Homework 10

Gustavo Esparza

5/11/2019

1

Libraries Used:

```
library(readr)
library(ggplot2)
```

Import data:

```
#Import Data
data= read.table("/Users/gustavo/Desktop/MATH 534/14/Coal Data.txt", header=TRUE)
data=data$disasters
```

Generate Priors

For the SIR algorithm, we must first draw samples of θ, λ_1 and λ_2 from their respective prior functions. An m of size 100,000 was chosen to estimate the population of our data. From this m, we have priors defined by:

$$\theta \sim DiscreteUNIF(1, 2, ..., 111)$$

$$\lambda_i | a_i \sim Gamma(3, a_i), \text{ where } a_i \sim Gamma(10, 10), i = 1, 2$$

Then, our priors are generated as follows:

```
m = 80000

theta = sample(1:111, m, replace=T) #theta prior

a1 = rgamma(m,shape=10, rate=10) #independent a1
lambda1 = rgamma(m, shape=3, rate=a1) #lambda1 prior

a2 = rgamma(m,shape=10,rate=10)  #independent a2
lambda2 = rgamma(m,shape=3,rate=a2) #lambda2 prior
```

STANDARDIZED IMPORTANCE SAMPLING WEIGHTS

Now that our priors have been generated, we can now define our Sampling Importance Weights. Having generated our priors, $\pi(\theta, \lambda_1, \lambda_2)$, we can define our Weights as the Likelihood of our priors, $L(\theta, \lambda_1, \lambda_2)$. We have been provided the following information for our population:

$$X_1, ..., X_\theta \sim Poisson(\lambda_1)$$
$$x_{\theta+1}, ..., X_{112} \sim Poisson(\lambda_2)$$

Then, our Likelihood function is defined as follows:

$$L(\theta, \lambda_1, \lambda_2) = \prod_{i=1}^{\theta} \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x_i!} \prod_{i=\theta+1}^{112} \frac{e^{-\lambda_2}\lambda_2^{x_i}}{x_i!} = \frac{\prod_{i=1}^{\theta} e^{-\lambda_1}\lambda_1^{x_i} \prod_{i=\theta+1}^{112} e^{-\lambda_2}\lambda_2^{x_i}}{\prod_{i=1}^{112} x_i!}$$

$$\propto \prod_{i=1}^{\theta} e^{-\lambda_1}\lambda_1^{x_i} \prod_{i=\theta+1}^{112} e^{-\lambda_2}\lambda_2^{x_i}$$

Thus, our likelihood function is reduced for the sake of computation, while still keeping the equation up to a constant.

Now, having defined our Likelihood, we can define our sampling weights as follows:

$$w(\theta, \lambda_1, \lambda_2) = \frac{L(\theta, \lambda_1, \lambda_2)}{\sum_{i=1}^{m} L(\theta_i, \lambda_{1i}, \lambda_{2i})}$$

In review of the paper written by James Albert, it appears that it is convenient to compute the log-likelihood in coding and then raise that value to the power of e in order to have the original Likelihood. Thus, we have the following Log-Likelihood function:

$$l(\theta, \lambda_1, \lambda_2) = -\theta\lambda_1 - (112 - \theta)\lambda_2 + \sum_{i=1}^{\theta} \log(\lambda_1)x_i + \sum_{i=\theta+1}^{112} \log(\lambda_2)x_i$$

```
#log-likelihood
loglike = function(data, theta, lambda1, lambda2){
sumtheta = c()
sum112 = c()

for(i in 1:m){

xtheta = data[1:theta[i]]
x112   = data[(theta[i]+1):112]

sumtheta = c(sumtheta, sum(xtheta))
sum112 = c(sum112, sum(x112))
}

return(-theta*lambda1 - (112-theta)*lambda2 +
      log(lambda1)*sumtheta + log(lambda2)*sum112)
}
```

```
llike = loglike(data, theta, lambda1, lambda2)

Likelihood = exp(llike) #Convert back to Likelihood

weights = Likelihood/sum(Likelihood)
```

