School of Mathematical and Statistical Sciences
Arizona State University

Student: Juyoung Choi

Other students: Ziwei Chen, Qingqing Wu

Course: STP 540

Program: Statistics MS

Instructor: Robert McCulloch

Date: Fall 2019

Computational Statistics Final Project

EM Algorithm and Gibbs Sampler on latent variable model

Juyoung Choi, Ziwei Chen, Qingqing Wu

November 28, 2019

Abstract

EM Algorithm and Gibbs sampler are two useful Bayesian simulation methods for parameter estimation of finite normal mixture model. The EM Algorithm is an iterative estimate of maximum likelihood for incomplete data problem. Gibbs sampler is an approach of generating random sample from a multivariate distribution. Our research based on mixture latent gaussian model. Compared two methods on both simulated data and real data.

Data

We use two datasets to do the experiments.

1. California housing data
   We focus on one column, named `longitude`. Do one-dimension method.

2. Simulated Data
   We use some parameters to generate simulated data.
   Parameters: `mu`=(15, 33), `sigma`=(2, 4), `p`=(0.5, 0.5)
   Using function `rmultinom(1, size=100, prob=p)`, we generate the latent variable (I) to set the number of observations in each group. Then we generate the observation data using `rnorm(n=I.i[i], mean=mu[i], sd=sigma[i])` in each group.

EM Algorithm

The EM (Expectation-Maximization) algorithm is a method for finding maximum likelihood estimates of parameters in statistical models, where the model depends on unobserved latent variables. It is an iterative procedure that works well in many situations. The algorithms consisted of two steps. One is expectation step, usually called E Step. The other is maximization step, usually called M Step. Actually, indicator is an approach to get different groups. If k=2 and we have only one Y_i, model will become mixture binormal distribution. If $k > 2$, that will become a mixture multivariate normal model. We will see the difference on plots if K changes from 2 to 4. Also, we will use a package `mixtools` to see if our result is right or not.

E-step

- Step 1: our density function is

$$f(x; \mu, \sigma, p) = \sum_{i=1}^{2} p_i f_i(x; \mu_i, \sigma_i, p_i)$$

- Step 2: conditional probability to which subpoulation each observation is

$$Pr(\mu_j = i|x_j; \mu, \sigma, p) = \frac{p_r f_r(x_j; \mu_i, \sigma_i, p_i)}{\sum_{i=1}^{2} p_i f_i(x_j; \mu_i, \sigma_i, p_i)}$$

- Step 3: expected value of the log-likelihood

$$L = \sum_{j=1}^{n} \sum_{i=1}^{2} Pr(\mu_j = i|x_j; \mu, \sigma, p) \left[logp_i + logf_i(x; \mu_i, \sigma_i, p_i) \right]$$

M-step

Here comes our maximizing values for μ, σ, and p

- $p_i' = \frac{\sum_{j=1}^{n} Pr(\mu_j = i|x_j; \mu, \sigma, p)}{n}$
- $\mu_i' = \frac{\sum_{j=1}^{n} x_j Pr(\mu_j = i|x_j; \mu, \sigma, p)}{\sum_{j=1}^{n} Pr(\mu_j = i|x_j; \mu, \sigma, p)}$
- $\sigma_i' = \sqrt{\frac{\sum_{j=1}^{n} (x_j - \mu_i')^2 Pr(\mu_j = i|x_j; \mu, \sigma, p)}{\sum_{j=1}^{n} Pr(\mu_j = i|x_j; \mu, \sigma, p)}}$

In the loop, we make a threshold for L, it does iterations until the difference between next L and last L is smaller than $1e-6$. Usually, we can make an assumption for the value of k. We can plot a density plot for data. For instance, the density plot of `longitude` has two apparent peaks. But there still two small peaks in the density. For the plot like normal distribution, we can use K-means to deal with indicator. Make them into several groups and get EM algorithm on it.

A. For Real Data

1. K=2
   First, we try $k = 2$,
   true data: $\mu = (-118.0571, -121.8482), \sigma = (0.7705283, 0.7266195), p = (0.6010174, 0.3989826)$
   EM results: $\mu = (-118.0773, -121.8667), \sigma = (0.7971874, 0.7139101), p = (0.6061658, 0.3938342)$
   From plot, we can see that it is a mixture normal distribution, consisted of two normal distributions. We just try $k = 2$ to see if it fits dataset well or not. From plot, it seems like k should be 4. We use EM Algorithm to make a simulation, use a k-means clustering to get an estimate for the starting values for μ, σ, and p. Actually, indicators are latent variables and it represents for groups.
   From plot for $k = 2$, it fits dataset well, but not very well. Results for μ, σ and p are very closed to real data. Stop at the 21th iteration.

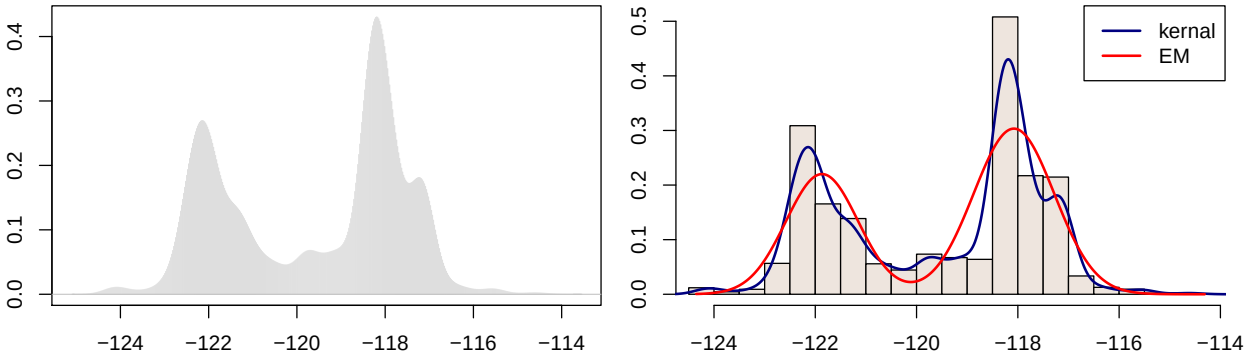


Figure 1: Density plot for `longtitude` (left) and comparison density plot for EM and `longtitude` density (right) when $k = 2$

2. library `mixtools`
   True data: $\mu = (-118, -122), \sigma = (0.77, 0.73), p = (0.61, 0.39)$
   Stops at 21th, with result: $\mu = (-118.0773, -121.8667), \sigma = (0.79719, 0.71391), p = (0.60617, 0.39383)$
   Pretty good! Almost totally same with our EM algorithm!

