Normal Mixture Model, Gibbs/Metropolis-Hastings Sampler

Jordan Aron

October 15, 2017

$z_i \sim Bern(\frac{a}{a+b})$ where $a = (p * f(x_i|\mu_1, \sigma_1))$ and $b = ((1-p) * f(x_i|\mu_2, \sigma_2))$. 1000 $z_i's$ are drawn each iteration.

$p \sim beta(\sum z, n - \sum z)$

$\sigma_1 \sim inverse - gamma(\frac{\kappa_{n1}}{2}, \frac{\kappa_{n1}}{2}\sigma^2_{n_1})$ where $\kappa_{n_1} = \kappa_{0_1} + n_1$, κ_{0_1} is the prior sample size for population 1, and n_1 is the sample size of the data for population 1

$\sigma_2 \sim inverse - gamma(\frac{\kappa_{n2}}{2}, \frac{\kappa_{n2}}{2}\sigma^2_{n_2})$ where $\kappa_{n_2} = \kappa_{0_2} + n_2$, κ_{0_2} is the prior sample size for population 2, and n_2 is the sample size of the data for population 2

$\theta_1 \sim normal(\mu_{n_1}, \frac{\sigma_1^2}{\kappa_{n_1}})$ where $\mu_{n_1} = \frac{\kappa_{0_1}}{\kappa_{n_1}}\mu_{0_1} + \frac{n_1}{\kappa_{n_1}}\bar{y_1}$, μ_{0_1} is the prior mean for population 1, and $\bar{y_1}$ is the mean of the data for population 1

$\theta_2 \sim normal(\mu_{n_2}, \frac{\sigma_2^2}{\kappa_{n_2}})$ where $\mu_{n_2} = \frac{\kappa_{0_2}}{\kappa_{n_2}}\mu_{0_2} + \frac{n_2}{\kappa_{n_2}}\bar{y_2}$, μ_{0_2} is the prior mean for population 1, and $\bar{y_2}$ is the mean of the data for population 1

If the Metropolis-Hastings algorihtm is used to calculate p:

We are attempting to sample from $P(p|\sum z)$.

The prior for p is: $P(p) = \frac{sin(5*\pi*p)^2}{c}$ where c is some normalizing constant.

As the $z_i's$ are distributed the same as above, $P(p|\sum z) = \binom{n}{\sum z} * p^{\sum z} * (1-p)^{n-\sum z}$

The proposal distribution for p^* is the beta distribution

We then calculate log(r) as it is more stable than r. $log(r) = log(\frac{P(p^*|\sum z)}{P(p^{(s)}|\sum z)} * \frac{J_s(p^{(s)}|p^*)}{J_s(p^*|p^{(s)})}) = log(\frac{P(\sum z|p^*)P(p^*)}{P(\sum z|p^{(s)})P(p^{(s)})} * \frac{J_s(p^{(s)})}{J_s(p^*)}) = \sum(log(P(z|p^*)) - log(P(z|p^{(s)}))) + (log(P(p^*)) - log(P(p^{(s)}))) + (log(J_s(p^{(s)})) - log(J_s(p^*)))$. Note that both the c's and the $\binom{n}{\sum z}$ are cancelled out.

$J_s(p)$ randomly samples from a beta distribution with parameters $b = (\frac{p(1-p)}{v} - 1)(1-p)$, $a = b\frac{p}{1-p}$, and v is a constant (set at .0005)

We accept p^* as a new value with probability r. If rejected p is instead accepted.

```
#function of Gibbs/Metropolis-Hastings algorithm
gibbs.MH <- function(data, trials, use.MH){
  library(car)

  #Dats is the data used for the algorithm
  #trials is number of trials to be done
  #Use.MH determines whether or not to use a metropolis-hastings algorithm for p
  #A 0 is no, anything else is yes

  n <- length(data)
  p<-0.5
  p.star<-0.5
```

```
r.log<-1

#creates priors
prior.pop1 <- rnorm(pop1.prior.sampleSize, pop1.prior.trueMean, pop1.prior.trueSD)
prior.pop2 <- rnorm(pop2.prior.sampleSize, pop2.prior.trueMean, pop2.prior.trueSD)

#creates matrix where gibbs will be done
simlist <- matrix(rep(0,5*trials), ncol = 5)
colnames(simlist) <- c("p", "SD of Pop1", "SD of Pop2", "Mean of Pop1", "Mean of Pop2")

#saves value for later usage
pop1.prior.mean <- mean(prior.pop1)
pop1.prior.variance <- var(prior.pop1)
pop2.prior.mean <- mean(prior.pop2)
pop2.prior.variance <- var(prior.pop2)

#initialize first entry in gibbs so it will run correctly
#p sigma1 sigma2 theta1 theta2
simlist[1,1] <- .5
simlist[1,2] <- sqrt(pop1.prior.variance)
simlist[1,3] <- sqrt(pop2.prior.variance)
simlist[1,4] <- pop1.prior.mean
simlist[1,5] <- pop2.prior.mean


#The actual gibbs sampler
#starts at 2 since we initialized values for when i=1
for(i in 2:trials){

  #max size will be z
  z <- numeric(n)
  a <- numeric(n)
  b <- numeric(n)


  j = 1
  #calculates a_j and b_j for each data_j
  #then calculates z_i's
  for (j in 1:n){
    a[j] = simlist[i-1,1]*dnorm(data[j],simlist[i-1,4],simlist[i-1,2])
    b[j] = (1-simlist[i-1,1])*dnorm(data[j],simlist[i-1,5],simlist[i-1,3])
    z[j] = rbinom(1,1,(a[j]/(a[j]+b[j])))
  }

  #calculates p using beta distribution for gibbs
  if (use.MH == 0){
    simlist[i,1] <- rbeta(1,sum(z), n-sum(z))
  #calculates p ising metropolis hastings with beta as the proposal distribution
  } else {
    #metropolis hastings algorithm to calculate p
    #constant, larger values allow larger skips, but also can produce NaN
    variance.constant = .0005
    c = ((p*(1-p)/variance.constant) - 1)*(1 - p)
```

```
  d = c*(p/(1-p))
  p.star = rbeta(1,d,c)

  #The algorithm ran into problems with really large and small values of p.star
  #Played around with these values, using 0.01 and 0.99 produced a weird shaped gamma for the next
  #doesnt impact calculations done later
  #One caveat is this method can't be used on a population where p is extrmeley small
  # if (p.star < 0.03){
  #   p.star <- 0.03
  # }
  # if (p.star > 0.97){
  #   p.star <- 0.97
  # }
  metro.log <- (log(p.star^sum(z)) + log((1-p.star)^(n-sum(z))) + log(sin(5*pi*p.star)^2)) - (log(p
  hast.log <- log(dbeta(p.star,d,c)) - log(dbeta(p,d,c))
  r.log <- metro.log + hast.log
  if (log(runif(1)) < r.log){
    if (p.star != 0 & p.star != 1){
      p <- p.star
    }
  }
  simlist[i,1] <- p
}


pop1 <- numeric(n)
pop2 <- numeric(n)

j = 1
#dividing data into pop1 and pop2 as best as we can at the moment
for (j in 1:n){
  pop1[j] = z[j]*data[j]
  pop2[j] = ((1-z[j])*-1)*data[j]
  pop2[j] = -1*pop2[j]
}

#gets rid of zeros in list
pop1 <- pop1[pop1!=0]
pop2 <- pop2[pop2!=0]

#calculates length, used later
n1 <- length(pop1)
n2 <- length(pop2)

#calculates SD for population 1
#first calculated precision with gamma, then variance, finally SD
pop1.Vn <- pop1.prior.sampleSize + n1
pop1.sigmaN <- ((pop1.prior.sampleSize*pop1.prior.variance) + ((n1 - 1)*var(pop1)) + (((pop1.prior.s
prec1 <- rgamma(1,pop1.Vn/2,(pop1.Vn * pop1.sigmaN)/2)
variance1 = 1/prec1
simlist[i,2] <- sqrt(variance1)

#similar to above except uses population 2 parameters
```

```
    pop2.Vn <- pop2.prior.sampleSize + n2
    pop2.sigmaN <- ((pop2.prior.sampleSize*pop2.prior.variance) + ((n2 - 1)*var(pop2)) + (((pop2.prior.s
    prec2 <- rgamma(1,pop2.Vn/2,(pop2.Vn * pop2.sigmaN)/2)
    variance2 = 1/prec2
    simlist[i,3] <- sqrt(variance2)


    #Calculates mean of population 1
    mu1 <- ((pop1.prior.sampleSize*pop1.prior.mean)+(n1*mean(pop1)))/(pop1.prior.sampleSize+n1)
    var1 <- simlist[i,2]/(sqrt(pop1.prior.sampleSize+n1))
    simlist[i,4] <- rnorm(1,mu1,var1)

    #same as above except uses population 2 parameters
    mu2 <- ((pop2.prior.sampleSize*pop2.prior.mean)+(n2*mean(pop2)))/(pop2.prior.sampleSize+n2)
    var2 <- simlist[i,3]/(sqrt(pop2.prior.sampleSize+n2))
    simlist[i,5] <- rnorm(1,mu2,var2)



  }
  return (simlist)
}
```

