534 Final

Gustavo Esparza

5/18/2019

Libraries

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

1

A

We will be estimating $E(X^4)$, where $f(x) \sim e^{\frac{-x^3}{3}}, \infty < x < \infty$. Then, let $h(x) = x^4$. We will estimate the desired Kurtosis with the following Importance Sampling Algorithm:

1.) Draw X values from $g(x) \sim N(0,1)$
2.) Using the generated X values, compute the standardized weights:

$$w(X_i) = \frac{f(x_i)/g(x_i)}{\sum_{i=1}^{n} f(x_i)/g(x_i)}$$

3.) Use the standardized weights to compute the estimated value of $E(X^4)$, defined by

$$estimate = \sum_{i=1}^{n} h(X_i) * w(X_i)$$

```
f = function(x) {
  return(exp(-abs(x)^3/3)) }

h = function(x) {
  return(x^4)}

g = function(x) {
  return(dnorm(x,0,1)) }

w_star = function(x) {
  return(f(x)/g(x))}

#1
n = 50000
X = rnorm(n,0,1)

#2
w_star = w_star(X)
w = w_star / sum(w_star)
```

```
#3
kurtosis = sum(h(X)*w)
```

The variance for our Importance Sampling estimate canbe defined as follows:

if $\bar{hw}$ is the Importance Sampling estimate found above, then

$$\sigma^2_{IS} = \frac{1}{n*(n-1)}\sum_{i=1}^{n}(h(x)w(x) - \bar{hw})^2)$$

Thus, we have the following results:

```
se_kurtosis = (1/(n*(n-1)))*sum(((h(X)*f(X)/g(X)) - kurtosis)^2)
cat("Kurtosis estimate using Importance Sampling = ",kurtosis,"")
```

```
## Kurtosis estimate using Importance Sampling =  1.468246
```

```
cat("\nStandard error of Kurtosis Estimate = ",se_kurtosis,"")
```

```
##
## Standard error of Kurtosis Estimate =  0.0007954583
```

B

To begin our Accept/Reject Algorithm, we must first find an appropriate envelope distribution. For this case, we will consider using the Standard Normal Distribution once more. Therefore, we must find some constant "c" (including the constant term for the standard normal) such that

$$e^{\frac{-x^3}{3}} \leq c * e^{\frac{-x^2}{2}}$$

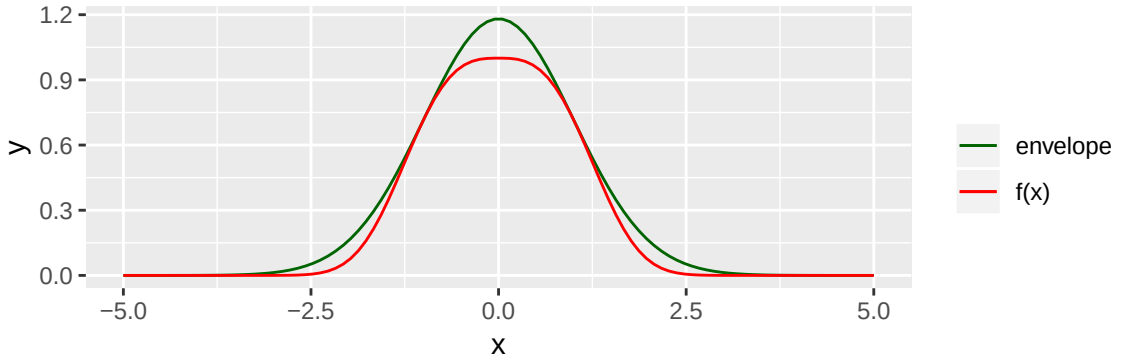
Thus, we have the following result:

$c \geq exp(\frac{-x^3}{3} + \frac{x^2}{2})$
Now, taking the derivative of $log(c)$, we have
$\longrightarrow log(c) = \frac{-x^3}{3} + \frac{x^2}{2}$
$\longrightarrow \frac{\partial log(c)}{\partial x} = -x^2 + x = 0$
Thus, we end up with $x = 1$ and therefore $c = exp(1/6)$
We will show that this value does create an envelope function:

```
g = function(x){
  return(exp(-abs(x**2)/2))}

c= exp(1/6)
cg= function(x) {
  return(c*g(x))}
```

```
x = seq(-5,5, length=100)
df = data.frame(x,f(x), cg(x))
ggplot(df,aes(x=x))+
geom_line(aes(y=cg(x), colour="envelope"))+
geom_line(aes(y=f(x), colour="f(x)"))+ labs(y="y")+
scale_colour_manual("", values=c('dark green','red'))
```



We can clearly see that our envelope is effective, and we can now utilize the Accept/Reject Algorithm to find our estimate for Kurtosis. Here is the algorithm we will be using:

1.) Sample X from N(0,1)
2.) Sample U from UNIF(0,1)
3.) Reject X if $U > \frac{f(x)}{cg(x)}$ and accept otherwise.

