STP540 Final Project

Mikayla Geiger

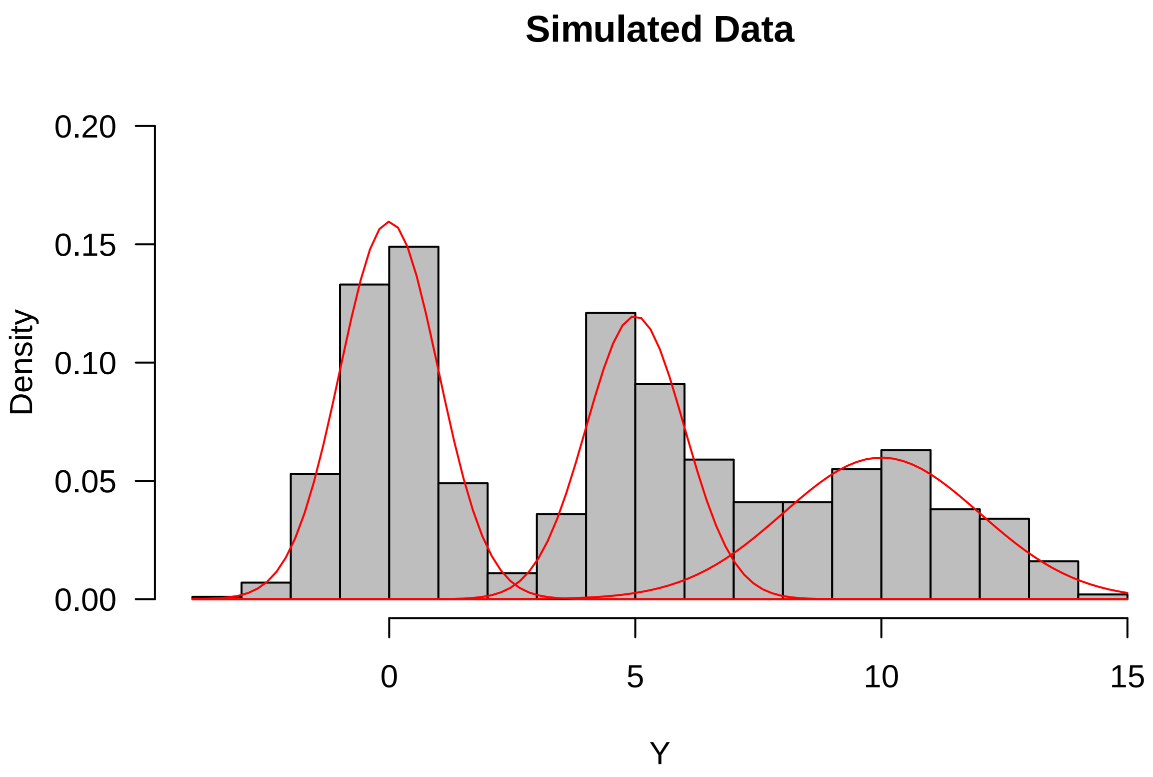
4/30/2021

Introduction

For this project, I will do a mixture model using the EM algorithm and the Gibbs Sampler and compare the results the two give me. I will start with simulated data to confirm my code works and then move on to real data.

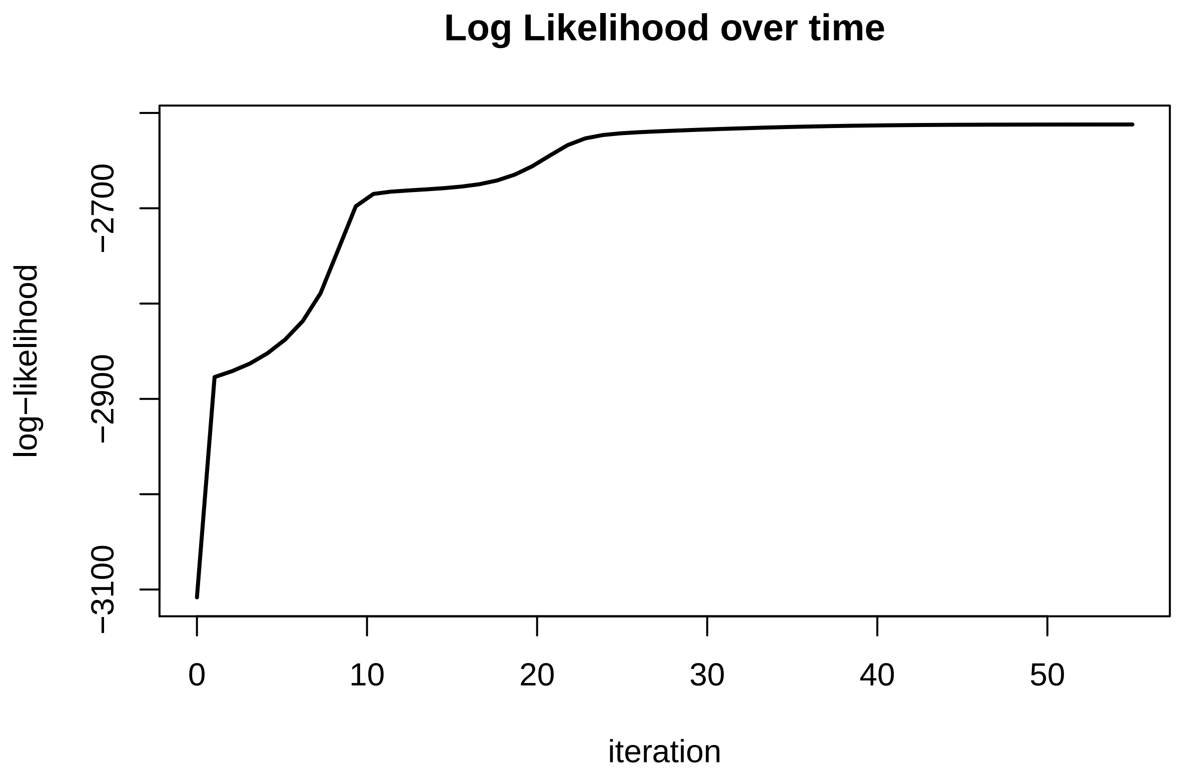
Simulated Data

For my simulated data I set up a 3 component model the probability vector is $p = (.4, .3, .3)$, the means are $\mu = (0, 5, 10)$ and the standard deviation is $\sigma = (1, 1, 2)$. A histogram of the data is shown below. Along with the densities of the three components