3. K=4
   We want to see if it can fit the data better. Stop at 177th iteration, with
   $\mu = (-117.5701, -121.9414, -119.6373, -118.2136)$
   $\sigma = (0.6984963, 0.6406363, 0.5566126, 0.1823193)$
   $p = (0.2787177, 0.3752978, 0.09582889, 0.2501556)$
   Density plot shows $k = 4$ fits dataset better, we like $k = 4$!

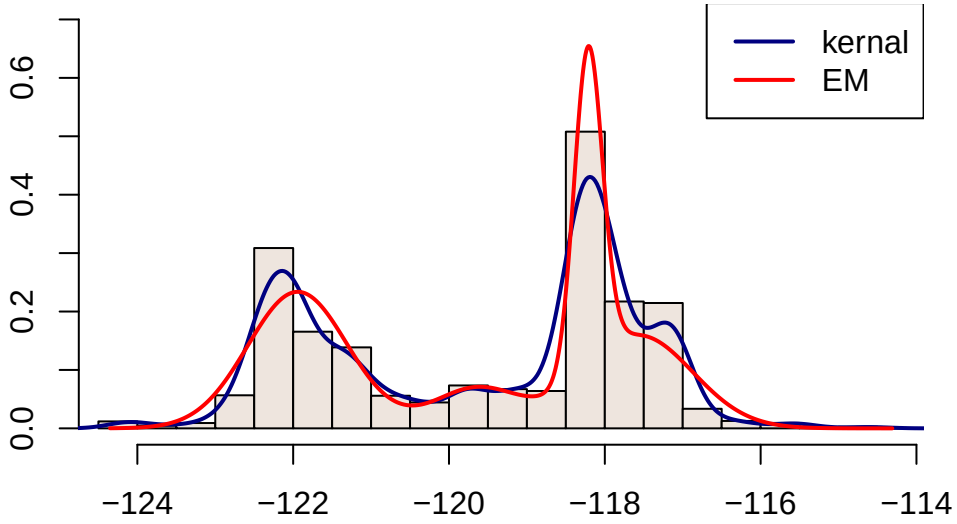


Figure 2: Comparison density plot for EM and `longtitude` density when $k = 4$

B. For Simulated Data

Density plot shows there two peaks, there is no doubt we choose k=2 to do EM. It is a two normal mixture model.
True data: $\mu = (15, 33), \sigma = (2, 4), p = (0.5, 0.5)$
Iteration stops at 5th, with results
$\mu = (14.90974, 33.27854), \sigma = (2.085509, 3.969223), p = (0.4784425, 0.5215575)$
Comparing results from EM Algorithm to true value, we found that values are very near to true value, converges really well. And density plots also show that two group (k=2) fits initial dataset well.

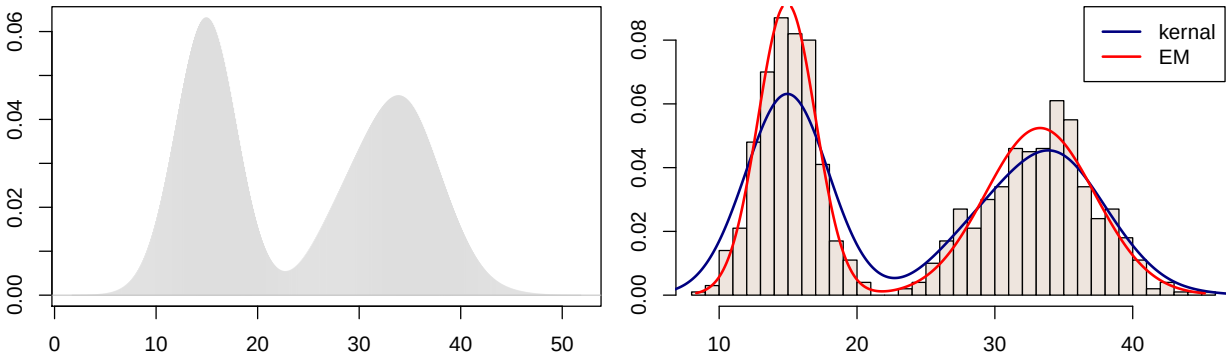


Figure 3: Density plot (left) and comparison density plot for EM and the simulated data (right) when $k = 2$

Gibbs Sampling

In this section, we give the posterior distribution for two-component normal mixture model under a conjugate prior and present the Gibbs sampler for normal mixture application. Gibbs sampler is a useful simulation method which generates sample from the posterior distribution. In order to specify the model, we need to put priors on μ, σ and p. Here are the conditional distributions of each draws below:

The Gibbs sampler is:

$$p|I, \quad I|\mu, \sigma, p, y, \quad \mu|\sigma, I, y, \quad \sigma|\mu, I, y$$

where,

- the draw of p is a Dirichlet
- the draw of each I_i is an independent multinomial
- the draw of each μ_j is an independent normal
- the draw of each σ_j is an independent inverted chi-squared.

Then we give the Gibbs sampler for two-component normal mixture model.

Algorithm:

1. First, take some initial values $\theta^{(0)} = (p^{(0)}, \mu_1^{(0)}, \mu_2^{(0)}, (\sigma_1^2)^{(0)}, (\sigma_2^2)^{(0)})$, where those parameters come from the prior distributions.