Then the accepted values, X will now follow the distribution of our original distribution,F. Here is the algorithm in effect:

```
#Accept/reject algorithm
n= 50000

X = rnorm(n,0,1)
U = runif(n,0,1)
X= X[U<=f(X)/cg(X)]
```

Now that we have generated our values that follow the original distribution, we can estimate the kurtosis and standard error quite simply:

```
se_kurtosis = (1/(n*(n-1)))*sum(((X^4 - mean(X^4))^2))
cat("Kurtosis estimate using Importance Sampling = ",mean(X^4),"")
```

```
## Kurtosis estimate using Importance Sampling =  1.478977
```

```
cat("\nStandard error of Kurtosis Estimate = ",se_kurtosis,"")
```

```
##
## Standard error of Kurtosis Estimate =  0.0002071621
```

C

We can see that both of our estiamates are similar to one another. We can also see that the standard errors are quite similar, with the standard error for the Accept/Reject algorithm being consistently smaller than the Importance Sampling standard error. I prefer the Accept/Rejection method because it allows us to generate values from the distribution and compute various expected values whereas Importance Sampling seems to be less flexible in that regard.

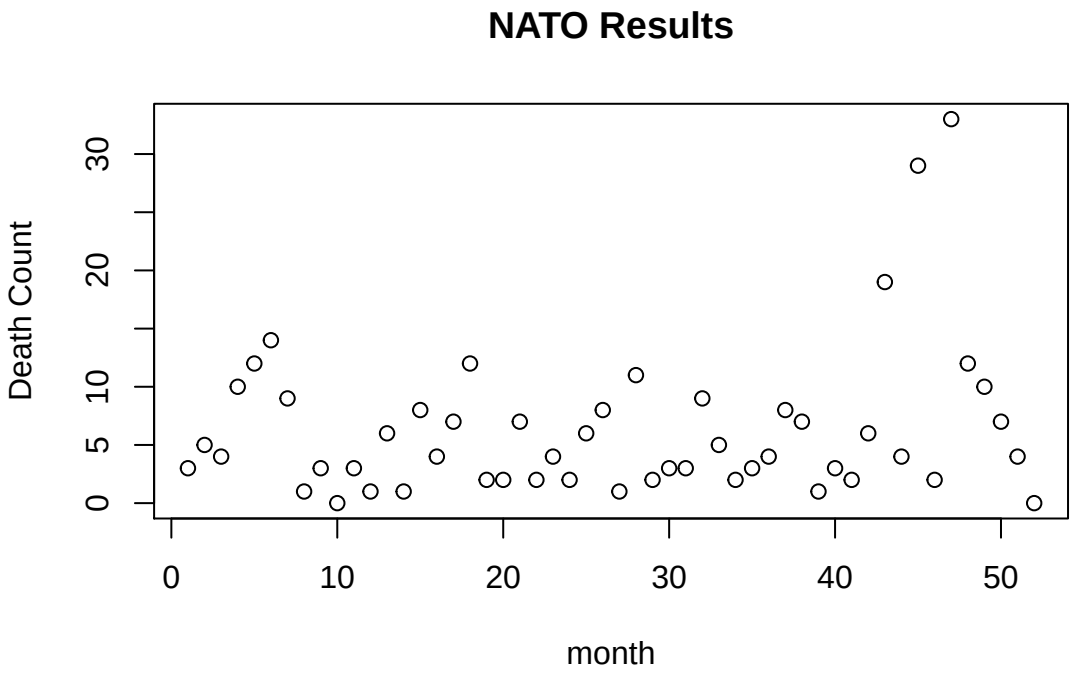
2

A

Let us perform an exploratory analysis on the data to investigate and decide on a changepoint, θ.

```
time= seq(1,52)
data= c(3,5,4,10,12,14,9,1,3,0,3,1,6,1,8,4,7,12,2,2,7,2,4,2,6,8,1,11,
        2,3,3,9,5,2,3,4,8,7,1,3,2,6,19,4,29,2,33,12,10,7,4,0)

plot(time,data,xlab="month",ylab="Death Count",main="NATO Results")
```



From the plot of our data, we can see that the death counts is pretty consistent up until the last few months recorded. At some point in the early 40 range, we can see a dramatic jump in death count followed by a sharp decline to nearly zero. It is a valid to claim that the change point could very well be anywehere between months 39 and 43. These months are right in the middle of the dramatic drop, and would seem to be a good destination for two poisson curves to intersect/change off. Thus, we will go forward with a changing point value of $\theta = 42$.

B

NOTE: I was unsure how to approach this section of the problem. I first attempted to built an optimization function that would compute all three MLE estimates at once. Due to limited time, I have displayed results that were computed to show two different MLE computations for λ_1 and λ_2 with the previously decided θ value. This is no excuse for any incorrect methodology/computations, just a clarification that the original goal was understood and attempted.

We will be using our θ value of 42 to compute the MLE values for λ_1 and λ_2. Thus by partitioning the data into two sets of data, we can run two independent Newton Algorithms that can give an estimate value for our λ values. We will now present the elements necessary for the Newton Algorithm.

Log-Likelihood

For the poisson distribution, we have the following density definition:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

Then, provided a sample of n, we have the following Likelihood function:

$$L(x) = \prod_{i-1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x!}$$

And, thus our log-likelihood function is defined as

$$l(x) = \sum_{i=1}^{n} x_i log(\lambda) - log(x!) - \lambda$$

As, a result we have the following gradient value

$$\frac{\partial l(x)}{\partial \lambda} = \sum_{i=1}^{n} \frac{x_i}{\lambda} - 1$$

and a hessian value of

$$\frac{\partial^2 l(x)}{\partial \lambda^2} = \sum_{i=1}^{n} \frac{-x_i}{\lambda^2}$$

Then, we can utilize a newton algorithm with both sets of data (partitioned at $\theta = 42$) to find our MLE value.