EM Algorithm on Simulated Data

The code for my EM algorithm is set up to stop once the change in log-likelihood is less than or equal to 0.01. Below is a plot of the log-likelihood over time, which shows that it flattens out for some time before the algorithm finally ends.



This EM algorithm gave the following values, which we can see are close to the true values.

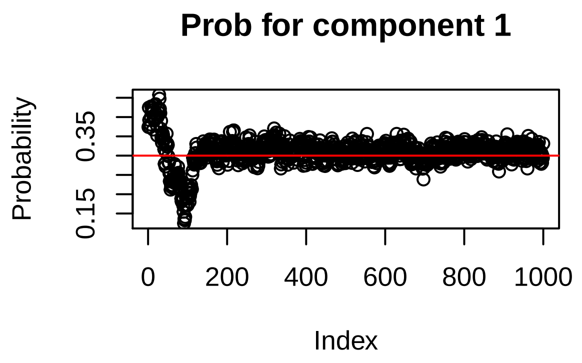
$p = (0.30809, 0.29263, 0.39927)$

$\mu = (9.87847, 4.97949, -0.00464)$

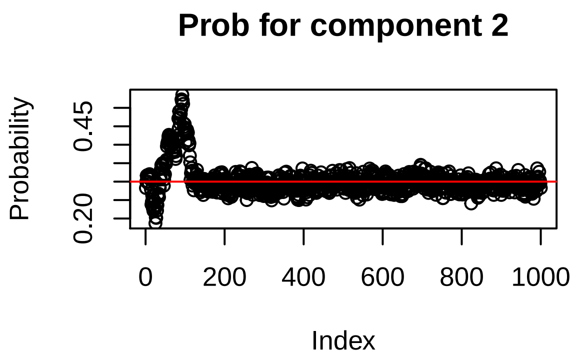
$\sigma = (1.99953, 0.96953, 0.97005)$

Gibbs Sampler

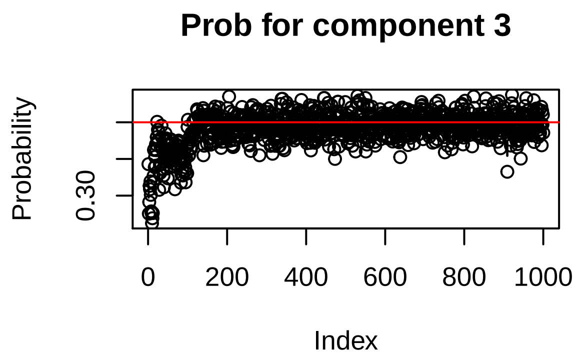
My Gibbs Sampler is set to run through 1000 draws. Below are the graphs for each variable and each component.



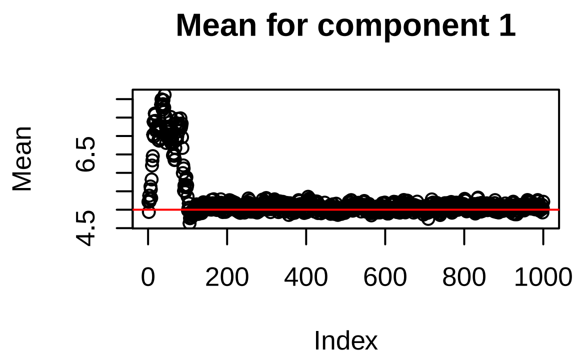
Prob for component 1
Probability
0.15
0.35
0
200
400
600
800
1000
Index

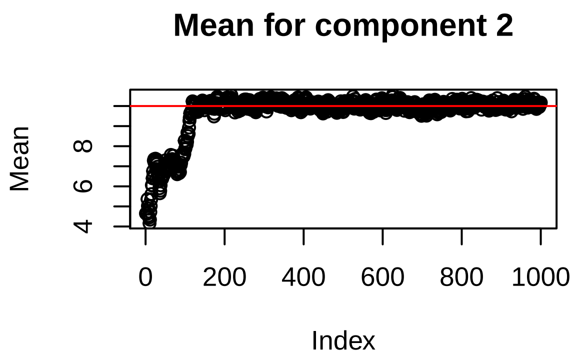


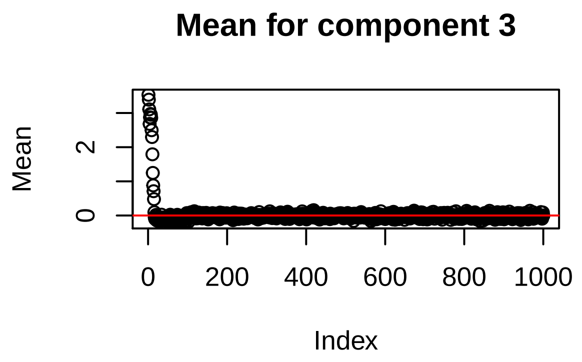
Prob for component 2
Probability
0.20
0.45
0
200
400
600
800
1000
Index