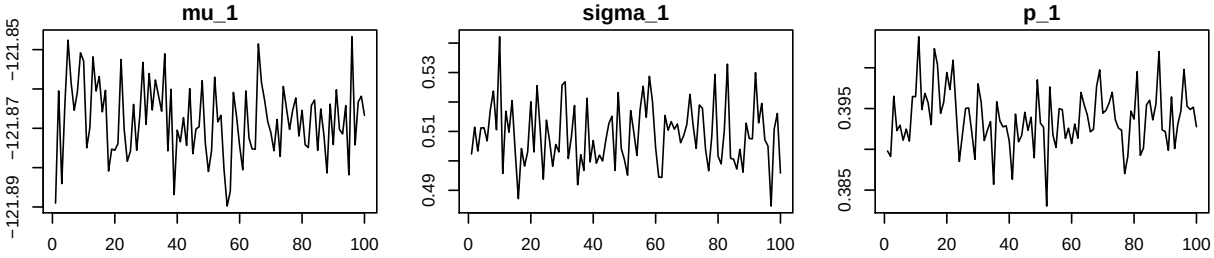
2. Repeat for $t = 1, 2, 3, \ldots$
   (a). For $i = 1, 2, \ldots, n$, generate $z_i^{(t)} \in \{0, 1\}$ with $z_i^{(t)} \sim Ber\left(\frac{p^{(t)}\varphi_{\theta_2}(y_i)}{(1-p^{(t)})\varphi_{\theta_1}(y_i)+p^{(t)}\varphi_{\theta_2}(y_i)}\right)$
   (b). For $j = 1, 2$, generate parameters as followed:
   - $p^{(t+1)} \sim Be\left(\overline{z}_2^{(t)} + \alpha,\ \overline{z}_1^{(t)} + \beta_2\right)$
   - $(\sigma_j^2)^{(t+1)} \sim IG\left(\frac{\nu_j+\overline{z}_j^{(t)}}{2},\ \frac{1}{2}\left[s_j^2 + \sum_{i=1}^n z_{ij}^{(t)}(y_i - \overline{y}_j(z)^{(t)})^2 + \frac{n_j\overline{z}_j^{(t)}(\overline{y}_j(z)^{(t)}-\xi_j)^2}{n_j+\overline{z}_j^{(t)}}\right]\right)$
   - $\mu_j^{(t+1)} \sim N\left(\frac{n_j\xi_j+\overline{z}_j^{(t)}\overline{y}_j(z)^{(t)}}{n_j+\overline{z}_j^{(t)}},\ \frac{(\sigma_j^2)^{(t+1)}}{n_j+\overline{z}_j^{(t)}}\right)$

3. Continue step 2 until the joint distribution of $(z^{(t)}, \theta^{(t)})$ does not change.

A. For Real Data

We get the following results:
true data: $\mu = (-118.0571, -121.8482), \sigma = (0.7705283, 0.7266195), p = (0.6010174, 0.3989826)$
Gibbs results: $\mu = (-118.0778, -121.8679), \sigma = (0.6371326, 0.5089723,), p = (0.6061779, 0.3938221)$

Compared the true value of μ, σ, and p, we can get that μ and p are close to the true value. However, σ is way less than the true value.



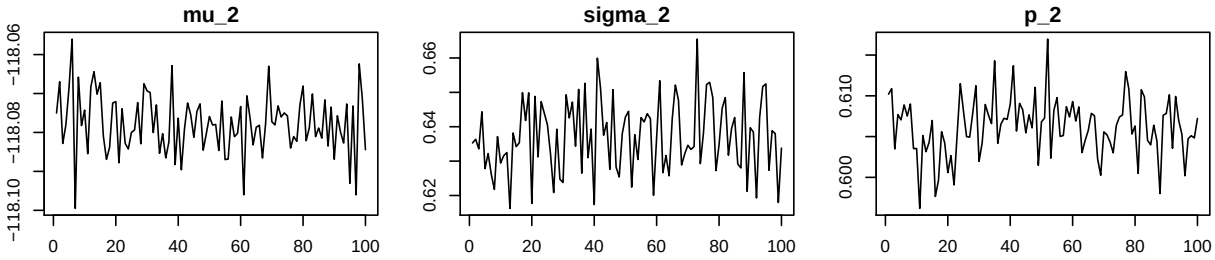


Figure 4: Iteration plots on the California Housing Data

From Figure 4, we can see that the parameters are fluctuating within the iterations. And until the convergence, it will not change.

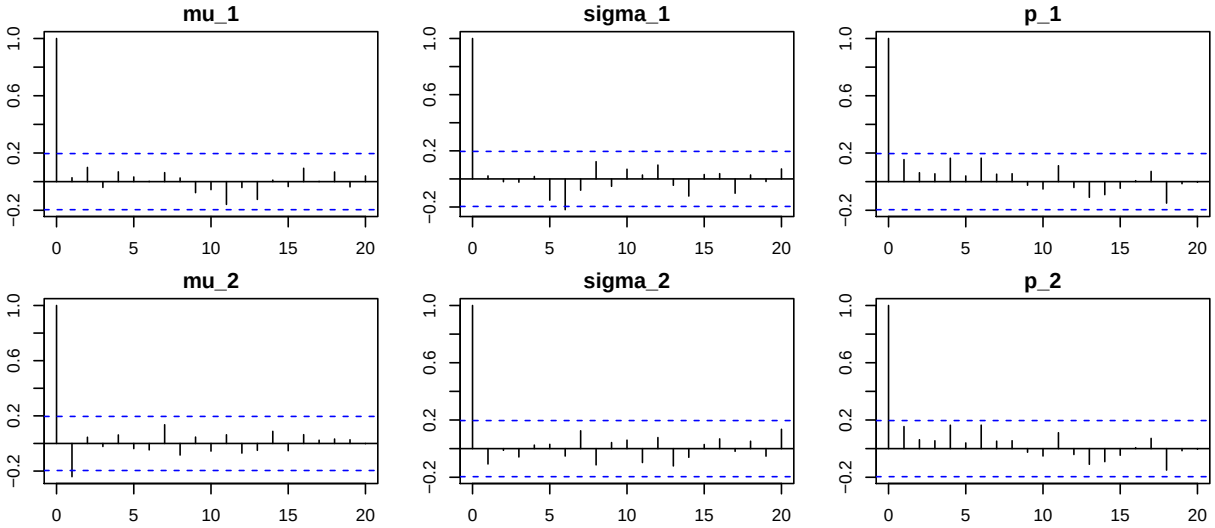


Figure 5: ACF plots on the California Housing Data

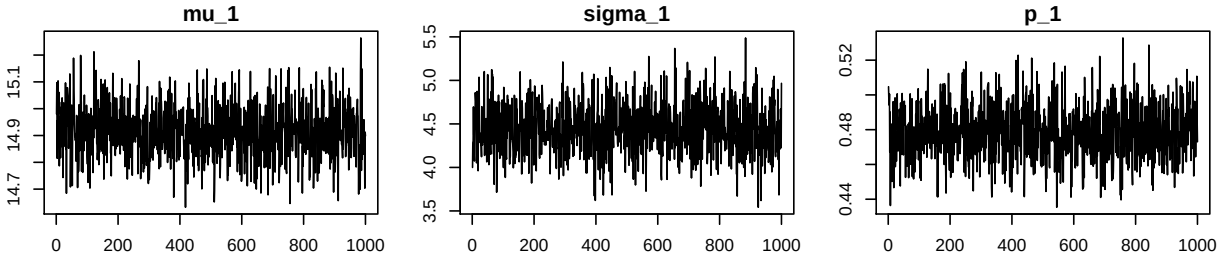
From Figure 5, it is obvious that there is no burn-in. It looks nice.