Newton Elements and Algorithm

```
#log-likelihood#
likelihood = function(x,lambda){

  l = sum( x*log(lambda) - log( factorial(x) ) - lambda   )
  return(l)}

#gradient#

gradient=function(x,lambda){

  dlambda = sum( (x/lambda) -1 )
  return(dlambda)
}

#Hessian#

hessian=function(x,lambda){

  ddlambda = sum (-x/(lambda^2))
```

```
  return(ddlambda)
}
```

Using our newton elements, we can build a newton algorithm that employs a stopping criteria based on the relative error of consecutive estimate values.

```
Newton = function (maxit,x,lambda, tolerr,tolgrad)
  {

  cat(" b" , "    lambda(b)\n")

     cat(sprintf('%2.0f     %12.12f\n',
                 0,lambda))

  for (b in 1:maxit){

    dl=gradient(x,lambda)
    ddl=hessian(x,lambda)

    lambdab = lambda - dl/ddl

    relerr = abs(lambdab-lambda)/max(1,abs(lambdab))
    dlb=gradient(x,lambdab)

    if(relerr<tolerr & abs(dlb)<tolgrad){
      break
    }
    cat(sprintf('%2.0f     %12.12f\n',
                b,lambdab))
    lambda=lambdab
  }}
```

Lambda 1

```
data1 =data[1:42]

maxit=20
lambda=.5
x=data1
tolerr=1e-6
tolgrad=1e-9

Newton(maxit,x,lambda,tolerr,tolgrad)
```

```
##  b     lambda(b)
##  0     0.500000000000
##  1     0.949029126214
##  2     1.714429301646
##  3     2.829590404873
##  4     4.026770916092
##  5     4.747594606705
##  6     4.899725664459
##  7     4.904756733519
```

```
##  8     4.904761904756
```

We can see that our newton algorithm converged relatively quickly to a value of $\lambda_1 = 4.90$.

Lambda2

```
data2 = data[42:52]

maxit=20
lambda=1
x=data2
tolerr=1e-6
tolgrad=1e-9

Newton(maxit,x,lambda,tolerr,tolgrad)
```

```
##  b     lambda(b)
##  0     1.000000000000
##  1     1.912698412698
##  2     3.506011369862
##  3     5.938901525585
##  4     8.798627935007
##  5     10.838728973779
##  6     11.421438079007
##  7     11.454449763423
##  8     11.454545453746
```

We can see that our newton algorithm converged relatively quickly to a value of $\lambda_2 = 11.45$.

C

Now, we will be utilizing the SIR algorithm to estimate our parameters. We have the priors as $\lambda_1 \sim \Gamma(2, 1/2)$, $\lambda_2 \sim \Gamma(4, 1/2)$

Then, we will first generate our priors with a sample size of 20,000:

```
m = 20000

theta = sample(1:51, m, replace=T) #theta prior

lambda1 = rgamma(m,shape=2, rate=.5) #lambda1 prior

lambda2 = rgamma(m,shape=4,rate=.5) #lambda2 prior
```

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:

$$X_1, ..., X_\theta \sim Poisson(\lambda_1)$$
$$x_{\theta+1}, ..., X_{52} \sim Poisson(\lambda_2)$$

Then, our Likelihood function is defined as follows:

$$L(\theta, \lambda_1, \lambda_2) = \prod_{i=1}^{\theta} \frac{e^{-\lambda_1}\lambda_1^{x_i}}{x_i!} \prod_{i=\theta+1}^{52} \frac{e^{-\lambda_2}\lambda_2^{x_i}}{x_i!} = \frac{\prod_{i=1}^{\theta} e^{-\lambda_1}\lambda_1^{x_i} \prod_{i=\theta+1}^{52} e^{-\lambda_2}\lambda_2^{x_i}}{\prod_{i=1}^{52} x_i!}$$

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

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:

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

Thus, we have the following Log-Likelihood function:

$$l(\theta, \lambda_1, \lambda_2) = -\theta\lambda_1 - (52-\theta)\lambda_2 + \sum_{i=1}^{\theta} \log(\lambda_1)x_i + \sum_{i=\theta+1}^{52} \log(\lambda_2)x_i$$

```
#log-likelihood
loglike = function(data, theta, lambda1, lambda2){
sumtheta = c()
sum52 = c()

for(i in 1:m){

xtheta = data[1:theta[i]]
x52   = data[(theta[i]+1):52]

sumtheta = c(sumtheta, sum(xtheta))
sum52 = c(sum52, sum(x52))
}

return(-theta*lambda1 - (52-theta)*lambda2 +
      log(lambda1)*sumtheta + log(lambda2)*sum52)
}
```

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

Likelihood = exp(llike) #Convert back to Likelihood

weights = Likelihood/sum(Likelihood)
```

```
n = 20000
resample= sample(1:m, n, replace=TRUE, prob=weights)
#getting resampled values for each posterior

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

THETA

Here is a summary of the mean and Confidence Interval for the posterior θ.