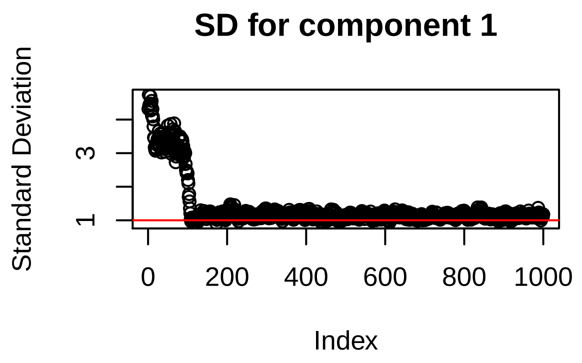


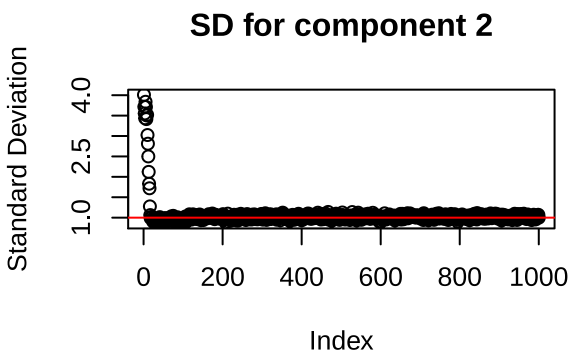
Prob for component 3
Probability
0.30
0
200
400
600
800
1000
Index

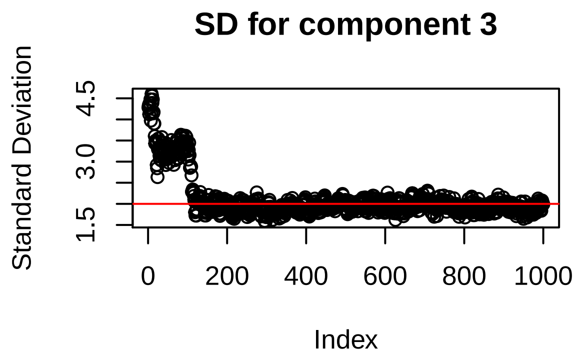

Mean for component 1
Mean
4.5
6.5
0
200
400
600
800
1000
Index


Mean for component 2
Mean
4
6
8
0
200
400
600
800
1000
Index


Mean for component 3
Mean
0
2
0
200
400
600
800
1000
Index







The first 50-100 observations are burn in and then they are right around the true values which are shown in red. I will discard the first 100 observations and take the mean of the rest. The results are shown below. Again, these are pretty close to the true values.

$p = (0.30864, 0.29498, 0.39638)$

$\mu = (5.06124, 10.02377, -0.00786)$

$\sigma = (1.14001, 1.94982, 1.0189)$

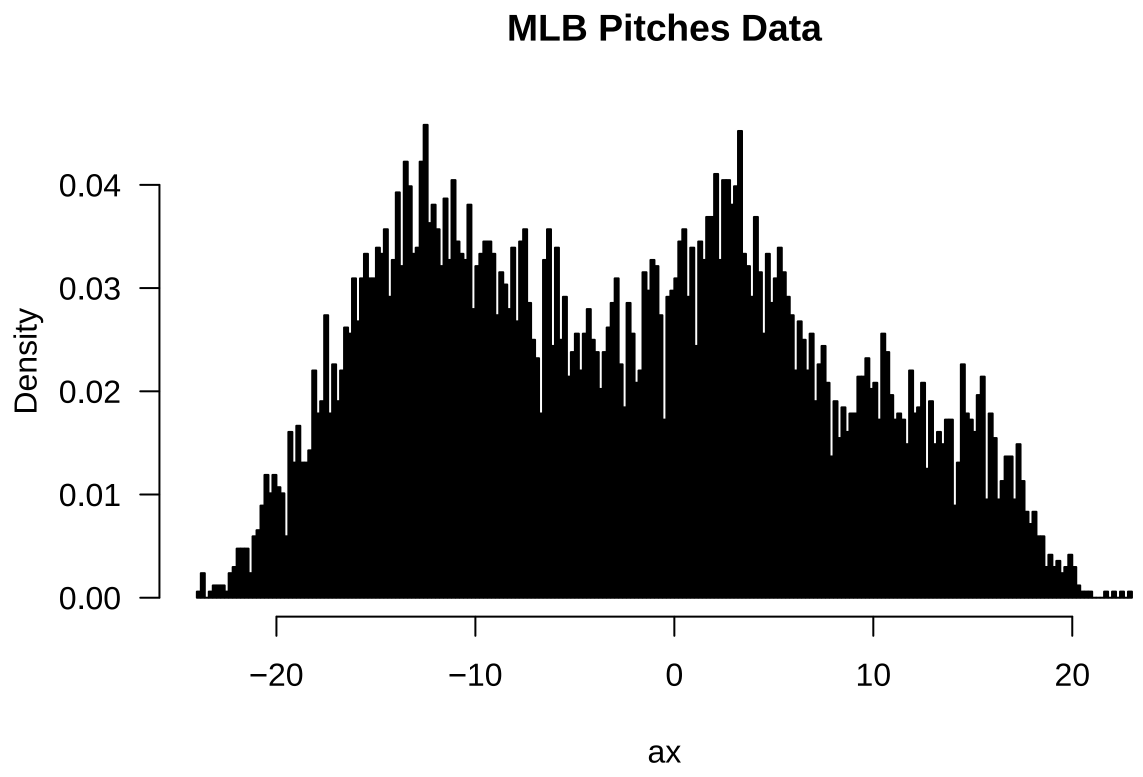
For the last part of the simulated data, there is a table comparing both algorithms and the true vales. Both perform pretty well and are close to the real values. On some the Gibbs Sampler is a little bit closer and on others the EM algorithm is closer. Overall they both perform pretty well though.