Resampling

Now that we have our Sampling Weights, we can resample $\theta, \lambda_1, \lambda_2$ with replacement with our respective weights as the probabilities.

```
n = 15000
resample= sample(1:m, n, replace=TRUE, prob=weights)
#getting resampled values for each posterior

thetars= theta[resample]
lambda1rs= lambda1[resample]
lambda2rs= lambda2[resample]
```

Confidence Intervals & Histograms

Theta

Here is the Credible 95% Confidence Interval for Theta:

```
L = sort(thetars)[n*0.025]
U = sort(thetars)[n*0.975]

cat("Mean for Theta = ",mean(thetars),"")
```

```
## Mean for Theta =  39.4158
```

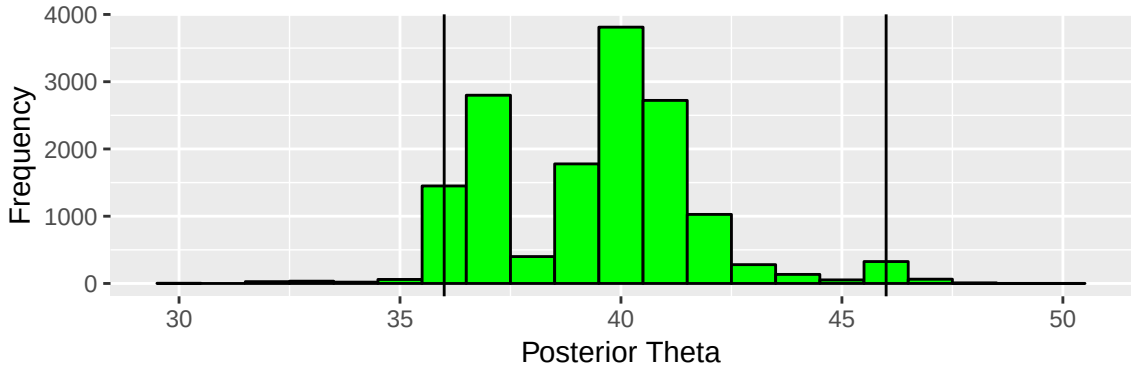
```
cat("\nCredible Confidence Interval for Theta = (",L,",",U,")")
```

```
##
## Credible Confidence Interval for Theta = ( 36 , 46 )
```

Here is the histogram for the resampled values of θ :

```
df = data.frame(x = thetars)

ggplot(df, aes(x = x)) +
geom_histogram(fill="green", color="black", binwidth=1)+
geom_vline(xintercept=L)+
geom_vline(xintercept=U)+
labs(x="Posterior Theta", y='Frequency')
```



LAMBDA 1

Here is the Credible 95% Confidence Interval for λ_1:

```
L = sort(lambda1rs)[n*0.025]
U = sort(lambda1rs)[n*0.975]
```

```
cat("Mean for Lambda 1 = ",mean(lambda1rs),"")
```

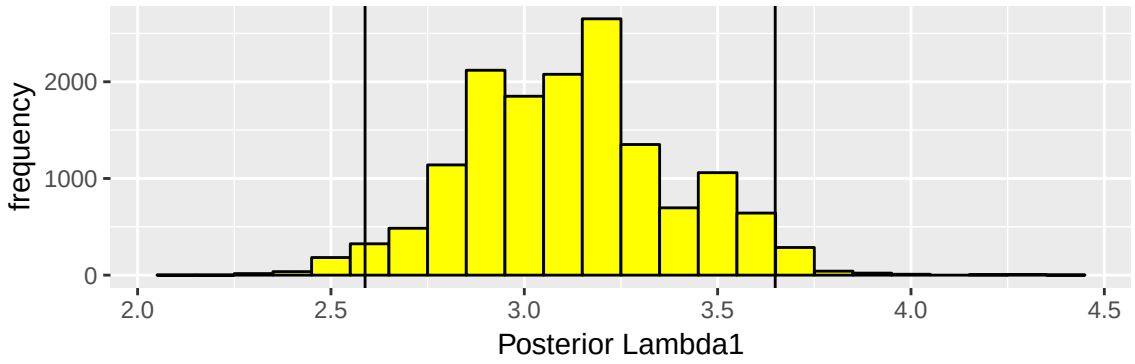
```
## Mean for Lambda 1 =  3.118406
```

```
cat("\nCredible Confidence Interval for Lambda 1 = (",L,",",U,")")
```

```
##
## Credible Confidence Interval for Lambda 1 = ( 2.5882 , 3.648834 )
```

