

Assignment Covering Chapter 4 of *Bayesian Computation with R*

Chris Hayduk

May 10, 2019

1. **Chapter 4: Multiparameter Models** Write your own code for the following:

- (a) `normchi2post()` on pg. 64
- (b) `mycontour()` on pg. 64
- (c) `normpostsim()` on pg. 64
- (d) `rdirichlet()` on pg. 66
- (e) `beta.select()` on pg. 71
- (f) `logisticpost()` on pg. 72
- (g) `simcontour()` on pg. 73
- (h) `howardprior()` on pg. 77

Solution:

```

(a) normchi2post <- function(theta, data){
  mean <- theta[1]
  variance <- theta[2]

  y_bar <- mean(data)

  S <- sum((data-y_bar)^2)
  n <- length(data)

  posterior_density <- 1/((variance^(n/2+1))) *
    exp(-1/(2*variance)*(S+n*(mean-y_bar)^2))

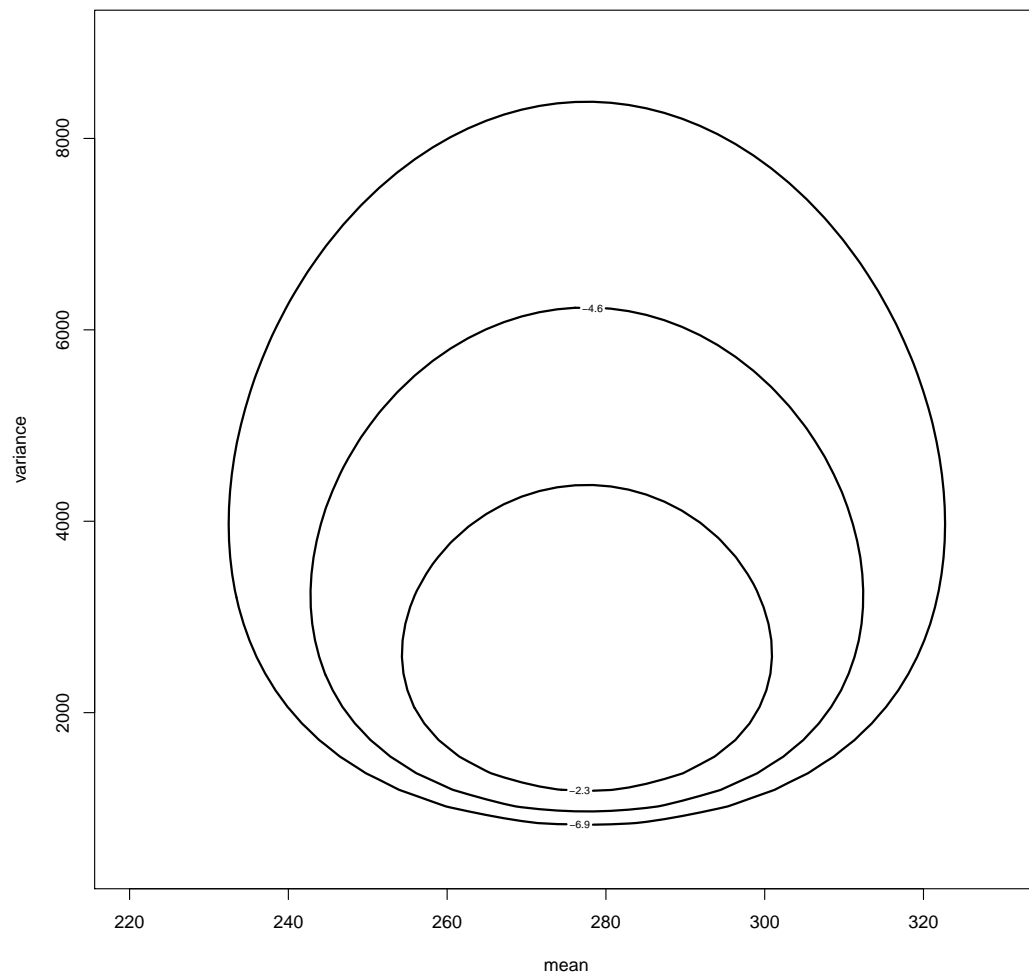
  log_density <- log(posterior_density)
  #print(log_density)
  return(log_density)
}

#Example run

data("marathontimes")
attach(marathontimes)

d <- LearnBayes::mycontour(normchi2post, c(220, 330, 500, 9000),
  time, xlab="mean",
  ylab="variance")