	True_Value	EM	Gibbs
p1	0.4	0.3992726	0.3963785
p2	0.3	0.2926327	0.3086426
p3	0.3	0.3080947	0.2949789
mu1	0.0	-0.0046404	-0.0078642
mu2	5.0	4.9794942	5.0612363
mu3	10.0	9.8784684	10.0237684
sigma1	1.0	0.9700491	1.0188965
sigma2	1.0	0.9695317	1.1400075
sigma3	2.0	1.9995257	1.9498224

Real Data

The data I will be using is 2019 MLB pitch data, which the owner scraped from the web (Schale,2020). The original dataset contains radar gun information for all pitches across an entire MLB season, and has information for over 700,000 pitches. Due to the fact that I wanted to run both algorithms, and try running them with varying number of components I had to scale this back considerably for time purposes. I chose to take two day's worth of pitches. The data I will be using is for all games on 9/1/2019-9/2/2019. There are now 8,403 pitches in this dataset.

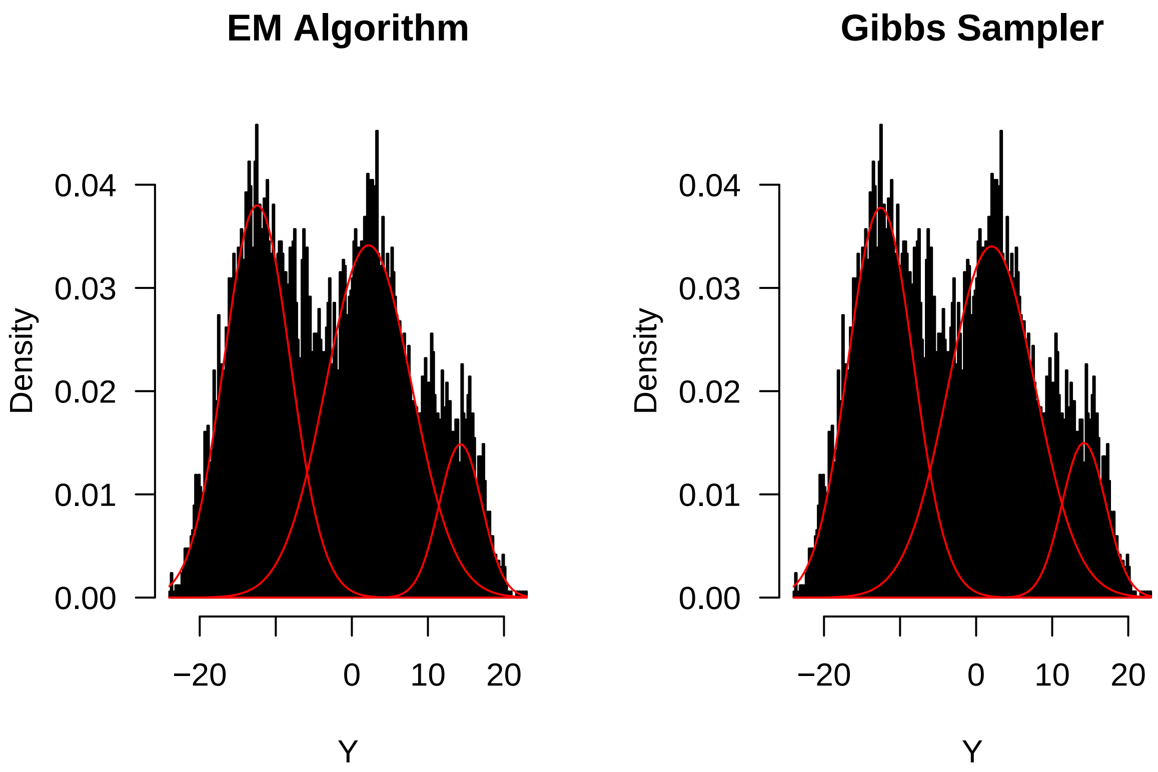
There are about thirty different variables in this dataset that could be used for this. I am just going to choose one to look at in this report. I'm going to use the ax variable, which measures the acceleration of the pitch in the horizontal direction 50 feet before home plate. (Woods, 2021). Below is the histogram of the data.



At first glance, it looks like there are about 3 components, so I will start by running both the EM Algorithm and Gibbs Sampler with 3 components. Below is a table with the results from both. As well as a side by side plot of the densities each algorithm produces.

	EM	Gibbs
p1	0.4137423	0.4068192
p2	0.4812734	0.4849036
p3	0.1049843	0.1082771
mu1	-12.4467679	-12.5378893
mu2	2.2097695	2.0409779
mu3	14.2933095	14.1746140
sigma1	4.3393009	4.2933035
sigma2	5.6224617	5.6826238

	EM	Gibbs
sigma3	2.8147921	2.8772466



So both algorithms gave pretty similar results and it looks like the models are a pretty good fit for the data. For my code I have to manually tell it how many components to use. (This is something I could improve on in the future by having the algorithm run for multiple components and picking the one with the lowest BIC. But for now, it is manual.) I wanted to see what R said the optimal number of components are so I used the mclust package and got the following results.

```
## ----------------------------------------------------
## Density estimation via Gaussian finite mixture modeling
## ----------------------------------------------------
##
## Mclust E (univariate, equal variance) model with 4 components:
##
##  log-likelihood    n df       BIC       ICL
##       -30811.72 8403  8 -61695.73 -64995.99
```

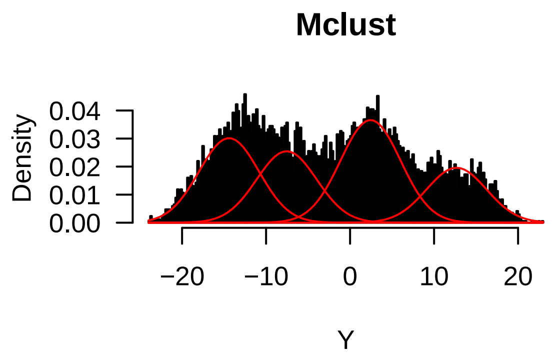
According to this, four components is actually the optimal amount. So I will run my code again using four components and compare the results of my code with with the mclust package. Again there is a table with the results from all three.