Here is the histogram for the resampled values of λ_1 :

```
df = data.frame(x = lambda1rs)
ggplot(df, aes(x = x)) +
geom_histogram(fill="yellow", color="black",binwidth = .1)+
geom_vline(xintercept=L)+
geom_vline(xintercept=U)+
labs(x="Posterior Lambda1", y='frequency')
```



LAMBDA 2

Here is the Credible 95% Confidence Interval for λ_2:

```
L = sort(lambda2rs)[n*0.025]
U = sort(lambda2rs)[n*0.975]

cat("Mean for Lambda 2 = ",mean(lambda2rs),"")
```

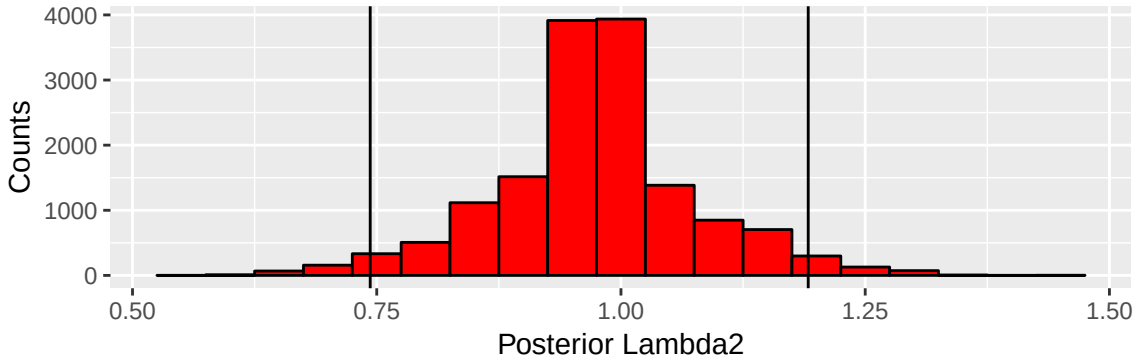
```
## Mean for Lambda 2 =  0.9719711
```

```
cat("\nCredible Confidence Interval for Lambda 2 = (",L,",",U,")")
```

```
##
## Credible Confidence Interval for Lambda 2 = ( 0.7432991 , 1.191683 )
```

Here is the histogram for the resampled values of λ_2 :

```
df = data.frame(x = lambda2rs)
ggplot(df, aes(x = x)) +
geom_histogram(fill="red", color="black",binwidth=.05)+
geom_vline(xintercept=L)+
geom_vline(xintercept=U)+
labs(x="Posterior Lambda2", y='Counts')
```



Utilizing the resampled values with weighted probabilities provided the previous three plots, which all follow similar shapes.