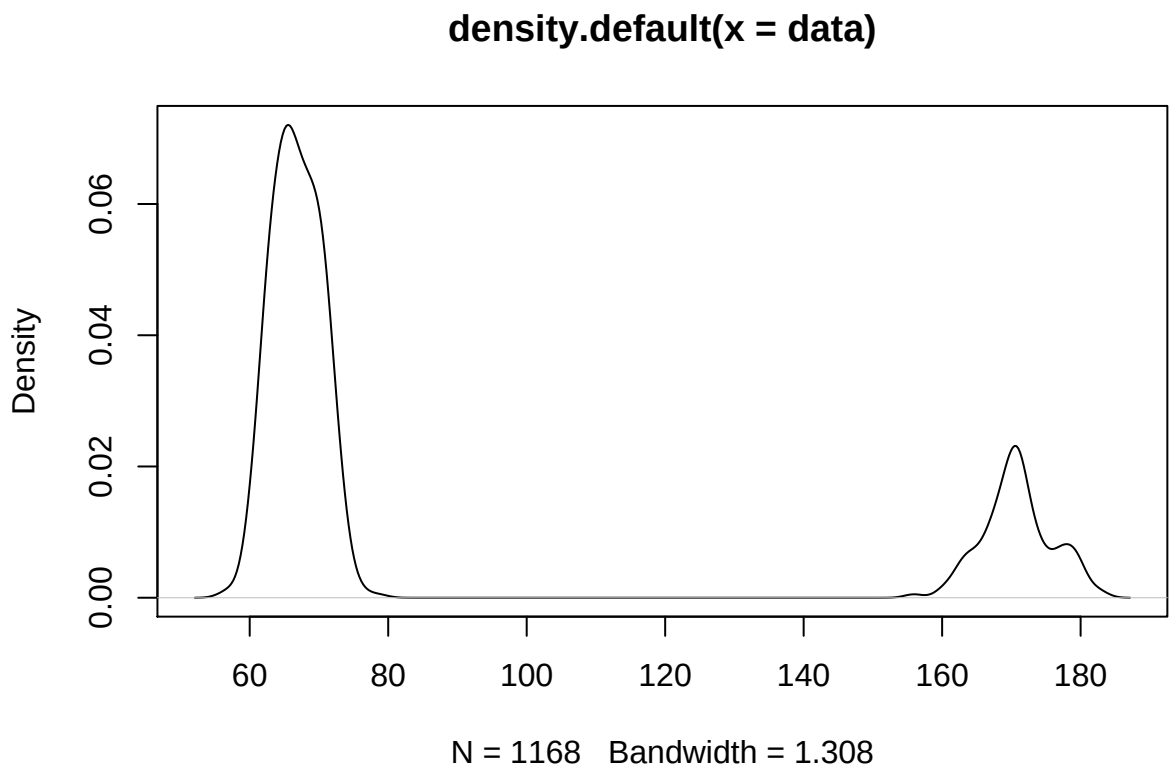
Function that outputs useful results and plots

```
printResults <- function(simlist){
  #plots and gives means of parameters
  plot(simlist[,1], main = "Chain for p", type = "l", ylab = "P")
  plot(density(simlist[,1]), main = "Distribution of p")
  print("The estimated value of p is")
  print(mean(simlist[,1]))
  plot(density(simlist[,2]),main = "Distribution of SD of Pop1")
  print("The estimated SD of population 1 is")
  print(mean(simlist[,2]))
  plot(density(simlist[,3]),main = "Distribution of SD of Pop2")
  print("The estimated SD of population 2 is")
  print(mean(simlist[,3]))
  plot(density(simlist[,4]),main = "Distribution of Mean of Pop1")
  print("The estimated mean of population 1 is")
  print(mean(simlist[,4]))
  plot(density(simlist[,5]),main = "Distribution of Mean of Pop2")
  print("The estimated mean of population 2 is")
  print(mean(simlist[,5]))
}
```