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

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

```
## Mean for Theta =  41.78315
```

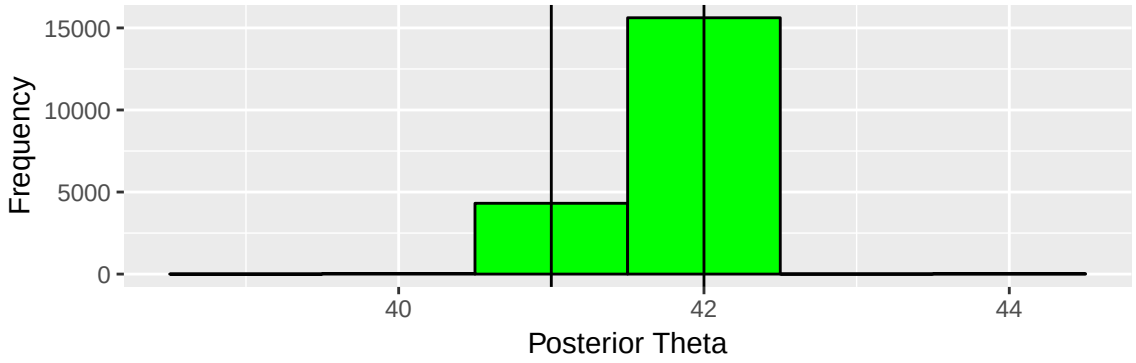
```
cat("\nCredible Confidence Interval for Theta = (",L,",",U,")")
```

```
##
## Credible Confidence Interval for Theta = ( 41 , 42 )
```

Here is a histogram of the posterior 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')
```



We can see that the Posterior theta values are heavily concentrated between 41 and 42, which is strikingly similar to the theta value that was derived by the exploratory analysis.

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),"")
```

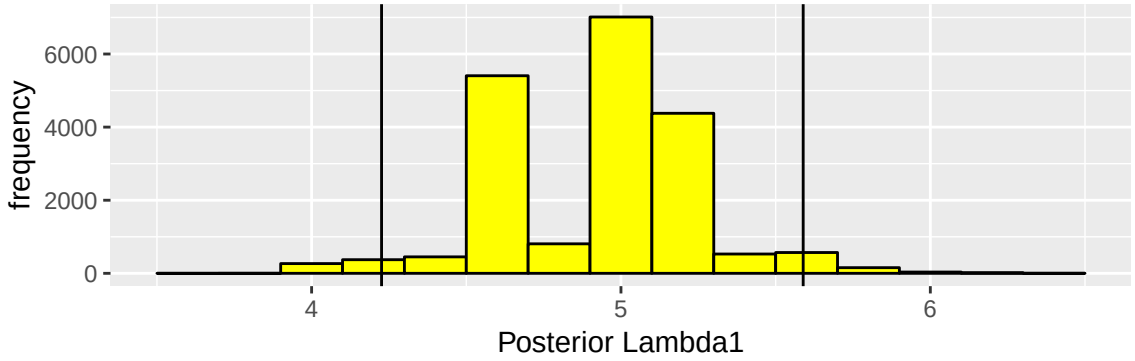
```
## Mean for Lambda 1 =  4.942007
```

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

```
##
## Credible Confidence Interval for Lambda 1 = ( 4.226104 , 5.589008 )
```

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 = .2)+
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),"")
```

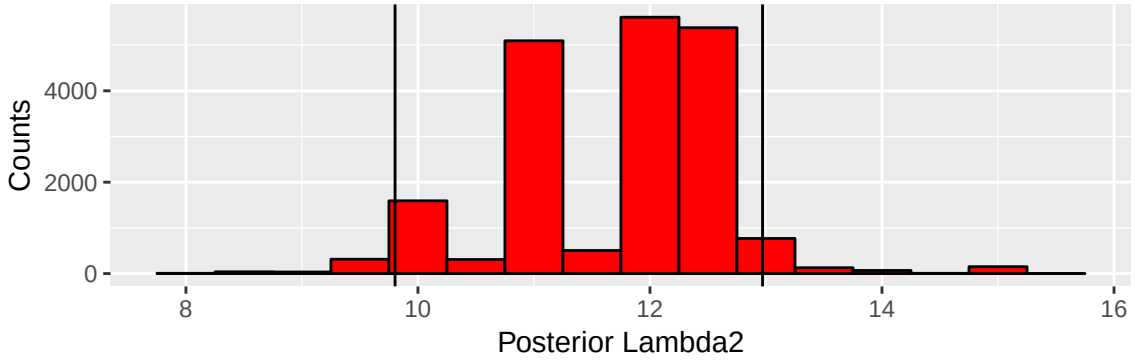
```
## Mean for Lambda 2 =  11.75261
```

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

```
##
## Credible Confidence Interval for Lambda 2 = ( 9.802754 , 12.97036 )
```

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 = .5)+
geom_vline(xintercept=L)+
geom_vline(xintercept=U)+
labs(x="Posterior Lambda2", y='Counts')
```



DISCUSSION

We can see that the values found in both methods are very similar, so we have confidence that our MLE estimates are consistent and accurate. Since an appropriate theta was chosen for the newton optimization, we can see that the resulting lambda values are consistent with the SIR alogorithm estimates.

3

A

Complete Likelihood

We are considering 20 independent Bernoulli trials with probability θ. Of the 20 trials, 19 were observed and from those 19 it was found that 5 were infected.