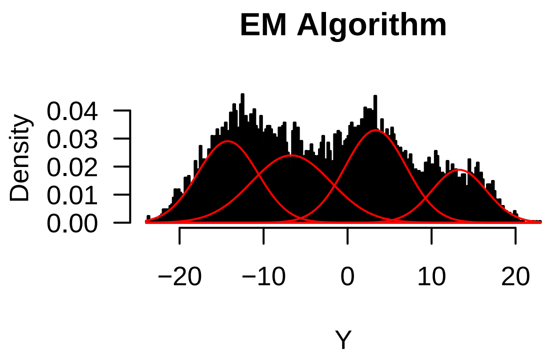
	Mclust	EM	Gibbs
p1	0.2696319	0.2618866	0.2451578
p2	0.2278321	0.2838230	0.2229564
p3	0.3273958	0.3005254	0.3968554

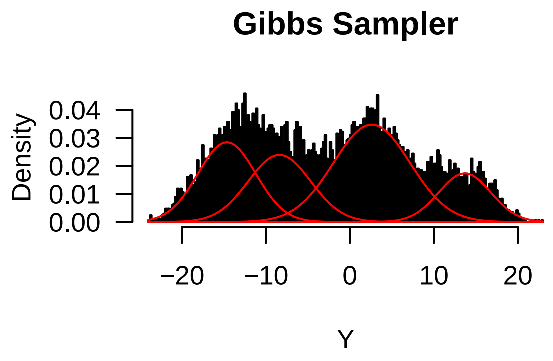
	Mclust	EM	Gibbs
p4	0.1751402	0.1537650	0.1350303
mu1	-14.4063917	-14.2535805	-14.6393480
mu2	-7.5456294	-6.7045555	-8.3781606
mu3	2.4383232	3.3541773	2.6068767
mu4	12.6731924	13.2802446	13.6825065
sigma1	3.5671067	3.5914547	3.4388817
sigma2	3.5671067	4.7140058	3.7174238
sigma3	3.5671067	3.6314661	4.5588164
sigma4	3.5671067	3.2460754	3.1048373

It does look like all three vary quite a bit especially in the probabilities, but they all are around the same values. It is interesting to note that mclust assigned the same standard deviation to all components. My codes both came up with similar standard deviations to this one, but they were not all the same.

Below are graphs with the densities from all three algorithms. Looking at the graphs confirms that the results are very similar.







Finally, I am going to assign each value to the correct component Because the results were pretty similar I'm just going to use one result, which is the four component Gibb Sampler. I will calculate the probability that each value is in a particular component, and then assign it to whichever component has the highest probability. Below is how many elements were assigned to each component and what percentage of the total data that is. We can see that these percentages are pretty close to the probabilities we got from the Gibbs Sampler.

Component 1	Component 2	Component 3	Component 4
2142.000000	1817.0000000	3298.0000000	1146.0000000
0.254909	0.2162323	0.3924789	0.1363799

Once we have these componenets, a practical application would be trying to see if there is some characteristic that pitches in the same component have in common. You could look at alot of things here like whether it was called a ball or strike, whether the batter got a hit off of it, or maybe the total speed of the pitch.

What I'm going to look at here is the pitch type. There are nine different pitch types that were thrown over the two days of my data. They are defined with the following abbreviations (Schale,2020):
CH: Changeup

CU: Curveball

FC: Cutter

FF: Four-seam Fastball

FS: Splitter

FT: Two-seam Fastball

KC: Knuckle curve

SI: Sinker

SL: Slider

What the table below shows is the percentage of each pitch that is in each component. So for example, there were 833 total changeups thrown over the two days. 361 of those were assigned to component 1. So component 1 contains approximately 43% of the total change ups thrown.

	CH	CU	FC	FF	FS	FT	KC	SI	SL
c1	0.4333733	0.0081855	0.0033557	0.2470667	0.4081633	0.7198391	0.0000000	0.6005548	0.0040027
c2	0.2220888	0.0859482	0.0721477	0.4203822	0.4829932	0.0563003	0.1379310	0.0499307	0.0687125
c3	0.0744298	0.7680764	0.9228188	0.2065035	0.1020408	0.0093834	0.7931034	0.0291262	0.8999333
c4	0.2701080	0.1377899	0.0016779	0.1260476	0.0068027	0.2144772	0.0689655	0.3203883	0.0273516

One thing that stands out in this table is component 3. This show that it contains the vast majority of cutters and sliders. So if someone wanted to know more about the acceleration of the pitch in the horizontal direction 50 feet before home plate for most cutters and sliders, component 3 should be a good representation of how these two pitches behave.

There's obviously alot more that can be done with not only these results, but also this data. This is just a small and simple example to show that my algorithms do work, and then how one could use these results in a practical way.