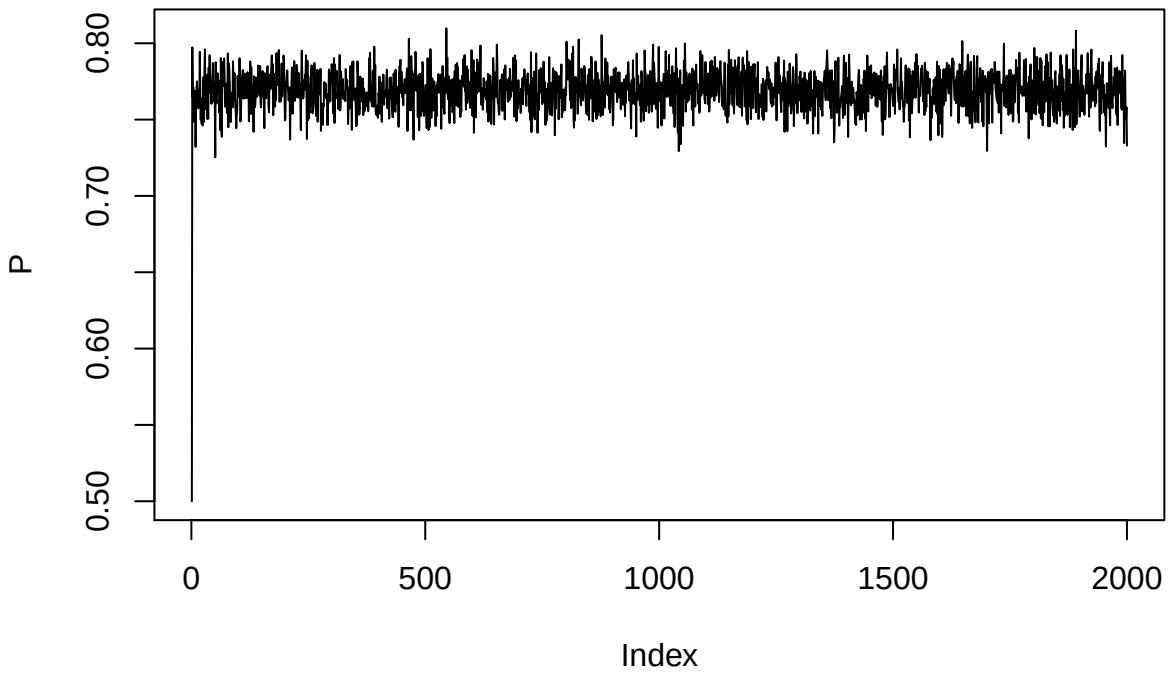
This chunck runns a Gibbs sampler on data comprised of kids heights and adult heights

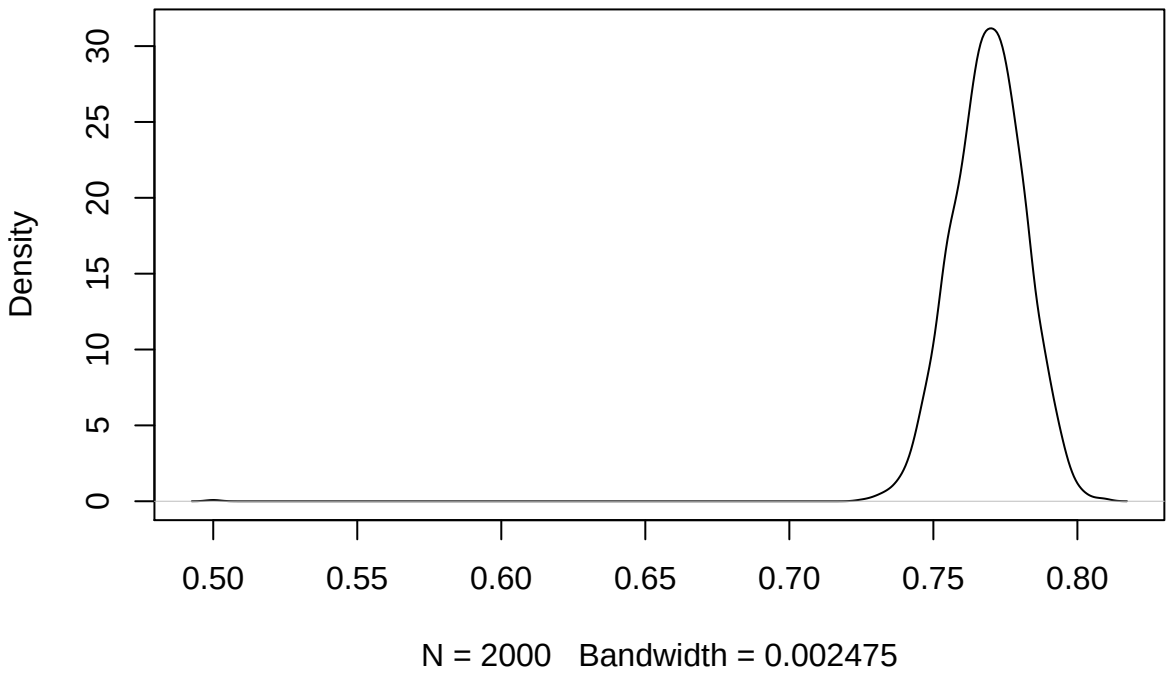
```
heightData = read.csv("height.csv")
adultHeight = heightData$Human.Height..University.of.Tuebingen..2015....cm.[heightData$Year > 1950]
kidHeight = read.csv("kidHeight.csv")
data = c(adultHeight,kidHeight$height)
plot(density(data))

truep <- length(kidHeight$height)/(length(adultHeight) + length(kidHeight$height))
```

```r
pop1.trueMean <- mean(kidHeight$height)
pop1.trueSD <- sqrt(var(kidHeight$height))

pop2.trueMean <- mean(adultHeight)
pop2.trueSD <- sqrt(var(adultHeight))


#initialize values for prior for population 1
pop1.prior.sampleSize <- 10
pop1.prior.trueMean <- 80
pop1.prior.trueSD <- 20

#initialize values for prior for population 2
pop2.prior.sampleSize <- 10
pop2.prior.trueMean <- 160
pop2.prior.trueSD <- 20

simlistHeight <- gibbs.MH(data,2000,0)
```