Thus, we have the following complete log-likelihood:

if

$$L(x,\theta) = \prod_{i=1}^{20} \theta^{x_i}(1-\theta)^{x_i}$$

$$l_c(x,\theta) = \sum_{i=1}^{19}[x_i log(\theta) + (1-x_i)log(1-\theta)] + x_{20}log(\theta) + (1-x_{20})log(1-\theta)$$

E-step

Since the expected value of the bernoulli distribution is simply θ, we have the following expectation function:

Setting $X_{20} = \theta^*$

$$Q(x,\theta) = \sum_{i=1}^{19}[x_i log(\theta) + (1-x_i)log(1-\theta)] + \theta^* log(\theta) + (1-\theta^*)log(1-\theta)$$

M-Step

Taking the derivative of our Expectation function, we have

$$Q'(x,\theta) = \sum_{i=1}^{19}\left[\frac{x_i - \theta}{\theta(1-\theta)}\right] + \frac{\theta^* - \theta}{\theta(1-\theta)} = 0$$

$$\longrightarrow \sum_{i=1}^{19} x_i + \theta^* = 20\theta \rightarrow \hat{\theta} = \frac{\sum_{i=1}^{19} x_i + \theta}{20}$$

Thus, we have $\theta_{em} = \frac{5+\theta^*}{20}$

Algorithm

Here is the EM algorithm in effect:
stopping criteria:

We chose to stop the algorithm once the relative error (difference between iterative values) was less than 10^{-7}.

```
EM = function (theta, maxit){

   cat("Iteration", "           theta",      "         MRE\n")

  for (it in 1:maxit){

    ### The E Step
    xstar = theta

    ### The M Step
    theta1 = (5+xstar)/20

    relerr = abs(theta1-theta)/max(1,abs(theta1))


    cat(sprintf(' %2.0f                    %6.6f        %2.2e\n'
              ,it, theta,relerr))

      theta = theta1

      if(relerr<1e-7){break}

  }
  print('_____________________________________________')
  print(sprintf('Theta'))
  print(sprintf('%6.6f',
                theta))
}
```

We will implement our algorithm with a θ value of .9 (not very likely to be the true value since we have 5/19 as an estimate) and 25 max iterations

```
EM(.9,25)
```

```
## Iteration            theta          MRE
##   1                   0.900000       6.05e-01
##   2                   0.295000       3.03e-02
##   3                   0.264750       1.51e-03
##   4                   0.263238       7.56e-05
##   5                   0.263162       3.78e-06
##   6                   0.263158       1.89e-07
##   7                   0.263158       9.45e-09
## [1] "_____________________________________________"
## [1] "Theta"
## [1] "0.263158"
```

We can see that our EM algorithm converges very quickly to a θ value of about .26. This is extremely close to the proportion of "successes" found at the 19 observation level. This should not be surprising, as we only had one missing value to determine and the probability density is fairly straight forward.

B

Now, we will consider the previous problem by applying the Gibbs Sampling algorithm. We have been provided with the prior on $\theta \sim Beta(\alpha, \beta)$.

Considering X_{20} as a random variable, we will obtain samples from the joint distribution $f(X_{20}, \theta)$ via the conditional distributions of $f(\theta)$ and $f(X_{20})$.

Joint Posterior

The Joint Posterior is defined as the product of our Likelihood function and our prior. Thus, we have the following

$$f(X_{20}, \theta) = \prod_{i=1}^{20} \theta^{x_i}(1-\theta)^{1-x_i} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1}$$

$$\propto \theta^5(1-\theta)^{n-5}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

Posterior Conditionals

By reviewing the previously found joint, we can see the following relation:

$$f(X_{20}, \theta) \propto \theta^5(1-\theta)^{n-5}\theta^{\alpha-1}(1-\theta)^{\beta-1} \sim Bern(\theta) \times Beta(\alpha + 5 + X_{20}, \beta + n - (5 + X_{20}))$$

Then, we have the following conditionals:

$$f(X_{20}) \sim bern(\theta)$$

$$f(\theta) \sim Beta(\alpha + 5 + X_{20}, \beta + n - (5 + X_{20}))$$

Now, we can employ our Gibbs Sampling Algorithm.
Here are the steps to our algorithm:

1.) Start with an initial x and θ
2.) generate in turn: $f(X_{20_i})$ then $f(\theta_i)$
3.) Iterate through i until complete iterations have been achieved.

Here is the implementation of our algorithm with a starting value of X_{20} =0 and $\theta = .5$

```
a=1
b=1
n=20
i=20000
x20= rep(0,i)
theta=rep(0,i)

x20[1]=0
theta[1]=.5
# GIBBS
for (i in 2:i){
  x20[i] =  rbinom(1,1,theta[i-1])
  theta[i] = rbeta(1,a+5+x20[i],b+n-(5+x20[i]))}
```

```
x20=x20[1001:i]
theta=theta[1001:i]
```

X(20)

Here is a summary for the postrior X_{20} values:

```
#lower and upper bounds of the CI of X(20)
L= sort(x20)[0.025*(19000)]
U= sort(x20)[0.975*(19000)]

cat("Generated X(20) values has mean ",mean(x20)," and standard deviation ",sd(x20),"")
```