Appendix

References

Schale, Paul. MLB Pitch Data 2015-2018. Kaggle. 2020. Retrieved from https://www.kaggle.com/pschale/mlb-pitch-data-20152018

Woods, James. "Radar Measurement Glossary of Terms." Trackman. 2021. Retrieved from https://trackman.zendesk.com/hc/en-us/articles/115002776647-Radar-Measurement-Glossary-of-Terms

Code

Gibbs Sampler

```r
## Note technically here we may also want to take in initial values for tau, alpha, and nu
## Becuase I just left them the same the whole time I did not give that option.

gibbs <- function(y,k){
library(DirichletReg)


## Draw p from a dirichlet
drawp=function(I, alpha){
  counts=as.vector(table(factor(I, levels = 1:length(alpha))))
  palpha=(alpha+counts)
  return(rdirichlet(1,alpha=palpha))
}

## Draw I's
drawi=function(y,I,p,mu,sigma){
  pI=rep(0,length(y))
  for(i in 1:length(y)){
    pI[i]=sample(seq(from=1,to=length(mu),length=length(mu)),size=1,prob=p*dnorm(y[i],mu,sigma))
  }
  return(pI)
}


## Draw mus
drawmu=function(y,sigma,mubar,tau,I)
{
  mu=rep(0,length(sigma))
  for(i in 1:length(sigma)){
    yi=y[which(I==i)]
    n=length(yi)
    a=n/sigma[i]^2
    b=1/tau^2
    ybar=mean(yi)
    pmu=(a*ybar+b*mubar)/(a+b)
    psigma=sqrt(1/(a+b))
    mu[i]=rnorm(1,pmu,psigma)
  }
  return(mu)
}
## Draw sigmas
drawsigma=function(y,mu,nu,lambda,I)
{
  sigma=rep(0,length(mu))
  for(i in 1:length(mu)){
    yi=y[which(I==i)]
    n=length(yi)
    S=sum((yi-mu[i])^2)
    pnu=nu+n
    plambda=nu*lambda+S
    sigma[i]=sqrt(plambda/rchisq(1,pnu))
  }
  return(sigma)
}

ndraws = 1000
pv=matrix(0,ncol=k,nrow=ndraws)
Iv=matrix(0,ncol=length(y),nrow=ndraws)
muv = matrix(0,ncol=k,nrow=ndraws)
sigv = matrix(0,ncol=k,nrow=ndraws)

#Initial values
alpha=rep(5,k)
I=sample(seq(from=1,to=k,length=k),size=length(y),replace=TRUE)
mu=y[c(runif(k,0,length(y)))]
sigma=rep(sd(y),k)
mubar=mean(mu)
tau=10
nu=5
lambda =3^2

for(i in 1:ndraws) {
  #Draw p
  p=drawp(I, alpha)

  #Draw I
  I=drawi(y,I,p,mu,sigma)

  #Draw mu
  mu=drawmu(y,sigma,mubar,tau,I)

  #Draw sigma
  sigma = drawsigma(y,mu,nu,lambda,I)

  pv[i,]=p
  Iv[i,]=I
  muv[i,] = mu
  sigv[i,]= sigma
}
return(list(pv=pv,muv=muv,sigv=sigv))
}
```

EM Algorithm

```
em <-function(y,j){
p=rep(1/j,length=j) #Intial p sets all weights equal

mu=y[c(runif(j,0,length(y)))] #Choose random values of y for the mean

sigma=sqrt(rep(var(y),j)) #Start with variances equal to the sample variance

#set up matrices where each row is a duplicate of the above vectors. This will
#allow calculations to be done without a for loop.
Y=matrix(y,nrow=length(y),ncol=j)
M=matrix(mu,nrow=length(y),ncol=j,byrow=TRUE)
S=matrix(sigma,nrow=length(y),ncol=j,byrow=TRUE)
P=matrix(p,nrow=length(y),ncol=j,byrow=TRUE)

#The initial log-likelihood
ll=sum((log(rowSums(P*dnorm(Y,M,S)))))

#empty alpha matrix to be filled in in each iteration
alpha=matrix(0,nrow=length(y),ncol=j)

#vector to track change in log likelihood
logl=c(0,ll)

#Take the difference betwen the current log-likelihood and the previous log likelihood.
#We will stop the algorithm once the difference is less than or equal to 0.01
while(abs(logl[length(logl)-1]-ll) > 0.01){
  ##E step
  py=rowSums(P*dnorm(Y,M,S))
  alpha=P*dnorm(Y,M,S)/py

  ## M step
  p=colSums(alpha)/length(y)
  mu=colSums(alpha*y)/colSums(alpha)

  #Again update matrices to do calcualtion of sigma and log likelihood
  M=matrix(mu,nrow=length(y),ncol=j,byrow=TRUE)
  P=matrix(p,nrow=length(y),ncol=j,byrow=TRUE)
  sigma=sqrt(colSums(alpha*(Y-M)^2)/colSums(alpha))

  #Update Sigma matrix
  S=matrix(sigma,nrow=length(y),ncol=j,byrow=TRUE)

  #Calculate log likelihood for current step
  ll=sum((log(rowSums(P*dnorm(Y,M,S)))))
  logl=append(logl,ll)
  }
    return(list(p=p,mu=mu,sigma=sigma,logl=logl))
}
```

Simulated Data

```
## SIMULATE DATA
probs=c(.4,.3,.3)
means=c(0,5,10)
sigmas=c(1,1,2)
n=1000
set.seed(2465)
y1=means[1] + sigmas[1]*rnorm(n*probs[1])
y2=means[2] + sigmas[2]*rnorm(n*probs[2])
y3=means[3] + sigmas[3]*rnorm(n*probs[3])
y=c(y1,y2,y3)
hist(y,probability= TRUE,main="Simulated Data",
     xlab="Y",
     border="black",
     col="gray",
     las=1,
     breaks=15,
     ylim=c(0,0.20))
curve(probs[1]*dnorm(x,mean=means[1],sd=sigmas[1]), add=TRUE,col="red")
curve(probs[2]*dnorm(x,mean=means[2],sd=sigmas[2]), add=TRUE,col="red")
curve(probs[3]*dnorm(x,mean=means[3],sd=sigmas[3]), add=TRUE,col="red")
```