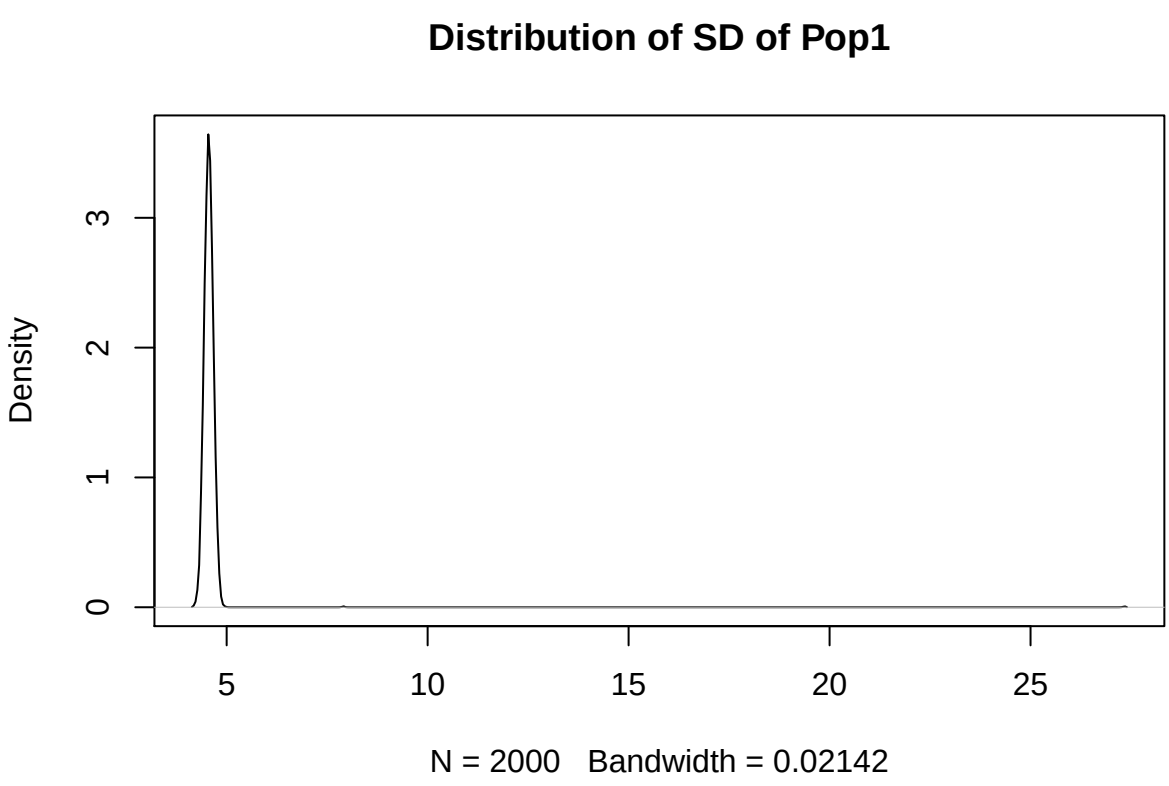



```r
printResults(simlistHeight)
```

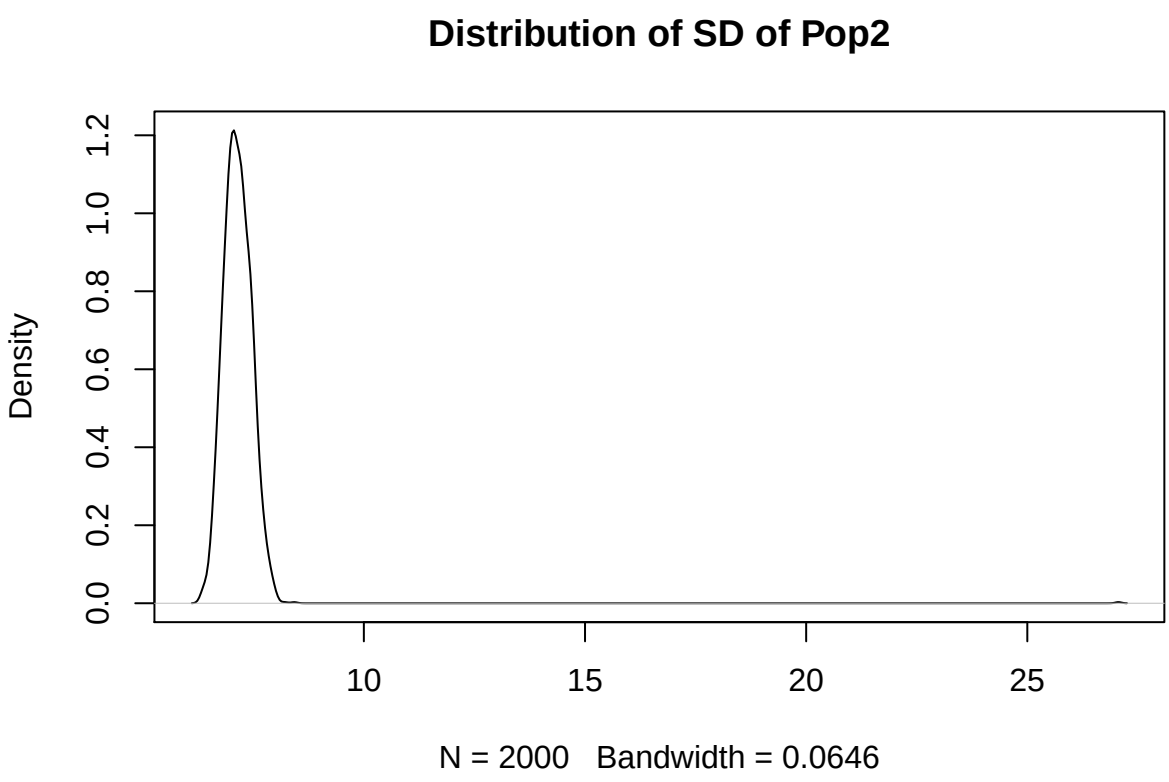

Chain for p
P
0.50
0.60
0.70
0.80
0
500
1000
1500
2000
Index



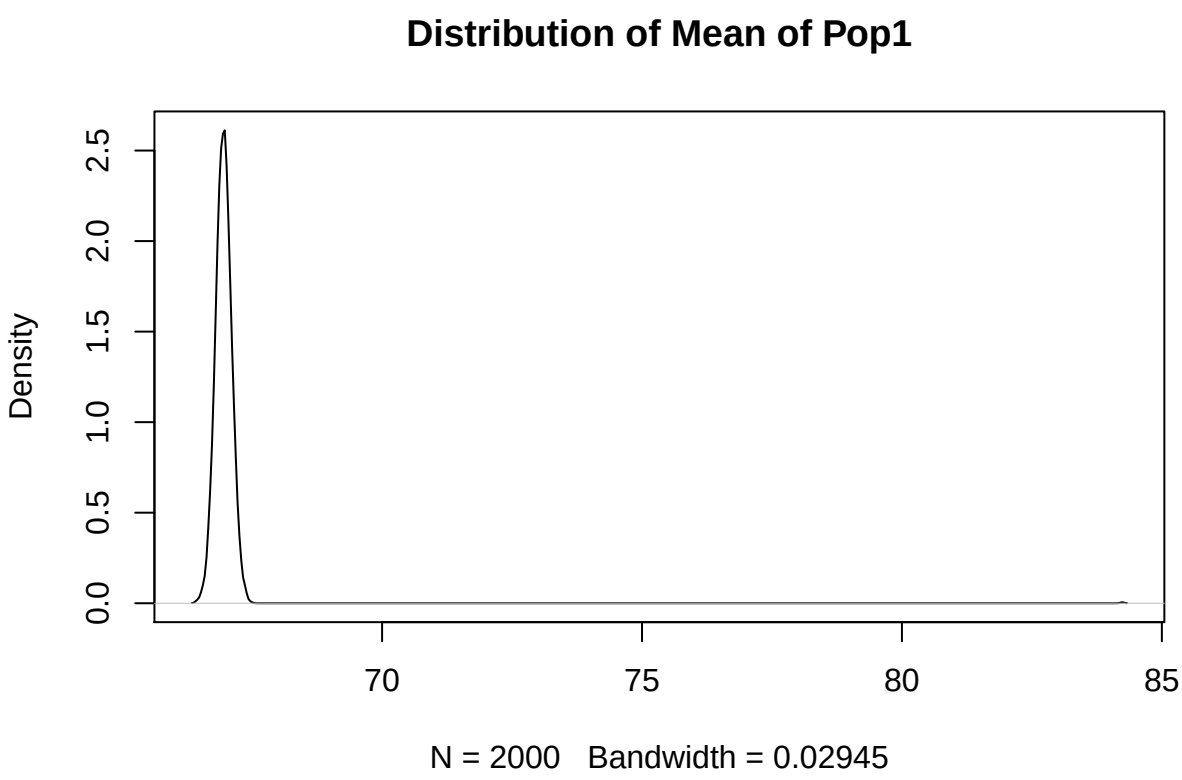

```
## [1] "The estimated value of p is"
## [1] 0.7689908
```