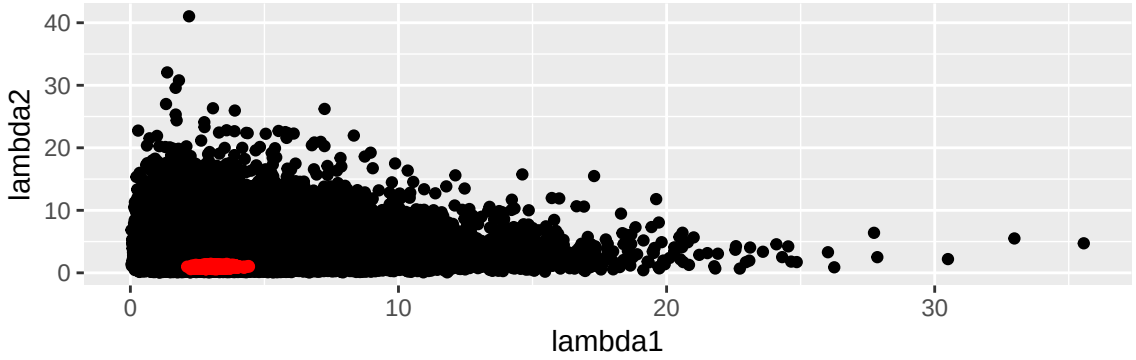
Scatter Plot of Lambda1 VS Lambda2

We will display the sampled and resampled values of λ_1 and λ_2 by highlighting the resampled values with red indicator marks.

```
#Sampled Values
df = data.frame(lambda1, lambda2)

#Resampled Values
dfrs = data.frame(lambda1rs, lambda2rs)

#Plot both Sampled and Resampled
ggplot(df, aes(x=lambda1, y=lambda2))+geom_point()+
geom_point(data=dfrs, aes(x=lambda1rs, y=lambda2rs), color="red")
```



We can see that the resampled points are densely located in the bottom left corner of our values, which come from the resampled posterior distributions.

Initial & Resampling Sizes

For the initial sample size, I chose 80000 because the sample size needed to be large enough to resemble an actual population size without getting differing results from resample to resample (n).

Number Of Unique Points & Highest Observed Frequency

Theta

Here are the number of unique points and highest observed frequency for θ :

```
num_unique_theta = length(unique(thetars))
cat("Number of unique points in resampled theta = ",num_unique_theta,"\n")
```

```
## Number of unique points in resampled theta =  20
```

```
thetasort = table(thetars)

mode=names(sort(thetasort, decreasing=T)[1])
count=as.numeric(thetasort[mode])

cat("Most Observed value In resample of Theta = ",mode," ,which occured ",count," times.")
```

```
## Most Observed value In resample of Theta =  40  ,which occured  3811  times.
```

Lambda1

```
num_unique_lambda1 = length(unique(lambda1rs))
cat("Number of unique points in resampled Lambda 1 = ",num_unique_lambda1,"\n")
```

```
## Number of unique points in resampled Lambda 1 =  362
```

```
lambda1sort = table(lambda1rs)

mode=names(sort(lambda1sort, decreasing=T)[1])
count=as.numeric(lambda1sort[mode])

cat("Most Observed value In resample of Lambda1 = ",mode," ,which occured ",count," times.")
```

```
## Most Observed value In resample of Lambda1 =  3.1157378109971  ,which occured  488  times.
```

Lambda2

```
num_unique_lambda2 = length(unique(lambda2rs))
cat("Number of unique points in resampled Lambda 2 = ",num_unique_lambda2,"\n")
```

```
## Number of unique points in resampled Lambda 2 =  362
```

```
lambda2sort = table(lambda2rs)

mode=names(sort(lambda2sort, decreasing=T)[1])
count=as.numeric(lambda2sort[mode])

cat("Most Observed value In resample of Lambda2 = ",mode," ,which occured ",count," times.")
```

```
## Most Observed value In resample of Lambda2 =  0.938206297378235  ,which occured  488  times.
```

2

```
library(mvtnorm)
```

A

We are using Gibbs Sampling to generate random values from the random vector (X, Y) having a bivariate normal distribution $\sim N(\mu, \Sigma)$ where

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \Sigma = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

We will be generating random values from two seperate conditional Normal distributions of X and Y. Specifically, we have:

$$f(X|Y = y) \sim N(\mu_1 + \rho\frac{\sigma_1}{\sigma_2} * (y - \mu_2), (1 - \rho^2)\sigma_1^2)$$
$$f(Y|X = x) \sim N(\mu_2 + \rho\frac{\sigma_2}{\sigma_1} * (x - \mu_1), (1 - \rho^2)\sigma_2^2)$$

Then, our Gibbs Sampling Algorithm is as follows:
1.Start with an initial value x_0
2a. Generate y_i from $f(y|x_0)$
2b. Generate x_{i+1} from $f(x|y_i)$
3. Repeat step 2, increase our i value after each iteration until n iterations have been completed.