Algorithms on simulated data

```
## em
emsim=em(y,3)
emp=emsim$p
emmu=emsim$mu
emsigma=emsim$sigma
eml=emsim$logl

plot(x=seq(from=0,to=length(emsim$logl),length=length(emsim$logl)-1),y=emsim$logl[-1],
     type='l',lwd=2,
xlab="iteration",ylab="log-likelihood", main="Log Likelihood over time")

## Gibbs

gibbsim=gibbs(y,3)
pv=gibbsim$pv
muv=gibbsim$muv
sigv=gibbsim$sigv

par(mfrow = c(2, 2))
p1=plot(pv[,1],type='b', ylab="Probability", main="Prob for component 1")
abline(h=probs[2],col="red", lwd=1)
p2=plot(pv[,2],type='b', ylab="Probability", main="Prob for component 2")
abline(h=probs[3],col="red", lwd=1)
p3=plot(pv[,3],type='b', ylab="Probability", main="Prob for component 3")
abline(h=probs[1],col="red", lwd=1)

par(mfrow = c(2, 2))
p3=plot(muv[,1],type='b', ylab="Mean", main="Mean for component 1")
abline(h=means[2],col="red", lwd=1)
```

```r
p4=plot(muv[,2],type='b', ylab="Mean", main="Mean for component 2")
abline(h=means[3],col="red", lwd=1)
p5=plot(muv[,3],type='b', ylab="Mean", main="Mean for component 3")
abline(h=means[1],col="red", lwd=1)

par(mfrow = c(2, 2))
p6=plot(sigv[,1],type='b', ylab="Standard Deviation", main="SD for component 1")
abline(h=sigmas[1],col="red", lwd=1)
p7=plot(sigv[,3],type='b', ylab="Standard Deviation", main="SD for component 2")
abline(h=sigmas[2],col="red", lwd=1)
p8=plot(sigv[,2],type='b', ylab="Standard Deviation", main="SD for component 3")
abline(h=sigmas[3],col="red", lwd=1)

estp=colMeans(pv[101:1000,])
estm=colMeans(muv[101:1000,])
ests=colMeans(sigv[101:1000,])

tv=c(probs,means,sigmas)
emres=c(emp[3],emp[2],emp[1],emmu[3],emmu[2],emmu[1],emsigma[3],emsigma[2],emsigma[1])
gibres=c(estp[3],estp[1],estp[2],estm[3],estm[1],estm[2],ests[3],ests[1],ests[2])
results=data.frame(tv,emres,gibres,
                   row.names=c("p1","p2","p3", "mu1","mu2","mu3","sigma1",
                                "sigma2","sigma3"))
colnames(results)=c("True_Value","EM", "Gibbs")
library(knitr)
kable(results)
```

Real Data

```r
library(plyr)

## Read in three datasets
pitches=read.csv("C:/Users/mickv/OneDrive/Documents/2019_pitches.csv", header=TRUE)
ab=read.csv("C:/Users/mickv/OneDrive/Documents/2019_atbats.csv", header=TRUE)
games=read.csv("C:/Users/mickv/OneDrive/Documents/2019_games.csv", header=TRUE)

## Join the three to get the dates of the pitches
newpitches=join(pitches,ab,by='ab_id')
newpitches=join(newpitches,games,by='g_id')

## Filter for the dates I want
newpitches$dates=as.Date(as.character(newpitches$date),"%Y-%m-%d")
newpitches=newpitches[newpitches$dates > "2019-08-31",]
newpitches=newpitches[newpitches$dates < "2019-09-03",]

## Keep only columns that were in the original pitches dataset,
## and drop the ones that don't contain data. also get rid of NAs
newpitches<- subset(newpitches, select = as.vector(colnames(pitches)))
newpitches <- subset(newpitches, select = -c(spin_rate,spin_dir,type_confidence,nasty))
newpitches <-na.omit(newpitches)

## Vairable to look at
y=newpitches$ax
```

```r
h2=hist(y,probability= TRUE,main="MLB Pitches Data",
     xlab="ax",
     border="black",
     col="black",
     las=1,
     breaks=250)
```

3 Components

```r
em3=em(y,3)
emp3=em3$p
emmu3=em3$mu
emsigma3=em3$sigma
eml=em3$logl

gibbs3=gibbs(y,3)
pv3=gibbs3$pv
muv3=gibbs3$muv
sigv3=gibbs3$sigv

estp=colMeans(pv3[101:1000,])
estm=colMeans(muv3[101:1000,])
ests=colMeans(sigv3[101:1000,])

emres=c(emp3[2],emp3[1],emp3[3],emmu3[2],emmu3[1],emmu3[3],emsigma3[2],emsigma3[1],emsigma3[3])
gibres=c(estp,estm,ests)
results=data.frame(emres,gibres,
                   row.names=c("p1","p2","p3", "mu1","mu2","mu3","sigma1","sigma2","sigma3"))
colnames(results)=c("EM", "Gibbs")

kable(results)

par(mfrow = c(1, 2))
h1=hist(y,probability= TRUE,main="EM Algorithm",
     xlab="Y",
     border="black",
     col="black",
     las=1,
     breaks=250)
curve(emp3[1]*dnorm(x,mean=emmu3[1],sd=emsigma3[1]), add=TRUE,col="red")
curve(emp3[2]*dnorm(x,mean=emmu3[2],sd=emsigma3[2]), add=TRUE,col="red")
curve(emp3[3]*dnorm(x,mean=emmu3[3],sd=emsigma3[3]), add=TRUE,col="red")
h2=hist(y,probability= TRUE,main="Gibbs Sampler",
     xlab="Y",
     border="black",
     col="black",
     las=1,
     breaks=250)
curve(estp[1]*dnorm(x,mean=estm[1],sd=ests[1]), add=TRUE,col="red")
curve(estp[2]*dnorm(x,mean=estm[2],sd=ests[2]), add=TRUE,col="red")
curve(estp[3]*dnorm(x,mean=estm[3],sd=ests[3]), add=TRUE,col="red")
```