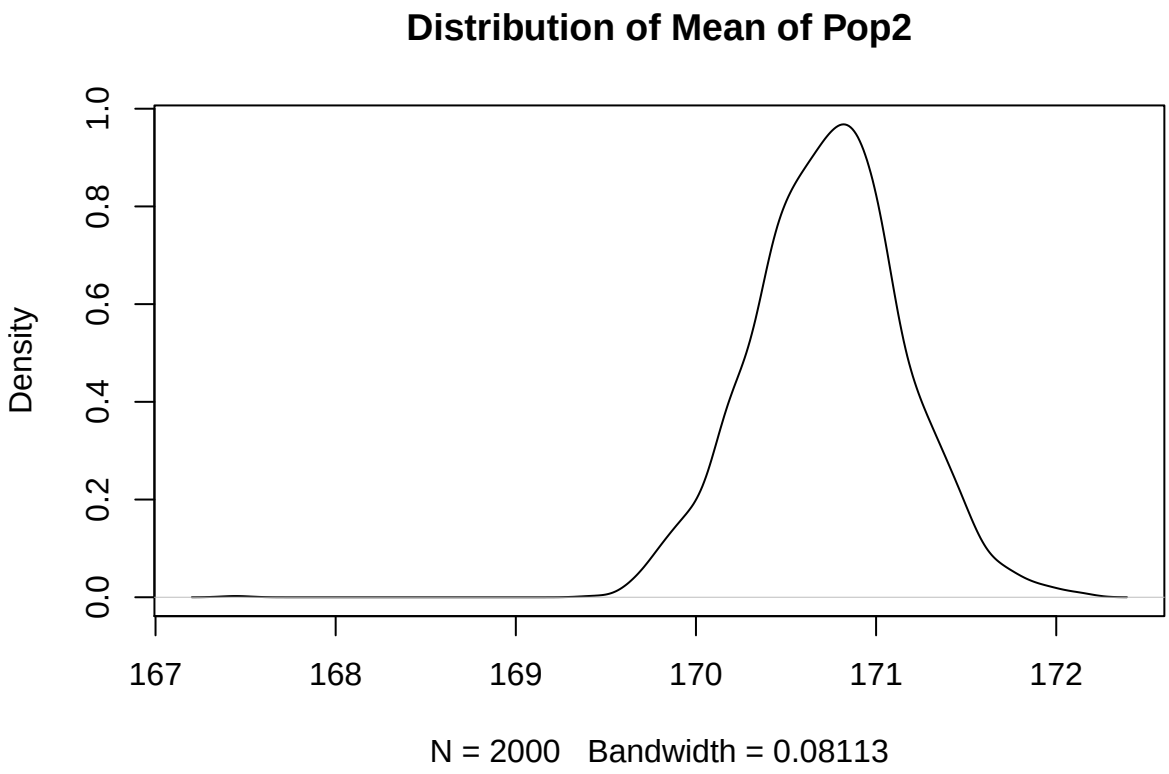

```
## [1] "The estimated SD of population 1 is"
## [1] 4.567972
```




```
## [1] "The estimated SD of population 2 is"
## [1] 7.155887
```




```
## [1] "The estimated mean of population 1 is"
## [1] 66.95931
```




```
## [1] "The estimated mean of population 2 is"
## [1] 170.7349
```

```
print("The true value of p is")
```

```
## [1] "The true value of p is"
```

```
truep
```

```
## [1] 0.7688356
```

```
print("The true value of the SD for population 1 is")
```

```
## [1] "The true value of the SD for population 1 is"
```

```
sqrt(var(kidHeight$height))
```

```
## [1] 3.582918
```

```
print("The true value of the SD for population 2 is")
```

```
## [1] "The true value of the SD for population 2 is"
```

```
sqrt(var(heightData$Human.Height..University.of.Tuebingen..2015....cm.))
```

```
## [1] 4.638687
```

```
print("The true value of the mean of population 1 is")
```

```
## [1] "The true value of the mean of population 1 is"
```

```r
mean(kidHeight$height)
```

```
## [1] 66.76069
```

```r
print("The true value of the mean of population 2 is")
```

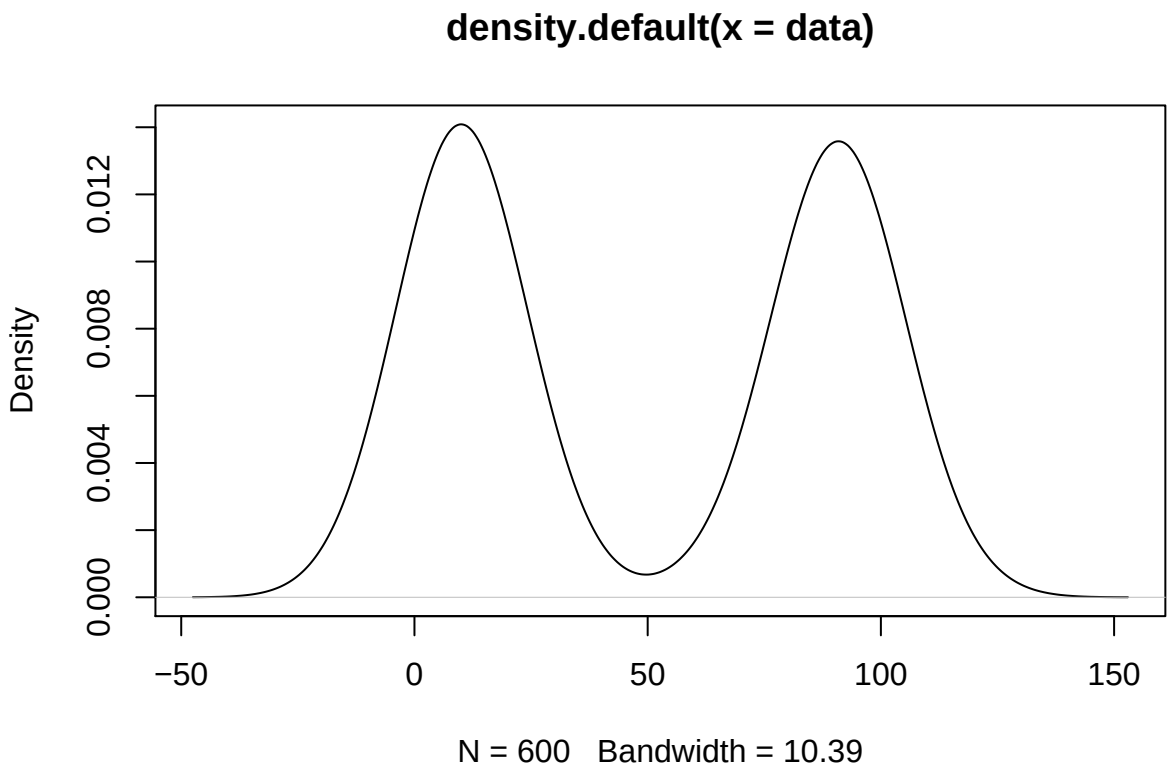
```
## [1] "The true value of the mean of population 2 is"
```

```r
mean(heightData$Human.Height..University.of.Tuebingen..2015....cm.)
```