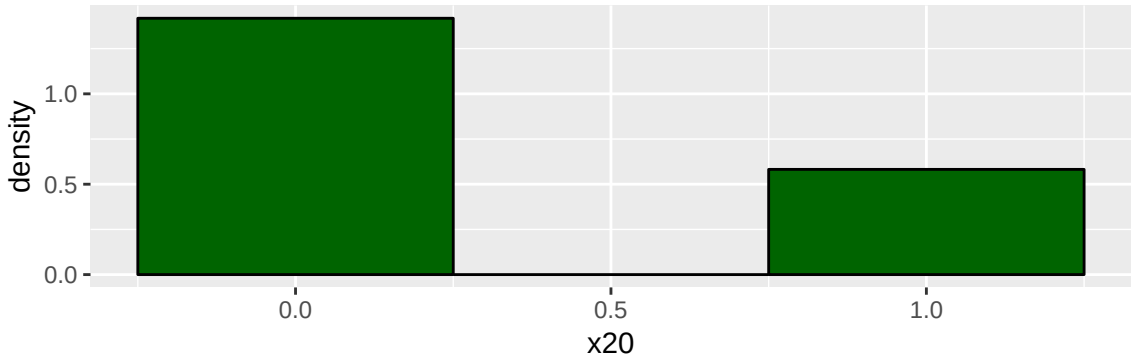
```
## Generated X(20) values has mean  0.291  and standard deviation  0.4542355
```

```
cat("\nCredible Confidence Interval for X20 = (",L,",",U,")")
```

```
##
## Credible Confidence Interval for X20 = ( 0 , 1 )
```

This doesn't make too much sense for this bernoulli model, but we will see from the histograms that we are more likely to see a resulting value of 0. However, this result does make sense in the context of the problem, as we expect to see a result of 0 far more times than 1. Here's a histogram for the posterior values of X_{20}:

```
df = data.frame(x = seq(1,19000),x20)
ggplot(df, aes(x=x20))+
geom_histogram(aes(y=..density..), fill="dark green", color="black", binwidth=.5)
```



Again, our histogram for X(20) does not make a great deal of sense in the context of the bernoulli problem, but it still points to the estimated $\theta = 5/19$. We can also see that x20 is much more likely to result in 0 than it is for 1, which also reflects the probability found in the 19 observed trials.

THETA

Here is a summary for the postrior θ values:

```
#lower and upper bounds of the CI of theta
L= sort(theta)[0.025*(19000)]
U= sort(theta)[0.975*(19000)]

cat("Posterior theta values has mean ",mean(theta)," and standard deviation ",sd(theta),"")
```

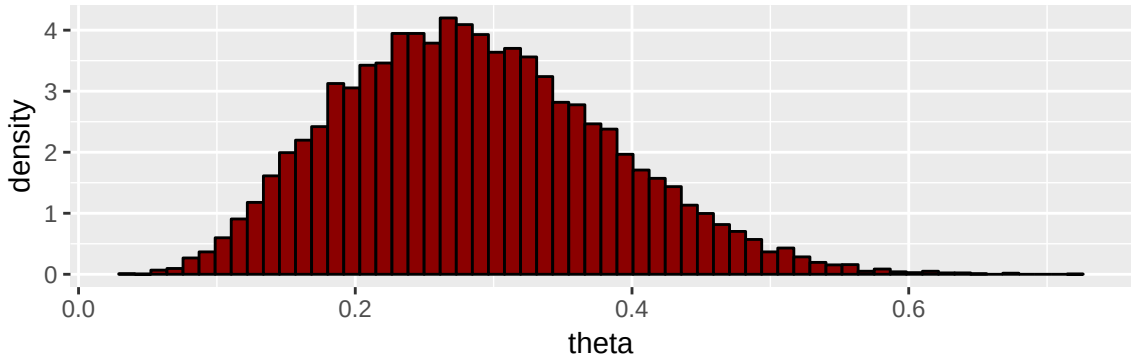
```
## Posterior theta values has mean  0.2862971  and standard deviation  0.09628206
```

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

```
##
## Credible Confidence Interval for theta = ( 0.120323 , 0.4885234 )
```

```r
df = data.frame(x= seq(1,19000),theta)

ggplot(df, aes(x=theta))+
geom_histogram(aes(y=..density..), fill="dark red", color="black",bins=60)
```



We can see that the values of θ does vary, but it still centered around the point estimate of 5/19.

Discussion

We can see that the θ values produced in both algorithms are consistently similar. The EM algorithm was useful for finding a point estimate value for the MLE of θ, but the Gibbs Sampling method allows us to see an interval of possible values as well as a projection for the missing value.

4

A

We are simulating 200 realizations from

$$\delta N(7, .5^2) + (1 - \delta)N(10, .5^2) \text{ , where } \delta = .7$$

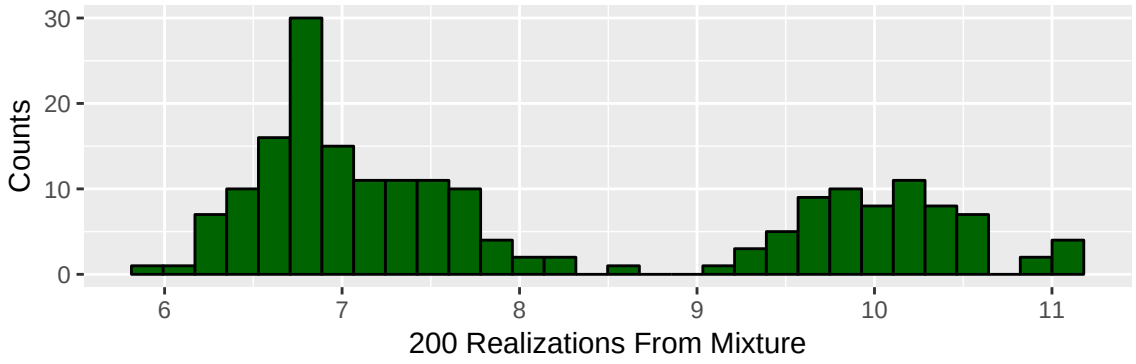
We can see that the mixture consists of two different normal distributions. Thus, we will sample 200 bernoulli trials to determine where we will draw our Normal values from.
Here is the resulting code and histogram:

```
set.seed(534)

delta =rbinom(200,1,.7)

mix=rep(0,200)
for(i in 1:200){
  if ( delta[i] ==1)
    mix[i] = rnorm(1,7,.5)
    else
    mix[i] = rnorm(1,10,.5)
}
```

```
df = data.frame(x = mix)
ggplot(df, aes(x = x)) +
geom_histogram(fill="Dark Green", color="black",bins=30)+
labs(x="200 Realizations From Mixture", y='Counts')
```



We can see that the histogram is bimodal, where the peaks are the two respective means, and resembles the graph provided in the textbook.