B. For Simulated Data

In order to avoid burn-in , after many tries, we discard the first 100 observations.
The results look good. No burn-in.

True data: $\mu = (15, 33), \sigma = (2, 4), p = (0.5, 0.5)$
Final result: $\mu = (14.91693, 33.27526), \sigma = (2.108684, 3.97925), p = (0.4786722, 0.5213278)$

Compared the true value of μ, σ and p, it is obvious that there is no much difference on it. It fits well.



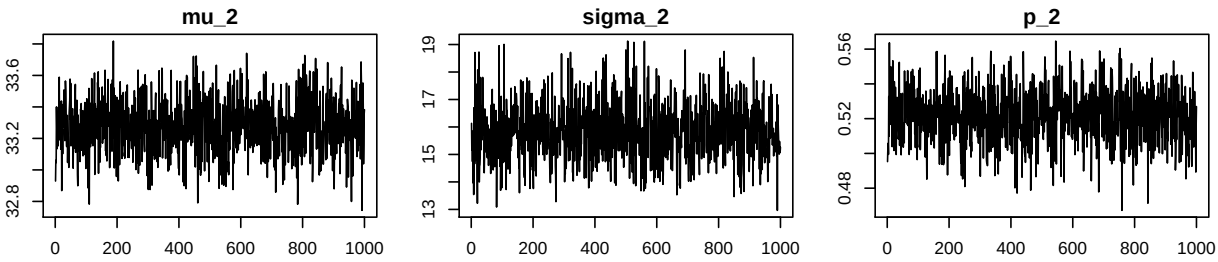


Figure 6: Iteration plots on the simulated Data

From Figure 6, we can still see that the parameters are fluctuated within the iterations. And until the convergence, it will not change.

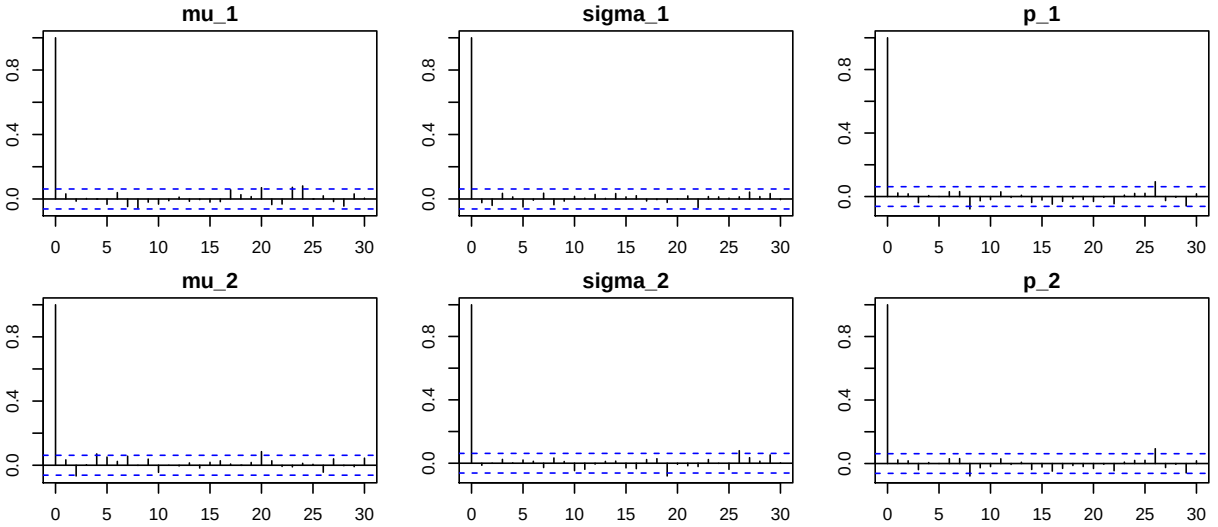


Figure 7: ACF plots on the simulated Data

From Figure 7, it is obvious that there is no burn-in since I have discarded the first few observations. It looks good!

Comparison of Results

Table 1. Comparison of True data and the results from EM and Gibbs

		True	EM	Gibbs
Real Data	`mu`	(-118.0571, -121.8482)	(-118.0773, -121.8667)	(-118.0778, -121.8679)
	`sigma`	(0.7705, 0.7266)	(0.7972, 0.7139)	(0.6371, 0.5090)
	`p`	(0.6010, 0.3990)	(0.6062, 0.3938)	(0.6062, 0.3938)
Simulated Data	`mu`	(15, 33)	(14.9097, 33.2785)	(14.9169, 33.2753)
	`sigma`	(2, 4)	(2.08551, 3.9692)	(2.1087, 3.9795)
	`p`	(0.5, 0.5)	(0.4784, 0.5216)	(0.4787, 0.5213)