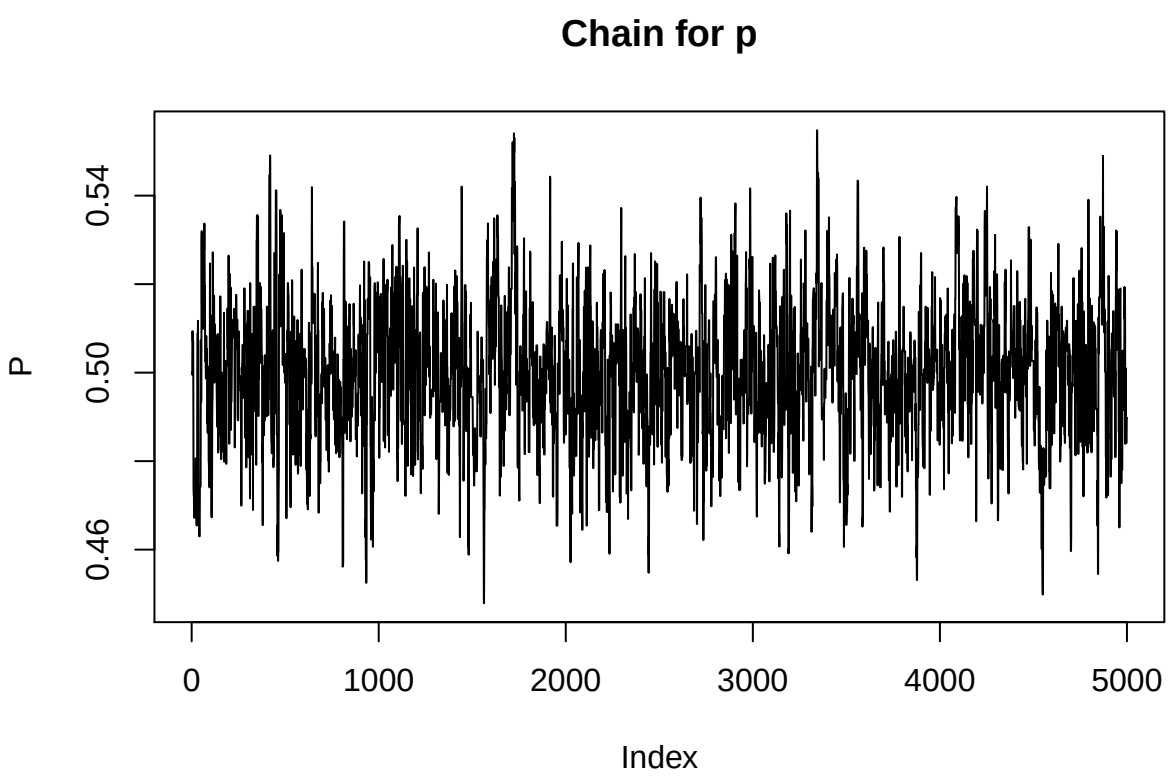
```
## [1] 167.6473
```

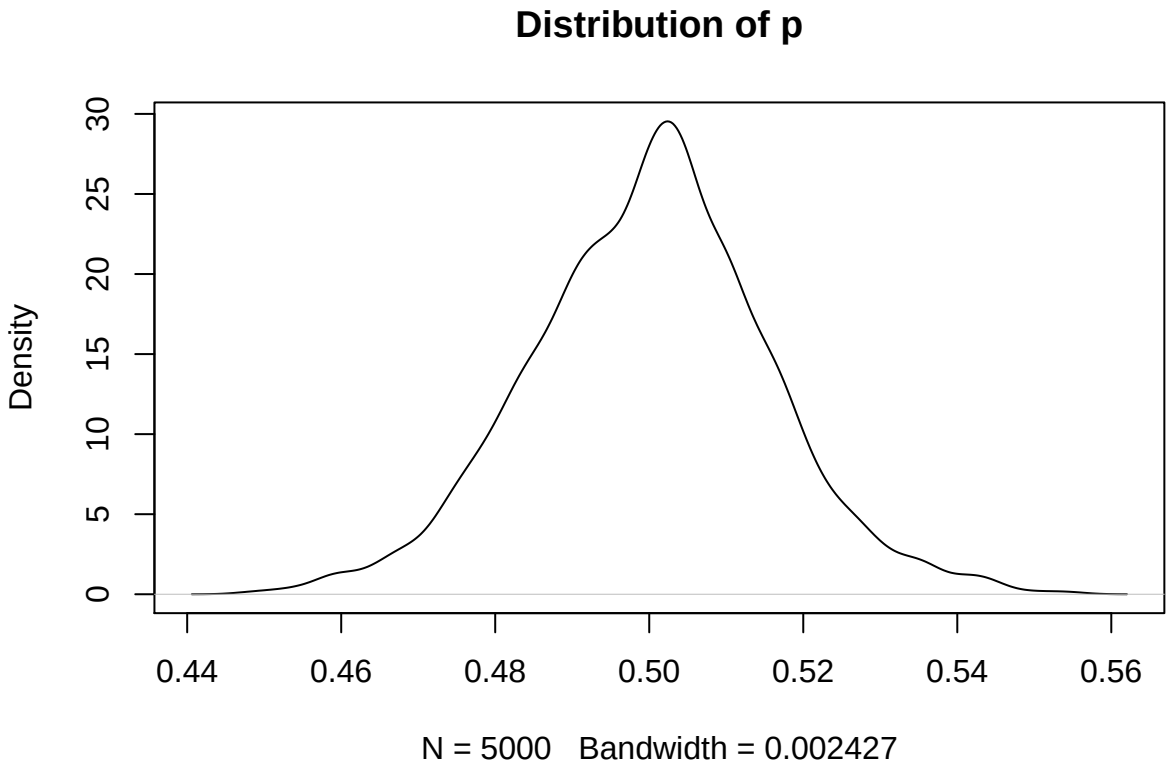
This chunk runs a combination Gibbs sampler/Metropolis-Hastings algorithm on simulated normal data

```r
obs1 <- 300
obs2 <- 300
#initialize values for population 1
pop1.trueMean <- 10
pop1.trueSD <- 10

#initialize values for population 2
pop2.trueMean <- 90
pop2.trueSD <- 10

#creates data, then binds together to create normal mixture model
samp1 <- rnorm(obs1,pop1.trueMean,pop1.trueSD)
samp2 <- rnorm(obs2,pop2.trueMean,pop2.trueSD)
data <- c(samp1,samp2)
plot(density(data))
```