B

Using the data generated in A, we will implement an Independence Chain MCMC procedure to simulate from the posterior distribution of δ.

For this procedure, our prior is $g \sim UNIF[0, 1]$ and as a result, our Metropolis Ratio is

$$R(x^{(t)}, x^*) = \frac{f(x^*)g(x^{(t)})}{f(x^{(t)})g(x^*)} = \frac{\delta_i N(7, .5^2) + (1 - \delta_i)N(10, .5^2)}{\delta_{t-1} N(7, .5^2) + (1 - \delta_{t-1})N(10, .5^2)}$$

which is simply the likelihood ratio of our sampling distribution (also found in the text). Thus, we have created a function that will calculate the desired likelihood values:

```
ratio=function(p,x) {
prod(p*dnorm(x,7,.5)+(1-p)*dnorm(x,10,.5) )}
```

Now, Here is our implementation of the Metropolis Hastings algorithm to simulate an independence chain. Our proposal distribution is $unif[0,1]$ so the text recommends using a $beta(1,1)$ distribution for computations:

```
set.seed(534)

it=10000
#Prior
p=rep(0,it)
p[1]=.2

#MCMC CHAIN
for (i in 2:it) {

p[i]=rbeta(1,1,1)

R= ratio(p[i],mix)/ratio(p[i-1],mix)

   if (R<1 && rbinom(1,1,R)==0){
        p[i]=p[i-1]}}
```

Here are two summary statistics for the resulting Independence Chain:

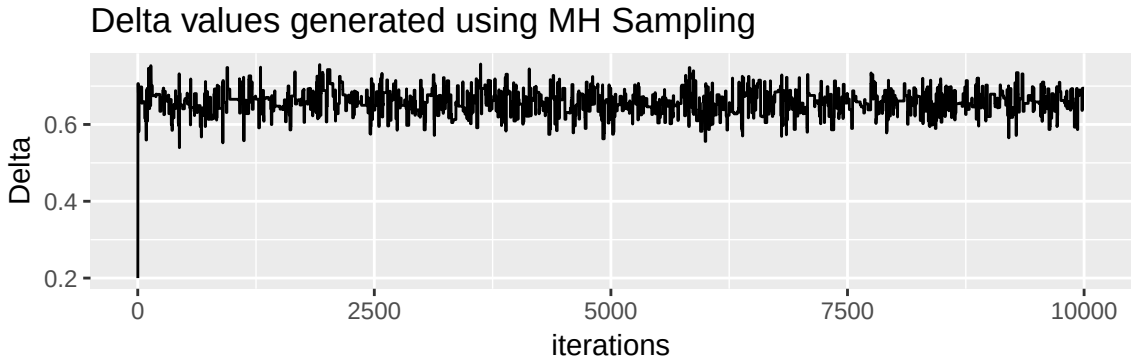
```
PostB=p

cat("Mean of = ",mean(PostB),
    "\nStandard Error of =",sd(PostB),"")
```

```
## Mean of =  0.6576316
## Standard Error of = 0.03287957
```

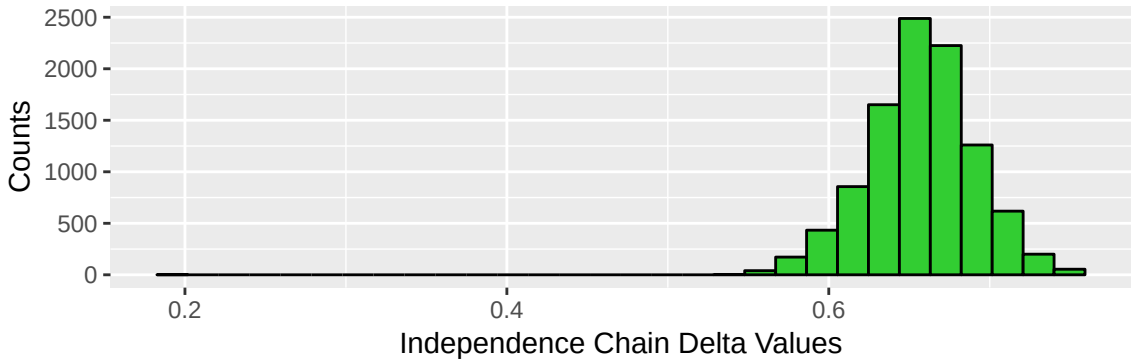
Finally, here is a plot and histogram that show the iterations and distribution of the resulting δ values:

```
df = data.frame(x= seq(1,10000),PostB)
ggplot(data=df, aes(x=x, y=PostB))+
geom_line()+
labs(x="iterations",
title = "Delta values generated using MH Sampling" , y='Delta')
```




```
df = data.frame(x = PostB)
ggplot(df, aes(x = x)) +
```

```
geom_histogram(fill="lime green", color="black",bins=30)+
labs(x="Independence Chain Delta Values", y='Counts')
```



We can see that our Independence Chain sampled δ values do in fact converge to a value that is very similar to 0.7. In addition, our histogram is also centered at 0.7, with minimal deviation.