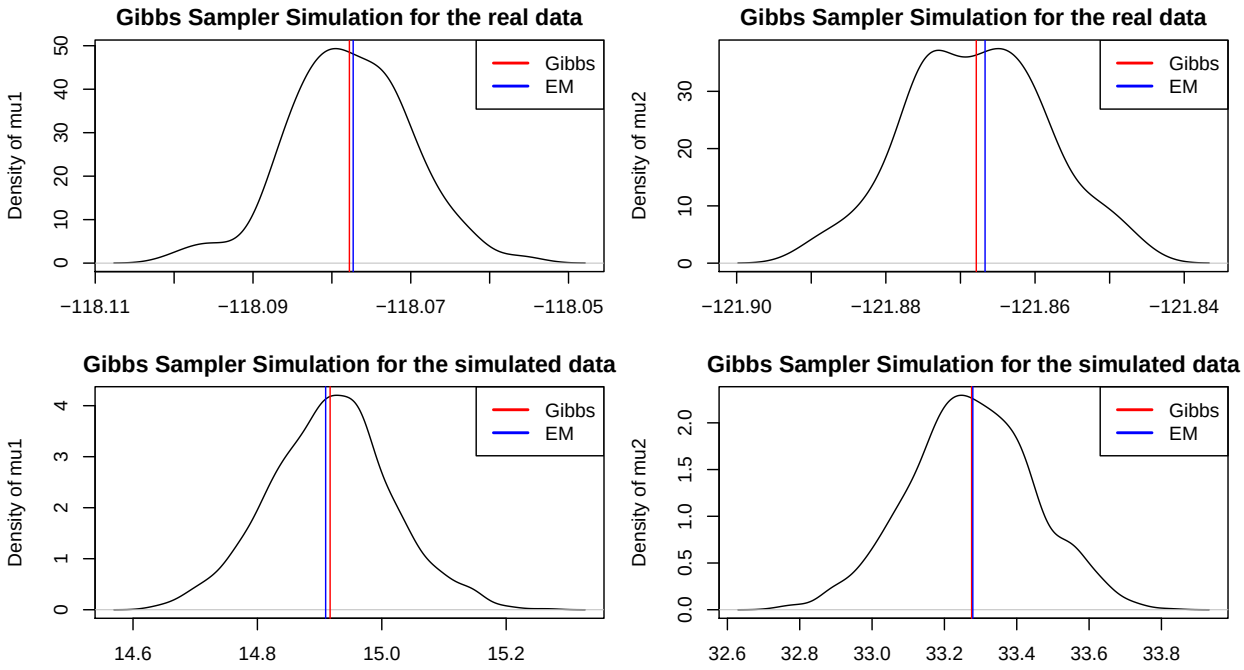


Figure 8. Comparison plots of EM Algorithm and Gibbs sampler

From the density plot (Figure 2), we consider the longitude data as following a two-component normal mixture model. Use the common way to choose the initial values of parameters and run the EM Algorithm and Gibbs sampler, we get the estimation results above. Figure 8 and Table 1 show that both algorithms also get similar estimate results for the real and simulated data, and the EM Algorithm still seems faster and more stable. But we notice that σ for the real data is not close to the real variance. We guess that it might include the noise for the real data. Also, Gibbs sampler might contain more information. If we set the iterations 1000, 10000, we want to know what results we get.

Conclusion

The EM Algorithm and the Gibbs sampler are both good simulation methods for parameter estimation, and they usually get similar results. The EM Algorithm needs no prior information and it is more stable. Gibbs sampler is more complicated in computing but also contains more information, which is better for further statistical analysis. Summarizing the simulation results, we suggest using the EM Algorithm when having a small number of data set, and choose Gibbs sampler when more data and information need to be dealt with.

References

McCulloch, Rob. Clustering. http://www.rob-mcculloch.org/2019_ml/webpage/notes/clustering.pdf.