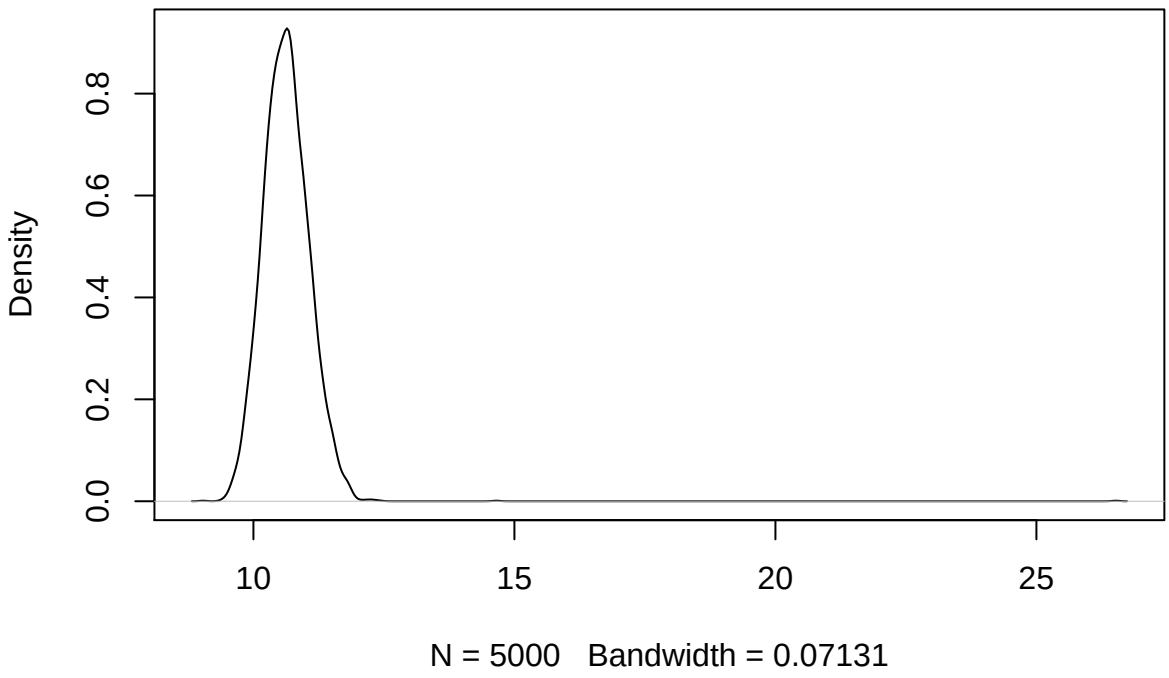



```
#initialize values for prior for population 1
pop1.prior.sampleSize <- 10
pop1.prior.trueMean <- 20
pop1.prior.trueSD <- 20

#initialize values for prior for population 2
pop2.prior.sampleSize <- 10
pop2.prior.trueMean <- 80
pop2.prior.trueSD <- 20

simlistSim <- gibbs.MH(data,5000,1)
printResults(simlistSim)
```