GENERATING VALUES

Before defining our Gibbs Sampling Algorithm, we must first create two functions that take our bivariate normal distribution and generate two seperate univariate normal values, X and Y. Here are the two functions that serve such utility:

```
gen_y=function(x,mean,covariance){
  mu1=mean[1]
  mu2=mean[2]
  sig1=sqrt(covariance[1,1])
  sig2=sqrt(covariance[2,2])
  rho = covariance[1,2]/(sig1*sig2)

  new_mean = mu2 +rho*(sig2/sig1)*(x-mu1)
  new_sigma =sqrt( (1-rho^2)*(sig2^2))

  return(rnorm(1,new_mean,new_sigma))}


gen_x= function(y,mean,covariance){
  mu1=mean[1]
  mu2=mean[2]
  sig1=sqrt(covariance[1,1])
  sig2=sqrt(covariance[2,2])
  rho = covariance[1,2]/(sig1*sig2)
```

```
  new_mean = mu1 +rho*(sig1/sig2)*(y-mu2)
  new_sigma =sqrt( (1-rho^2)*(sig1^2))

  return(rnorm(1,new_mean,new_sigma))}
```

Now that we have created our neccessary functions, we will create our Gibbs Sampling Algorithm:

```
Gibbs = function(n, mu, Sigma, x0){

#create vectors
x = c(x0)
y = c()

#iterate through n
for(i in 1:n){
#generate univariate values
y_i = gen_y(x0, mu, Sigma)
x0 = gen_x(y_i, mu, Sigma)
#fill in vector
x = c(x, x0)
y = c(y, y_i)
}
return(list(X=x, Y=y)) }
```

We will implement our Gibbs Samplig with the desired inputs:

```
mu = c(1,2)
Sigma = matrix(c(1,.5,.5,.4),2,2)

n= 5000
x0 = 7

list = Gibbs(n, mu, Sigma, x0)
X = list$X
Y = list$Y
```

```
cat("Generated X values has mean ",mean(X)," and variance ",var(X),"")
```

```
## Generated X values has mean  0.9432704  and variance  0.937128
```

```
cat("Generated Y values has mean ",mean(Y)," and variance ",var(Y),"")
```

```
## Generated Y values has mean  1.962308  and variance  0.3819214
```

We can see that Gibbs sampling produced X and Y such that they have the mean and variance provided in the Bivariate Normal Distribution, thus our Gibbs Sampling was effective.

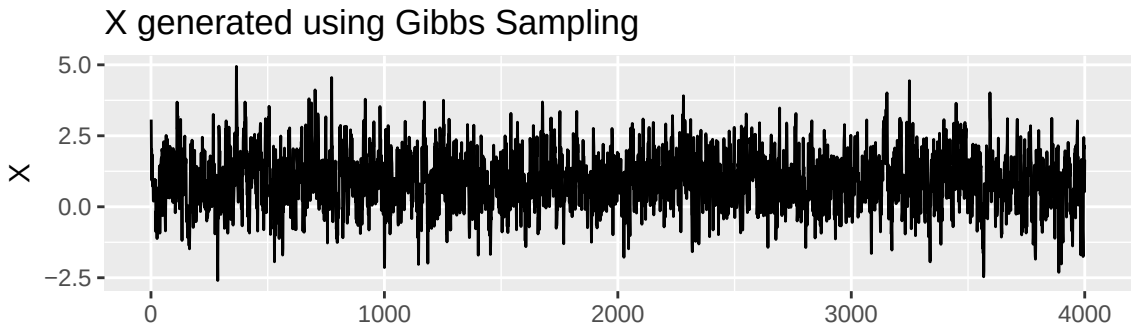
B

We will burn the first 1000 values from each of our vvariables and produce a plot of each individually first.

```
#burn 1000
X=X[1001:n]
Y=Y[1001:n]
```