Please see the following pages for the R script used in this project.

```
## R-Code for mixtures models
set.seed(1024)
rmix=function (n=100,
            k=2,mu=rep(0,k),
            sigma2=rep(1,k),
            p=rep(1/k,k)) {
 ## Set I the number of observations in each group
 I.i = rmultinom(1, size=n, prob=p)
 ## Create group membership indicator variable
 z = rep(1:k, times=I.i)
 ## Generate data
 x = NULL
 for (i in 1:length(I.i)) {
  x.i = rnorm(n=I.i[i], mean=mu[i], sd=sqrt(sigma2[i]))
  x = c(x, x.i)
 }
 ## Create data object
 data       = NULL
 data$n     = n
 data$k     = k
 data$mu    = mu
 data$sigma2 = sigma2
 data$pi    = p
 data$I.i   = I.i
 data$z     = z
 data$x     = x
 ## Return data object
 return (data)
}
dmix = function (x, par) {
 parmatrix = matrix(as.numeric(par), nrow=3, byrow=T)
 k        = dim(parmatrix)[2]
 p        = parmatrix[1,]
 mu       = parmatrix[2,]
 sigma2   = parmatrix[3,]
 ## Set x values to evaluate density over range of data +- 10%
 x0 = min(x)
 x1 = max(x)
 d0 = x0 - 0.1*(x1-x0)
 d1 = x1 + 0.1*(x1-x0)
 dx = seq(d0, d1, length=10*length(x))
 ## Calculate mixture density
 y<- rep(0, length(x))
 for (j in 1:k) {
  y = y + p[j] * dnorm(dx, mean=mu[j], sd=sqrt(sigma2[j]))
 }
 ## Create density object
 dens = NULL
 dens$y = y
 dens$x = dx
```

```
  ## Return object
  return (dens)
}

#-----------------EM algorithm on real data----------------------------------------

### From plot, we can see it is mixture normal distribution, consisted of two
## normal distribution. We just try k=2 to see if it fits dataset well or not.
## From plot, it seems like k should be 4.
## We use EM algorithm to make a simulation
#use a k means clustering to get an estimate for
#the starting values for mu, sigma, and p.
## Actually, indicatoris latent variables and it represent for groups.

## let's try k=4.
rm(list=ls())
cah = read.csv("http://www.rob-mcculloch.org/data/calhouse.csv")
x = cah$longitude
set.seed(1024)
Idt = kmeans(x,4)$cluster ## divided into four grops, Idt means indicator
mu1 = mean(x[Idt==1]);mu2 = mean(x[Idt==2])
mu3 = mean(x[Idt==3]);mu4 = mean(x[Idt==4])
sigma1 = sd(x[Idt==1]);sigma2 = sd(x[Idt==2])
sigma3 = sd(x[Idt==3]);sigma4 = sd(x[Idt==4])

pi1 = sum(Idt==1)/length(Idt);pi2 = sum(Idt==2)/length(Idt)
pi3 = sum(Idt==3)/length(Idt);pi4 = sum(Idt==4)/length(Idt)
sumlg= function(x) {
  sum(x[is.finite(x)])
}
Loglikd = 0
## Loglikelihood, according to the function for all groups
Loglikd[2] = sumlg(log(pi1)+log(dnorm(x, mu1, sigma1)))+
  sumlg(log(pi2)+log(dnorm(x, mu2, sigma2)))+
  sumlg(log(pi3)+log(dnorm(x, mu3, sigma3)))+
  sumlg(log(pi4)+log(dnorm(x, mu4, sigma4)))
I = 2
# give a threshold for interation
inter=1 ## wanted to see how results changes in interation
## do loop, for the threshold 1*10^-6
while (abs(Loglikd[I]-Loglikd[I-1])>= 1e-6) {
  {
    # E step
    g1 = pi1 * dnorm(x, mu1, sigma1);g2 = pi2 * dnorm(x, mu2, sigma2)
    g3 = pi3 * dnorm(x, mu3, sigma3);g4 = pi4 * dnorm(x, mu4, sigma4)
    g_sum = g1 + g2+ g3 + g4 ## density function for mixture groups (4)
    p1 = g1/g_sum ## conditional probalibity for group1
    p2 = g2/g_sum;p3 = g3/g_sum;p4 = g4/g_sum
    # M step: all these accrording to the function
    ## probability
    pi1 = sumlg(p1) / length(x) ;pi2 = sumlg(p2) / length(x)
    pi3 = sumlg(p3) / length(x);pi4 = sumlg(p4) / length(x)
```

```
   ## mu for group
   mu1 = sumlg(p1 * x) / sumlg(p1);mu2 = sumlg(p2 * x) / sumlg(p2)
   mu3 = sumlg(p3 * x) / sumlg(p3);mu4 = sumlg(p4 * x) / sumlg(p4)
   ##sigma(sd) for group
   sigma1 = sqrt(sumlg(p1 * (x-mu1)^2) / sumlg(p1)) ;sigma2 = sqrt(sumlg(p2 * (x-mu2)^2) / sumlg(p2))
   sigma3 = sqrt(sumlg(p3 * (x-mu3)^2) / sumlg(p3));sigma4 = sqrt(sumlg(p4 * (x-mu4)^2) / sumlg(p4))
   ## output p1,p1,p3,p4
   p1 = pi1 ;p2 = pi2
   p3 = pi3;p4 = pi4

   I = I + 1
   Loglikd[I] = sum(log(g_sum))
  }
  cat("On iteration", inter,"mu1", mu1,"mu2",mu2,"mu3",mu3,"mu4",mu4,
     "sigma1", sigma1, "sigma2",sigma2, "sigma3", sigma3,"sigma4", sigma4,
     "p1",p1, "p2",p2, "p3",p3,"p4",p4,'\n')
  inter = inter + 1
}
## interation stops at 117th.
## use plot to show is two mixture normal model fits the data well.
hist(x, prob=T, breaks=32, xlim=c(range(x)[1], range(x)[2]), ylim=c(0,0.7),main='',xlab="longitude",
col="seashell2")
lines(density(x), col="navy", lwd=2)
x1 = seq(from=range(x)[1], to=range(x)[2], length.out=1000)
y = pi1 * dnorm(x1, mean=mu1, sd=sigma1) + pi2 * dnorm(x1, mean=mu2, sd=sigma2)+
  pi3 * dnorm(x1, mean=mu3, sd=sigma3)+pi4 * dnorm(x1, mean=mu4, sd=sigma4)
lines(x1, y, col="red", lwd=2)
legend('topright', col=c("navy", 'red'), lwd=2, legend=c("kernal", "EM"))

##----------------------  k=2   ----------------------------------------
rm(list=ls())
cah = read.csv("http://www.rob-mcculloch.org/data/calhouse.csv")
x = cah$longitude
plot(density(x), main="",xlab="longitute", col="grey87", type="h")

set.seed(1024)
Idt = kmeans(x,2)$cluster ##
mu1 = mean(x[Idt==1])
mu2 = mean(x[Idt==2])
sigma1 = sd(x[Idt==1])
sigma2 = sd(x[Idt==2])
pi1 = sum(Idt==1)/length(Idt)
pi2 = sum(Idt==2)/length(Idt)
## If we don't add this, it will turn out NA.
sumlg= function(x) {
  sum(x[is.finite(x)])
}
# starting value of expected value of the log likelihood
## two groups
Loglikd = 0
Loglikd[2] = sumlg(log(pi1)+log(dnorm(x, mu1, sigma1)))+
  sumlg(log(pi2)+log(dnorm(x, mu2, sigma2)))
```

```
I = 2
set.seed(1024)
# give a threshold for interation:1e-6, search a lot and decided to use 1*10^-6
inter=1

while (abs(Loglikd [I]-Loglikd[I-1]) >=1e-6) {
 {
  # E step
  g1 = pi1 * dnorm(x, mu1, sigma1)
  g2 = pi2 * dnorm(x, mu2, sigma2); g_sum=g1+g2
  p1 = g1/g_sum
  p2 = g2/g_sum