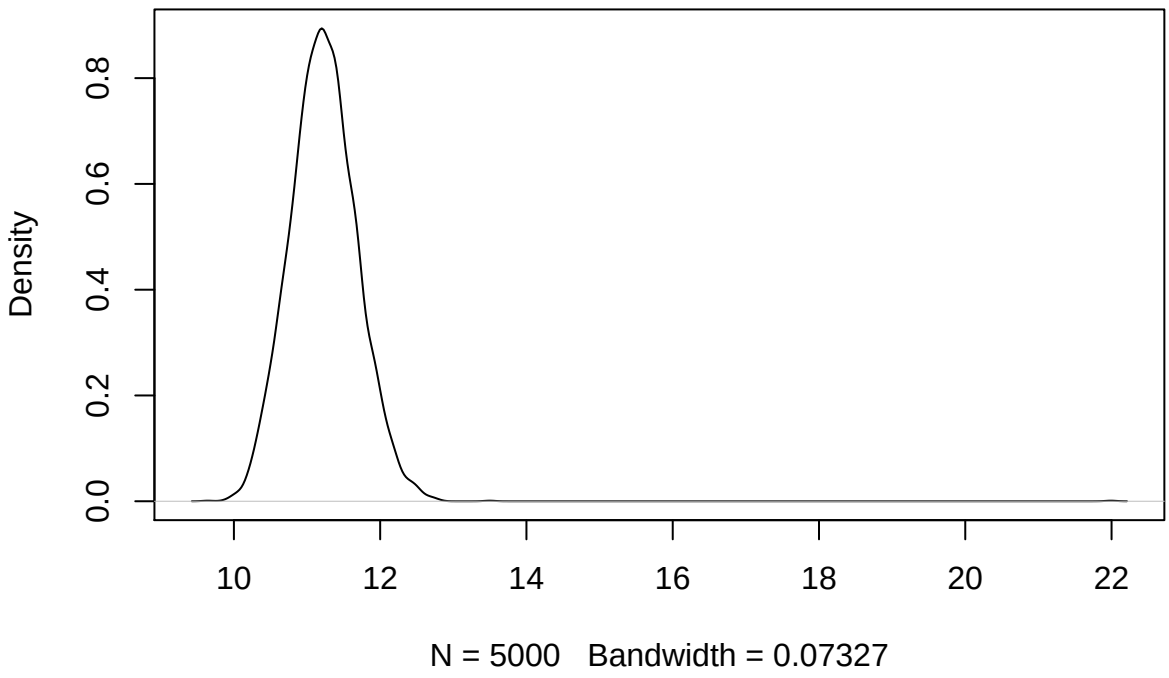

Chain for p
P
0.46
0.50
0.54
0
1000
2000
3000
4000
5000
Index



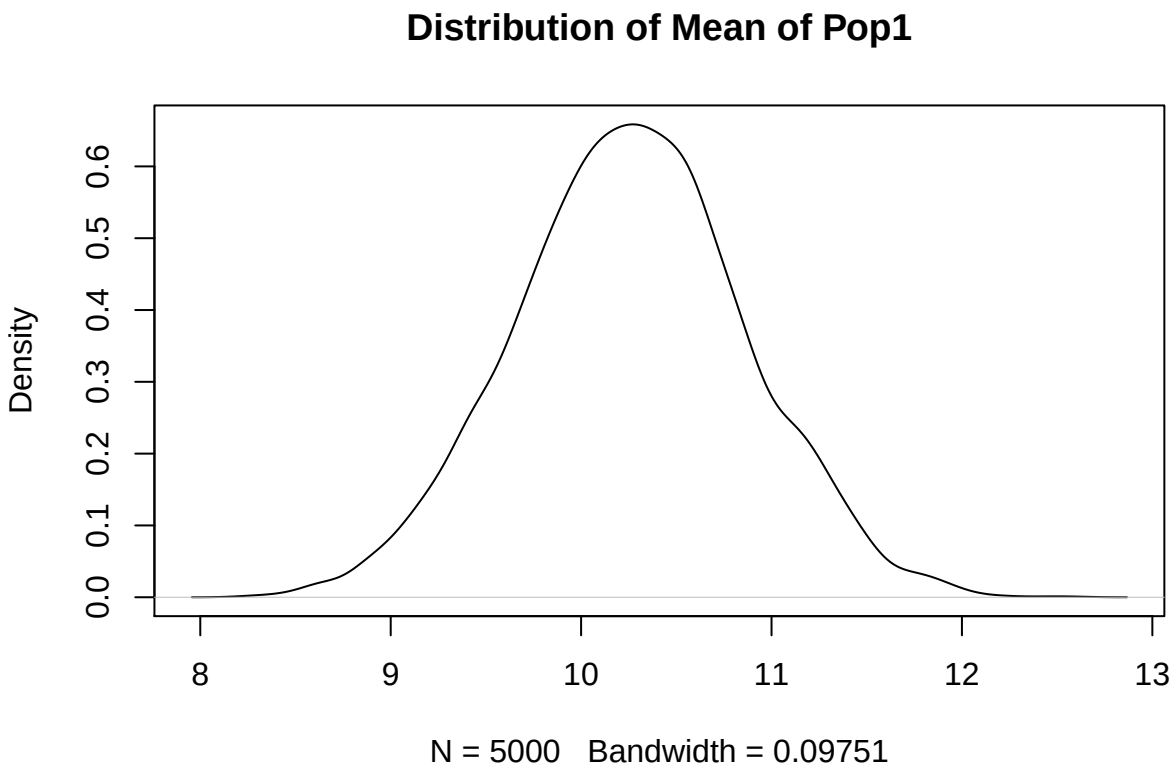

```
## [1] "The estimated value of p is"
## [1] 0.5000687
```



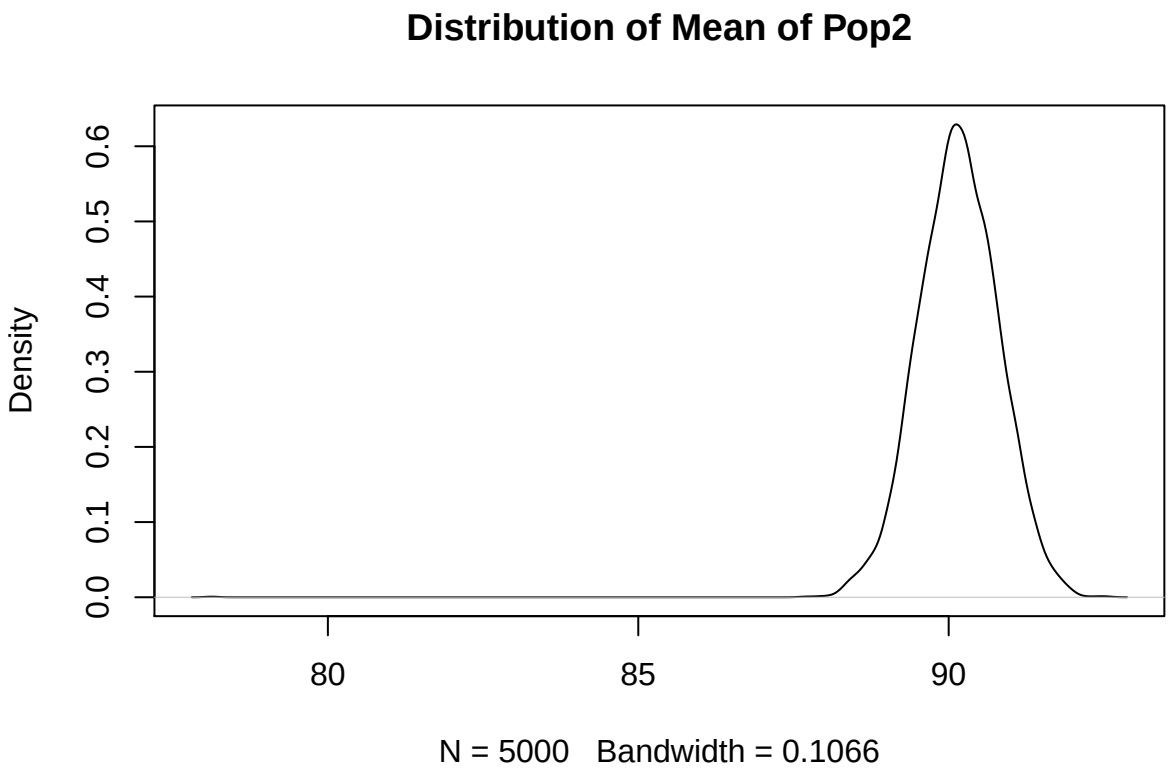

```
## [1] "The estimated SD of population 1 is"
## [1] 10.63307
```




```
## [1] "The estimated SD of population 2 is"
## [1] 11.23092
```




```
## [1] "The estimated mean of population 1 is"
## [1] 10.25959
```




```
## [1] "The estimated mean of population 2 is"
## [1] 90.15386
```

```
print("The true value of p is")
```

```
## [1] "The true value of p is"
```

```
obs1/(obs1+obs2)
```

```
## [1] 0.5
```

```
print("The true value of the SD for population 1 is")
```

```
## [1] "The true value of the SD for population 1 is"
```

```
pop1.trueSD
```

```
## [1] 10
```

```
print("The true value of the SD for population 2 is")
```

```
## [1] "The true value of the SD for population 2 is"
```

```
pop2.trueSD
```

```
## [1] 10
```

```
print("The true value of the mean of population 1 is")
```

```
## [1] "The true value of the mean of population 1 is"
```

```
pop1.trueMean
```

```
## [1] 10
```

```
print("The true value of the mean of population 2 is")
```

```
## [1] "The true value of the mean of population 2 is"
```

```
pop2.trueMean
```

```
## [1] 90
```