConnection(mod_string), data=data_jags, n.chains=3)
update(mod, 1e3)

mod_sim = coda.samples(model=mod,
                       variable.names=params,
                       n.iter=5e3)
mod_csim = as.mcmc(do.call(rbind, mod_sim))

## convergence diagnostics
plot(mod_sim, ask=TRUE)

autocorr.diag(mod_sim)
effectiveSize(mod_sim)
```

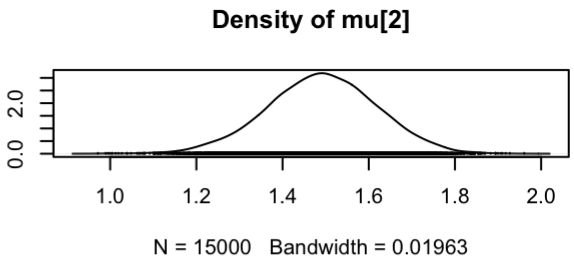
Results

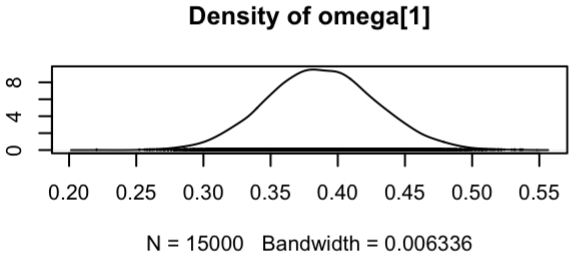
```
summary(mod_sim)
```

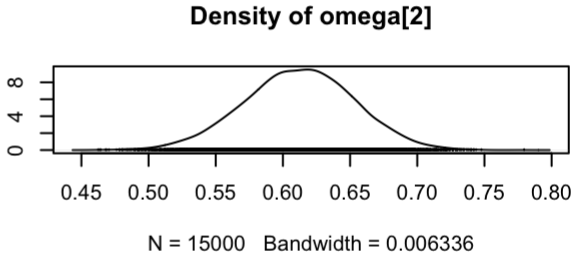
```
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean      SD  Naive SE Time-series SE
## mu[1]    -2.120 0.16681 1.362e-03      2.769e-03
## mu[2]     1.491 0.12704 1.037e-03      1.674e-03
## omega[1]  0.388 0.04090 3.339e-04      5.237e-04
## omega[2]  0.612 0.04090 3.339e-04      5.237e-04
## sig       1.137 0.07525 6.144e-04      1.005e-03
## z[1]      1.011 0.10399 8.491e-04      9.312e-04
## z[31]     1.572 0.49484 4.040e-03      4.375e-03
## z[49]     1.801 0.39901 3.258e-03      3.497e-03
## z[6]      2.000 0.01155 9.428e-05      9.428e-05
##
## 2. Quantiles for each variable:
##
##             2.5%     25%     50%     75%   97.5%
## mu[1]    -2.4504 -2.2316 -2.1201 -2.0098 -1.7901
## mu[2]     1.2332  1.4069  1.4921  1.5767  1.7356
## omega[1]  0.3095  0.3601  0.3873  0.4151  0.4696
## omega[2]  0.5304  0.5849  0.6127  0.6399  0.6905
## sig       1.0050  1.0845  1.1311  1.1834  1.3012
## z[1]      1.0000  1.0000  1.0000  1.0000  1.0000
## z[31]     1.0000  1.0000  2.0000  2.0000  2.0000
## z[49]     1.0000  2.0000  2.0000  2.0000  2.0000
## z[6]      2.0000  2.0000  2.0000  2.0000  2.0000
```

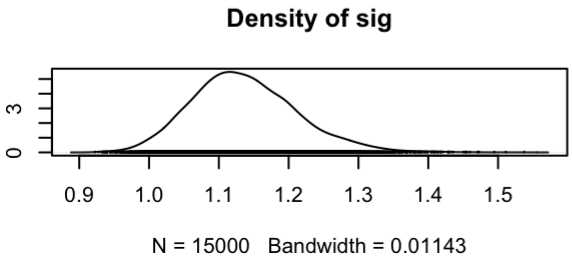
```
## for the population parameters and the mixing weights
par(mfrow=c(3,2))
densplot(mod_csim[,c("mu[1]", "mu[2]", "omega[1]", "omega[2]", "sig")])