C

We will now implement a random walk chain with $\delta^* = \delta^{(t)} + \epsilon$ where, $\epsilon \sim unif(-1, 1)$

From the textbook section notes, we can note the following results:

1.) The target density, δ transformed for our ϵ is

$$\delta = exp(\epsilon)/(1 + exp(\epsilon))$$

2.) As a result, our MH ratio becomes

$$\frac{L(x^{(t+1)})}{L(x^{(t)})} \times \frac{exp(\epsilon_t)(1 + exp(\epsilon_{t+1}))}{exp(\epsilon_{t+1})(1 + exp(\epsilon_t))}$$

where $L(x)$ is the likelihood function found in part B.

Thus, here is the implementation of our random walk chain:

```
set.seed(534)

it = 10000
e=rep(0,it)
e[1]= runif(1,-1,1)
p=rep(0,it)

p[1]=exp(e[1])/(1+exp(e[1]))

for (i in 2:it) {

   e[i]=e[i-1]+runif(1,-1,1)     #proposal

   p[i]=exp(e[i])/(1+exp(e[i]))
   #MH RATIO
```

```
   R=(ratio(p[i],mix)/ratio(p[i-1],mix))*
     (exp(e[i-1])*(1+exp(e[i])))/(exp(e[i])*(1+exp(e[i-1])))
   #Reject/Accept
   if (R<1 && rbinom(1,1,R)==0){
        p[i]=p[i-1]
        e[i]=e[i-1] }}
```

Here are two summary statistics for the resulting Random Walk:

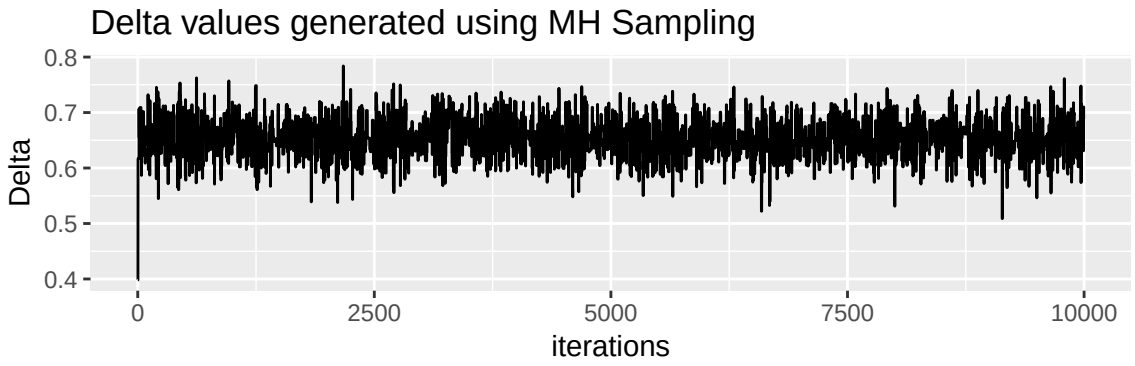
```
PostC=p

cat("Mean of = ",mean(PostC),
    "\nStandard Error of =",sd(PostC),"")
```

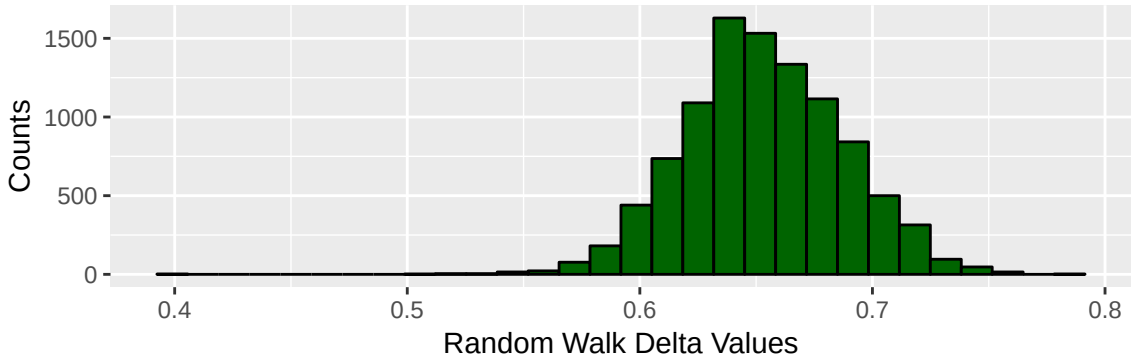
```
## Mean of =  0.653093
## Standard Error of = 0.03393043
```

Finally, here is a plot and histogram that show the iterations and distribution of the resulting δ values:

```
df = data.frame(x= seq(1,10000),PostC)
ggplot(data=df, aes(x=x, y=PostC))+
geom_line()+
labs(x="iterations",
title = "Delta values generated using MH Sampling" , y='Delta')
```




```
df = data.frame(x = PostC)
ggplot(df, aes(x = x)) +
geom_histogram(fill="Dark green", color="black",bins=30)+
labs(x="Random Walk Delta Values", y='Counts')
```



We can see that our Random Walk sampled δ values do in fact converge to a value that is very similar to 0.7. In addition, our histogram is also centered at 0.7, with minimal deviation.