4 components

```r
library(mclust)
modsim = densityMclust(y)
summary(modsim)

em4=em(y,4)
emp4=em4$p
emmu4=em4$mu
emsigma4=em4$sigma
eml=em4$logl

gibbs4=gibbs(y,4)
pv4=gibbs4$pv
muv4=gibbs4$muv
sigv4=gibbs4$sigv

mvf = modsim$parameters$mean
svf = rep(sqrt(modsim$parameters$variance$sigmasq),4)
pvf = modsim$parameters$pro

estp=colMeans(pv4[101:1000,])
estm=colMeans(muv4[101:1000,])
ests=colMeans(sigv4[101:1000,])

mc=c(pvf,mvf,svf)
emres=c(emp4[4],emp4[2],emp4[1],emp4[3],emmu4[4],emmu4[2],emmu4[1],
        emmu4[3],emsigma4[4],emsigma4[2],emsigma4[1],emsigma4[3])
gibres=c(estp[3],estp[1],estp[4],estp[2],estm[3],estm[1],estm[4],
         estm[2],ests[3],ests[1],ests[4],ests[2])
results=data.frame(mc,emres,gibres,
                   row.names=c("p1","p2","p3","p4", "mu1","mu2","mu3","mu4",
                               "sigma1","sigma2","sigma3", "sigma4"))
colnames(results)=c("Mclust","EM", "Gibbs")
kable(results)

par(mfrow = c(2, 2))
h1=hist(y,probability= TRUE,main="Mclust",
     xlab="Y",
     border="black",
     col="black",
     las=1,
     breaks=250)
curve(pvf[1]*dnorm(x,mean=mvf[1],sd=svf), add=TRUE,col="red")
curve(pvf[2]*dnorm(x,mean=mvf[2],sd=svf), add=TRUE,col="red")
curve(pvf[3]*dnorm(x,mean=mvf[3],sd=svf), add=TRUE,col="red")
curve(pvf[4]*dnorm(x,mean=mvf[4],sd=svf), add=TRUE,col="red")
h2=hist(y,probability= TRUE,main="EM Algorithm",
     xlab="Y",
     border="black",
     col="black",
     las=1,
     breaks=250)
curve(emp4[1]*dnorm(x,mean=emmu4[1],sd=emsigma4[1]), add=TRUE,col="red")
curve(emp4[2]*dnorm(x,mean=emmu4[2],sd=emsigma4[2]), add=TRUE,col="red")
curve(emp4[3]*dnorm(x,mean=emmu4[3],sd=emsigma4[3]), add=TRUE,col="red")
curve(emp4[4]*dnorm(x,mean=emmu4[4],sd=emsigma4[4]), add=TRUE,col="red")
h3=hist(y,probability= TRUE,main="Gibbs Sampler",
     xlab="Y",
     border="black",
     col="black",
     las=1,
     breaks=250)
curve(estp[1]*dnorm(x,mean=estm[1],sd=ests[1]), add=TRUE,col="red")
curve(estp[2]*dnorm(x,mean=estm[2],sd=ests[2]), add=TRUE,col="red")
curve(estp[3]*dnorm(x,mean=estm[3],sd=ests[3]), add=TRUE,col="red")
curve(estp[4]*dnorm(x,mean=estm[4],sd=ests[4]), add=TRUE,col="red")
```

Asignments

```r
## Just reordering so components match above
gibbp=c(estp[3],estp[1],estp[4],estp[2])
gibbm=c(estm[3],estm[1],estm[4],estm[2])
gibbs=c(ests[3],ests[1],ests[4],ests[2])

newpitches$prob=rep(0,nrow(newpitches))
newpitches$comp=rep(0,nrow(newpitches))
for(i in 1:nrow(newpitches)){
  probabilities=gibbp*dnorm(y[i],gibbm,gibbs)
  newpitches$prob[i]=max(probabilities)
  newpitches$comp[i]=as.integer(which.max(probabilities))
}

comp1=newpitches[newpitches$comp==1,]
comp2=newpitches[newpitches$comp==2,]
comp3=newpitches[newpitches$comp==3,]
comp4=newpitches[newpitches$comp==4,]
c1=c(nrow(comp1),nrow(comp1)/nrow(newpitches))
c2=c(nrow(comp2),nrow(comp2)/nrow(newpitches))
c3=c(nrow(comp3),nrow(comp3)/nrow(newpitches))
c4=c(nrow(comp4),nrow(comp4)/nrow(newpitches))
results=data.frame(c1,c2,c3,c4)
colnames(results)=c("Component 1","Component 2", "Component 3","Component 4")
kable(results)

type=as.vector(table(newpitches$pitch_type))
ctype1=as.vector(table(comp1$pitch_type))
ctype2=as.vector(table(comp2$pitch_type))
ctype3=as.vector(table(comp3$pitch_type))
ctype4=as.vector(table(comp4$pitch_type))

perc1=ctype1/type
perc2=ctype2/type
perc3=ctype3/type
perc4=ctype4/type

## Note, I had to get rid of the pitch types that we in the original data set, but never thrown
## over the two days I selected.

types=as.data.frame(rbind(perc1[c(2,3,5,6,8,9,10,12,13)],perc2[c(2,3,5,6,8,9,10,12,13)],
                          perc3[c(2,3,5,6,8,9,10,12,13)],perc4[c(2,3,5,6,8,9,10,12,13)]),
                    row.names=c("c1","c2","c3","c4"))
colnames(types)=c("CH", "CU", "FC", "FF", "FS", "FT", "KC", "SI", "SL")
kable(types)
```