  # M step
  pi1 = sumlg(p1) / length(x)
  pi2 = sumlg(p2) / length(x)
  mu1 = sumlg(p1 * x) / sumlg(p1)
  mu2 = sumlg(p2 * x) /sumlg(p2)
  sigma1 = sqrt(sumlg(p1 * (x-mu1)^2) / sumlg(p1))
  sigma2 = sqrt(sumlg(p2 * (x-mu2)^2) / sumlg(p2))

  p1 = pi1
  p2 = pi2
  I = I + 1

  Loglikd[I] = sum(log(g_sum))
 }
 cat("On iteration", inter,"mu1", mu1,"mu2",mu2,
    "sigma1", sigma1, "sigma2",sigma2,"p1",p1, "p2",p2, '\n')
 inter = inter + 1
}
## interation stops at the 21st interation. All the outcomes are good compared to the true value.
## We use package to prove the EM algorithm is right.

## use plot to show is two mixture normal model fits the data well.
hist(x, prob=T, breaks=32, xlim=c(range(x)[1], range(x)[2]), main='',xlab="longitude", col="seashell2")
lines(density(x), col="navy", lwd=2)
x1 <- seq(from=range(x)[1], to=range(x)[2], length.out=1000)
y <- pi1 * dnorm(x1, mean=mu1, sd=sigma1) + pi2 * dnorm(x1, mean=mu2, sd=sigma2)
lines(x1, y, col="red", lwd=2)
legend('topright', col=c("navy", 'red'), lwd=2, legend=c("kernal", "EM"))
## We like the plot when k=4, it was great. Plot shows that four normal distributions fits data
##way better than k=2.
## WOW! Much better!!!!! We like the plot when k=4.
library(mixtools)
## use the same initial value which obtained from K-means process.
gm = normalmixEM(x,k=2,lambda=c(0.61, 0.39),mu=c(-118,-122),sigma=c(0.77,0.73))
cat("mu", gm$mu, "sigma", gm$sigma, "p", gm$lambda, '\n')
## number of iterations= 21 which is exactly the same as our EM algorithm
## And the answer for mean, standard deviation and possibility is almost the same.
## Thus, we are right.
```

```
##-------------------------------------------------------------------------------
## EM algorithm for simulated data, same in Gibbs Sampling
rm(list=ls())
mu = c(15,33)
sigma2 = c(4,16)
p = c(0.5,0.5)
## have to use same simulated data with Gibbs, so not K-means cluster this time.
set.seed(1024)
smdata = rmix (n=1000, k=2, mu=mu, sigma2=sigma2, p=p)
x=smdata$x
plot(density(x), xlab="x", col="grey87", type="h")

### From plot, we can see it is mixture normal distribution, consisted of two
## normal distribution. We will try k=2 to see if it fits dataset well or not.
## rMixNorm loop gives indicator of the observations, divided it into two groups.
mu1=15
mu2=33
sigma1=2
sigma2=4
pi1=0.5
pi2=0.5

sumlg= function(x) {
  sum(x[is.finite(x)])
}

Loglikd= 0
# starting value of expected value of the log likelihood
Loglikd[2] = sumlg(log(pi1)+log(dnorm(x, mu1, sigma1)))+
  sumlg(log(pi2)+log(dnorm(x, mu2, sigma2)))

I = 2
# give a threshold for interation
inter=1
while (abs(Loglikd[I]-Loglikd[I-1])>=1e-6) {
  {
    # E step
    g1 = pi1 * dnorm(x, mu1, sigma1)
    g2 = pi2 * dnorm(x, mu2, sigma2); g_sum=g1+g2
    p1 = g1/g_sum
    p2 = g2/g_sum

    # M step
    pi1 = sumlg(p1) / length(x)
    pi2 = sumlg(p2) / length(x)
    mu1 = sumlg(p1 * x) / sumlg(p1)
    mu2 = sumlg(p2 * x) / sumlg(p2)
    sigma1 = sqrt(sumlg(p1 * (x-mu1)^2) / sumlg(p1))
    sigma2 = sqrt(sumlg(p2 * (x-mu2)^2) / sumlg(p2))
    p1 = pi1
    p2 = pi2
```

```
  l = l + 1
  Loglikd[l] = sum(log(g_sum))
 }
 cat("On iteration", inter,"mu1", mu1,"mu2",mu2,
    "sigma1", sigma1, "sigma2",sigma2,"p1",p1, "p2",p2, '\n')
 inter = inter + 1
}
## interation stops at the 5th interation. All the outcomes are good compared to the true value.

## use plot to show is two mixture normal model fits the data well.
hist(x, prob=T, breaks=32, xlim=c(range(x)[1], range(x)[2]), main='',ylim=c(0,0.1),, col="seashell2")
lines(density(x), col="navy", lwd=2)
x1 <- seq(from=range(x)[1], to=range(x)[2], length.out=1000)
y <- pi1 * dnorm(x1, mean=mu1, sd=sigma1) + pi2 * dnorm(x1, mean=mu2, sd=sigma2)
lines(x1, y, col="red", lwd=2)
legend('topright', col=c("navy", 'red'), lwd=2, legend=c("kernal", "EM"))
## Conclusion: Comparing results from EM algorithm to true value, we found that values are very
## near true value, converges really well. And density plots also show that two group(k=2)
## fits initial dataset well.

#----------------- Gibbs----------------------------------------------------
#set prior for the normal mixture model
mu = c(15,33)
sigma2 = c(2,4)^2
p = c(0.5,0.5)
data = rmix (n=1000, k=2, mu=mu, sigma2=sigma2, p=p)
hist(data$x, prob=T, breaks=30, col="thistle", ylim=c(0,0.1))
X = seq(0,40, by=0.01)
dens = dmix(X, c(p, mu, sigma2))
points(dens$y ~ dens$x, type="l", lwd=2, col="coral")
##gibbs sampler for normal mixture model
gibbsmixnorm = function (x,
                  k=2,
                  keep=1000,
                  gap=1,
                  burn=0,
                  seed=NULL) {
 ## Create setup object
 setup = NULL
 ## Store mcmc parameters
 sim       = burn + gap*keep
 setup$sim  = sim
 setup$keep = keep
 setup$gap  = gap
 setup$burn = burn
 ## Determine which iterations to keep
 ikeep  = seq((burn+gap),sim, by=gap)
 icount = rep(0, sim)
 icount[ikeep] = 1:keep
 ## Set sample size
 n = length(x)
 ## Create data object
```

```
data = NULL
data$x = x
data$n = n
data$k = k
## Set storage vectors and matrices
pmatrix      = matrix(0, nrow=keep, ncol=k)
mumatrix     = matrix(0, nrow=keep, ncol=k)
sigma2matrix = matrix(0, nrow=keep, ncol=k)
zmatrix      = matrix(0, nrow=n, ncol=k)
## Set parameters
x0 = min(x)
x1 = max(x)
R  = x1 - x0
gamma.0  = rep(1,k)
mu.0     = rep((x0+x1)/2, k)
sigma2.0 = rep(R/2, k)
rate.0   = rep(2,k)
shape.0  = rep(R^2/50,k)
## Initialize parameters
z = cbind(rep(1,n), matrix(0, nrow=n, ncol=k-1))
for (i in 1:n) { z[i,] = sample(z[i,]) }
## Mixture proportions
p = apply(z, 2, mean)
## mean and variance
mu     = rep(0, k)
sigma2 = rep(0, k)
for (j in 1:k) {