## for the z's
par(mfrow=c(2,2))
```



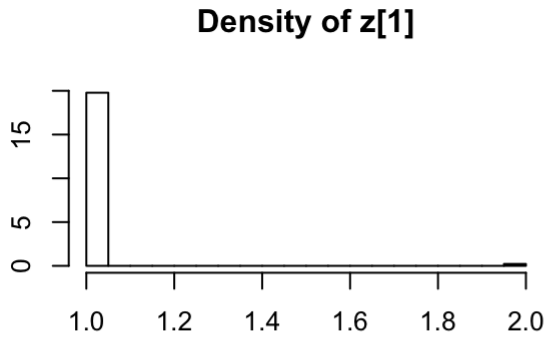


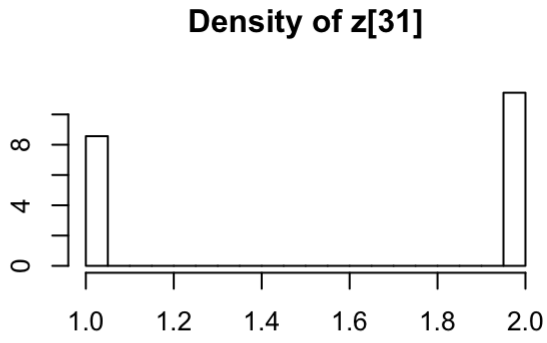


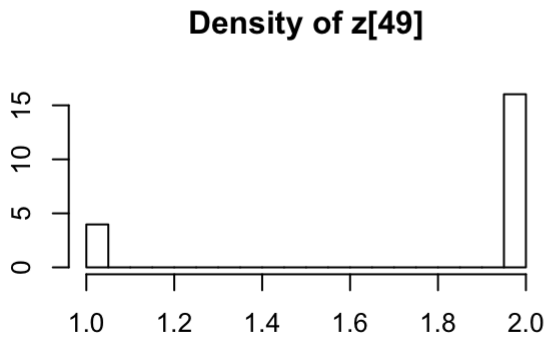


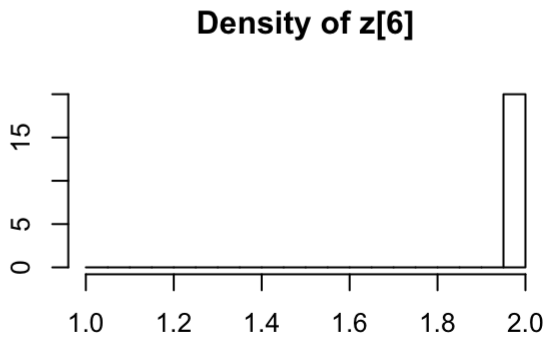



```
densplot(mod_csim[,c("z[1]", "z[31]", "z[49]", "z[6]")])
```










```
table(mod_csim[,"z[1]"]) / nrow(mod_csim) ## posterior probabilities for z[1], the membership of y[1]
```

```
##
##          1          2
## 0.98906667 0.01093333
```

```
table(mod_csim[,"z[31]"]) / nrow(mod_csim) ## posterior probabilities for z[31], the membership of y[31]
```

```
##
##         1         2
## 0.4282667 0.5717333
```

```
table(mod_csim[,"z[49]"]) / nrow(mod_csim) ## posterior probabilities for z[49], the membership of y[49]
```

```
##
##         1         2
## 0.1986667 0.8013333
```

```
table(mod_csim[,"z[6]"]) / nrow(mod_csim) ## posterior probabilities for z[6], the membership of y[6]
```

```
##
##            1            2
## 0.0001333333 0.9998666667
```

```
y[c(1, 31, 49, 6)]
```

```
## [1] -2.2661749 -0.3702666  0.0365564  3.7548080
```

If we look back to the y values associated with these z variables we monitored, we see that y_1 is clearly in Population 1's territory, y_{31} is ambiguous, y_{49} is ambiguous but is closer to Population 2's territory, and y_6 is clearly in Population 2's territory. The posterior distributions for the z variables closely reflect our assessment.