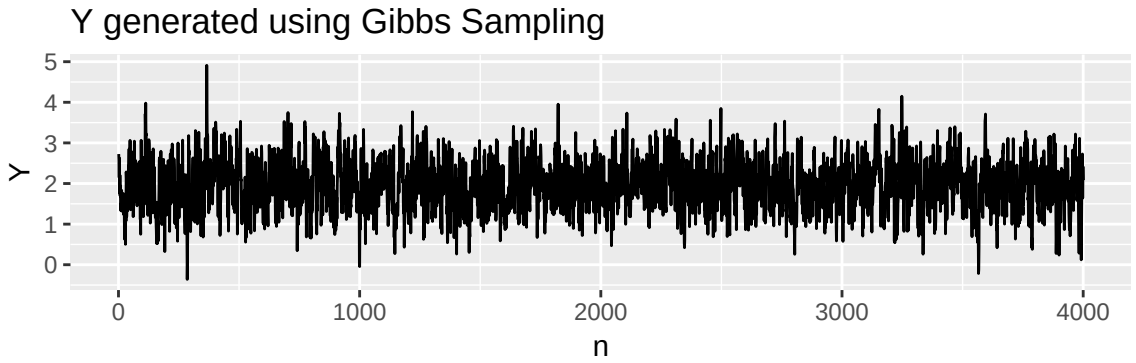
```
df = data.frame(x = seq(1,4000),X)

ggplot(data=df, aes(x=x, y=X))+
geom_line()+
labs(x="",
title = "X generated using Gibbs Sampling" , y='X')
```




```
df = data.frame(x= seq(1,4000),Y)

ggplot(data=df, aes(x=x, y=Y))+
geom_line()+
labs(x="n",
title = "Y generated using Gibbs Sampling" , y='Y')
```



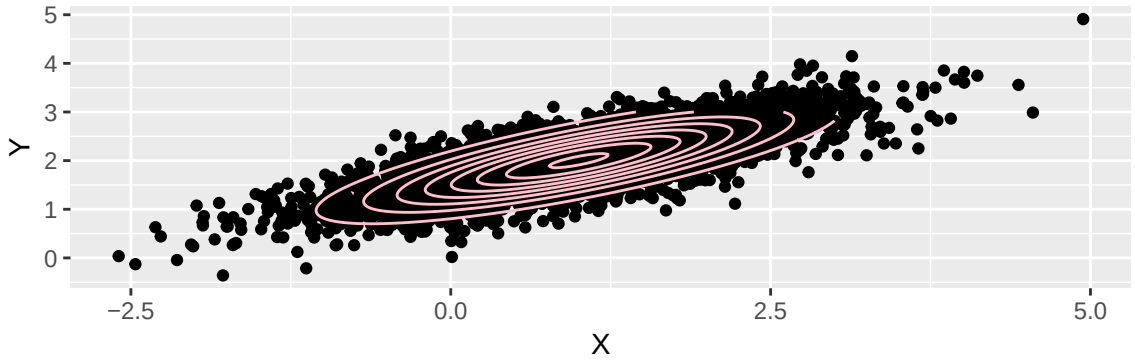
We can see that our values are stablized as the value of n increases, which is what we expected for the Gibbs sampling.

Now, we will plot our X and Y values in a bivariate manner.

```
df = data.frame(X,Y)

m = c(1,2)
sigma = matrix(c(1,.5,.5,.4),2,2)
data.grid = expand.grid(s.1 = seq(-3, 3, length.out=200),
            s.2 = seq(-3, 3, length.out=200))
q.samp = cbind(data.grid, prob =dmvnorm(data.grid, mean = m, sigma = sigma))

ggplot(df, aes(x=X, y=Y)) + geom_point()+
geom_contour(data=q.samp,aes(x=s.1, y=s.2, z=prob),color='pink',show.legend = TRUE)
```



From our bivariate plot, we can see that our data is centered at (1,2), which is the mean of our bivariate normal density. In addition, a contour plot of the bivariate normal distribution is provided. The values created fits the distribution contour quite well.