  mu[j]     = mean( x[z[,j] == 1] )
  sigma2[j] =  var( x[z[,j] == 1] )
}
Order  = order(mu)
p     = p[Order]
mu     = mu[Order]
sigma2 = sigma2[Order]
z      = z[, Order]
af = matrix(0, nrow=n, ncol=k)
w  = matrix(1/k, nrow=n, ncol=k)
## Run MCMC
for (iter in 1:sim) {
  for (j in 1:k) {
    af[,j] = p[j]*dnorm(x, mean=mu[j], sd=sqrt(sigma2[j]))
  }
  ## Normalize group membership probabilities
  w = af / matrix(rep(rowSums(af), k), ncol=k, byrow=F)
  for (i in 1:n) {
    z[i,] = t(rmultinom(n=1, size=1, prob=w[i,]))
  }
  ## Sample parameters
  mix = apply(z, 2, sum)
  for (j in 1:k) { p[j] = rgamma(n=1, rate=1, shape=gamma.0[j] + mix[j]) }
  ## Normalize to dirichelet
  p = p/sum(p)
```

```
  for (j in 1:k) {
    ## Set mixture sample mean
    sum.x = sum(x[z[,j]==1])
    ## Set posterior mean and sd
    mu.mean = (mu.0[j]/sigma2.0[j] + sum.x/sigma2[j]) / (1/sigma2.0[j] + mix[j]/sigma2[j])
    mu.sd = sqrt(1/( 1/sigma2.0[j] + mix[j]/sigma2[j] ))
    ## Sample new mean from normal distribution
    mu[j] = rnorm(n=1, mean=mu.mean, sd=mu.sd)
  }
  for (j in 1:k) {
    sigma2.shape = rate.0[j]  + 0.5*mix[j]
    sigma2.scale = 1/(shape.0[j] + 0.5*sum( z[,j] * (x - mu[j])^2 ))
    sigma2[j] = 1 / rgamma(n=1, scale=sigma2.scale, shape=sigma2.shape)
  }
  Order  = order(mu)
  p     = p[Order]
  mu     = mu[Order]
  sigma2 = sigma2[Order]
  z      = z[, Order]
  ## Store parameters
  if (any(iter == ikeep)) {
    zmatrix  = zmatrix + z
    ## Store mixture parameters
    pmatrix[icount[iter],]     = p
    mumatrix[icount[iter],]     = mu
    sigma2matrix[icount[iter],] = sigma2
  }
} ## End MCMC
zpostmean = zmatrix / keep
## Set storage vectors and matrices
par          = NULL
par$p        = pmatrix
par$mu       = mumatrix
par$sigma2    = sigma2matrix
par$zpostmean = zpostmean
## Set parameter vector names
parnames = paste(rep(c("p.","mu.","sigma2."),each=k), rep(1:k,times=3), sep="")
## Posterior means
postmean = data.frame(t(c(apply(pmatrix,     2, mean),
                          apply(mumatrix,     2, mean),
                          apply(sigma2matrix, 2, mean))))
names(postmean)     = parnames
row.names(postmean) = "posterior.mean"
## Store summary object
summary = NULL
summary$posterior.means = postmean
## Create final object
mcmc = NULL
mcmc$setup   = setup
mcmc$data    = data
mcmc$par     = par
mcmc$summary = summary
```

```
  return (mcmc)
}
####For simulated data
gibbssim=gibbsmixnorm(data$x, k=2, keep=200, gap=10, burn=50,seed=1024)
print(gibbssim$summary)
####draw the plot
par(mfrow=c(1, 3))
plot(gibbssim$par$p[,1], main=paste("p_1",sep=""),type="l")
plot(gibbssim$par$p[,2], main=paste("p_2",sep=""),type="l")
plot(gibbssim$par$mu[,1], main=paste("mu_1",sep=""),type="l")
plot(gibbssim$par$mu[,2], main=paste("mu_2",sep=""),type="l")
plot(gibbssim$par$sigma2[,1], main=paste("sigma_1",sep=""),type="l")
plot(gibbssim$par$sigma2[,2], main=paste("sigma_2",sep=""),type="l")
###draw the acf
par(mfrow=c(1, 3))
acf(gibbssim$par$p[,1], main=paste("p_1",sep=""))
acf(gibbssim$par$p[,2], main=paste("p_2",sep=""))
acf(gibbssim$par$mu[,1], main=paste("mu_1",sep=""))
acf(gibbssim$par$mu[,2], main=paste("mu_2",sep=""))
acf(gibbssim$par$sigma2[,1], main=paste("sigma_1",sep=""))
acf(gibbssim$par$sigma2[,2], main=paste("sigma_2",sep=""))
####For real data
cad = read.csv("http://www.rob-mcculloch.org/data/calhouse.csv")
x=cad$longitude
plot(density(x),main="longitude ")
gibbsreal=gibbsmixnorm(x, k=2, keep=100, gap=10, burn=50,seed=1024)
print(gibbsreal$summary)
####draw the plot
par(mfrow=c(1, 3))
plot(gibbsreal$par$p[,1], main=paste("p_1",sep=""),type="l")
plot(gibbsreal$par$p[,2], main=paste("p_2",sep=""),type="l")
plot(gibbsreal$par$mu[,1], main=paste("mu_1",sep=""),type="l")
plot(gibbsreal$par$mu[,2], main=paste("mu_2",sep=""),type="l")
plot(gibbsreal$par$sigma2[,1], main=paste("sigma_1",sep=""),type="l")
plot(gibbsreal$par$sigma2[,2], main=paste("sigma_2",sep=""),type="l")
###draw the acf
par(mfrow=c(1, 3))
acf(gibbsreal$par$p[,1], main=paste("p_1",sep=""))
acf(gibbsreal$par$p[,2], main=paste("p_2",sep=""))
acf(gibbsreal$par$mu[,1], main=paste("mu_1",sep=""))
acf(gibbsreal$par$mu[,2], main=paste("mu_2",sep=""))
acf(gibbsreal$par$sigma2[,1], main=paste("sigma_1",sep=""))
acf(gibbsreal$par$sigma2[,2], main=paste("sigma_2",sep=""))

#----------------- Comparison plot of EM and Gibbs------------------------
par(mfrow=c(2,2))
par(mar  =c(3,4,2,1))

# real (true mu1 = -118.0571)
plot(density(b$par$mu[,2]), ylab="Density of mu1",
    main="Gibbs Sampler Simulation for the real data")
abline(v=mean(b$par$mu[,2]), col = "red") # Gibbs=-118.0778
```

```
abline(v= -118.0773, col="blue") # EM
legend('topright', legend=c("Gibbs","EM"),lwd=2,col=c("red","blue"))#pch=c(19,21

# real (true mu2 = -121.8482)
plot(density(b$par$mu[,1]), ylab="Density of mu2",
    main="Gibbs Sampler Simulation for the real data")
abline(v=mean(b$par$mu[,1]), col = "red") # Gibbs=-121.8679
abline(v= −121.8667, col="blue") # EM
legend('topright', legend=c("Gibbs","EM"),lwd=2,col=c("red","blue"))

# simulated (true mu1 = 15)
plot(density(a$par$mu[,1]), ylab="Density of mu1",
    main="Gibbs Sampler Simulation for the simulated data")
abline(v=mean(a$par$mu[,1]), col = "red") # Gibbs=14.91693
abline(v=14.90974, col="blue")  # EM
legend('topright', legend=c("Gibbs","EM"),lwd=2,col=c("red","blue"))

# simulated (true mu2 = 33)
plot(density(a$par$mu[,2]), ylab="Density of mu2",
    main="Gibbs Sampler Simulation for the simulated data")
abline(v=mean(a$par$mu[,2]), col = "red") # Gibbs=33.27526
abline(v=33.27854, col="blue")  # EM
legend('topright', legend=c("Gibbs","EM"),lwd=2,col=c("red","blue"))
```