```



- (b) Needed to hide the `mycontour()` implementation due to the length of the code. It was causing LaTeX issues when compiling the document.

```

(c) normpostsim <- function(data, m){
  S <- sum((data-mean(data))^2)

  n <- length(data)

  sigma2 <- S/rchisq(m, n-1)

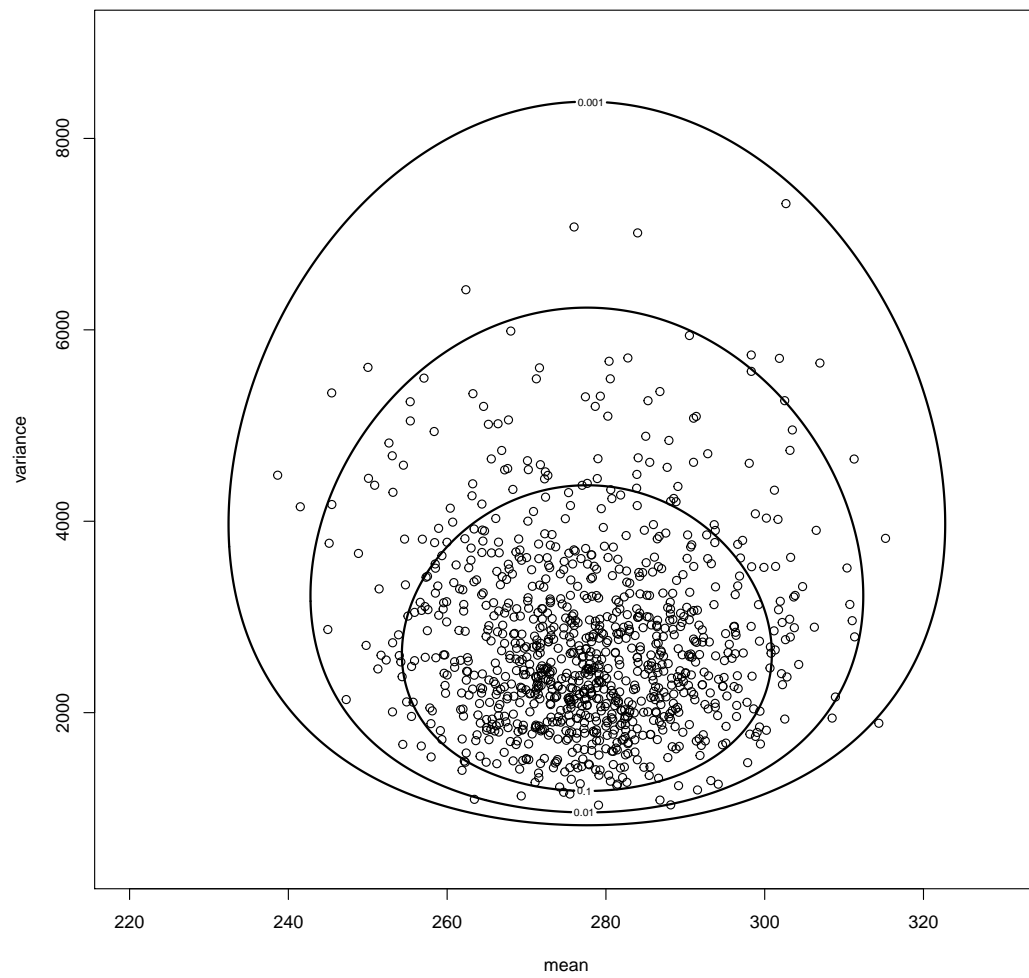
  mu <- rnorm(m, mean = mean(data), sd = sqrt(sigma2)/sqrt(n))

  results <- data.frame(mu = mu, sigma2 = sigma2)

  return(results)
}

#Example run
results <- normpostsim(time, 1000)
mycontour(normchi2post, c(220, 330, 500, 9000), time,
           size_grid = 1000, levels <- c(0.1, 0.01, 0.001),
           log_func = TRUE,
           labels = c(0.1, 0.01, 0.001),
           xlab="mean", ylab="variance")
points(results$mu, results$sigma2)

```



```
(d) rdirichlet <- function(m, alpha){
  theta <- matrix(nrow = m, ncol = length(alpha))

  for(i in 1:length(alpha)){
    theta[,i] <- rgamma(m, alpha[i], 1)
  }

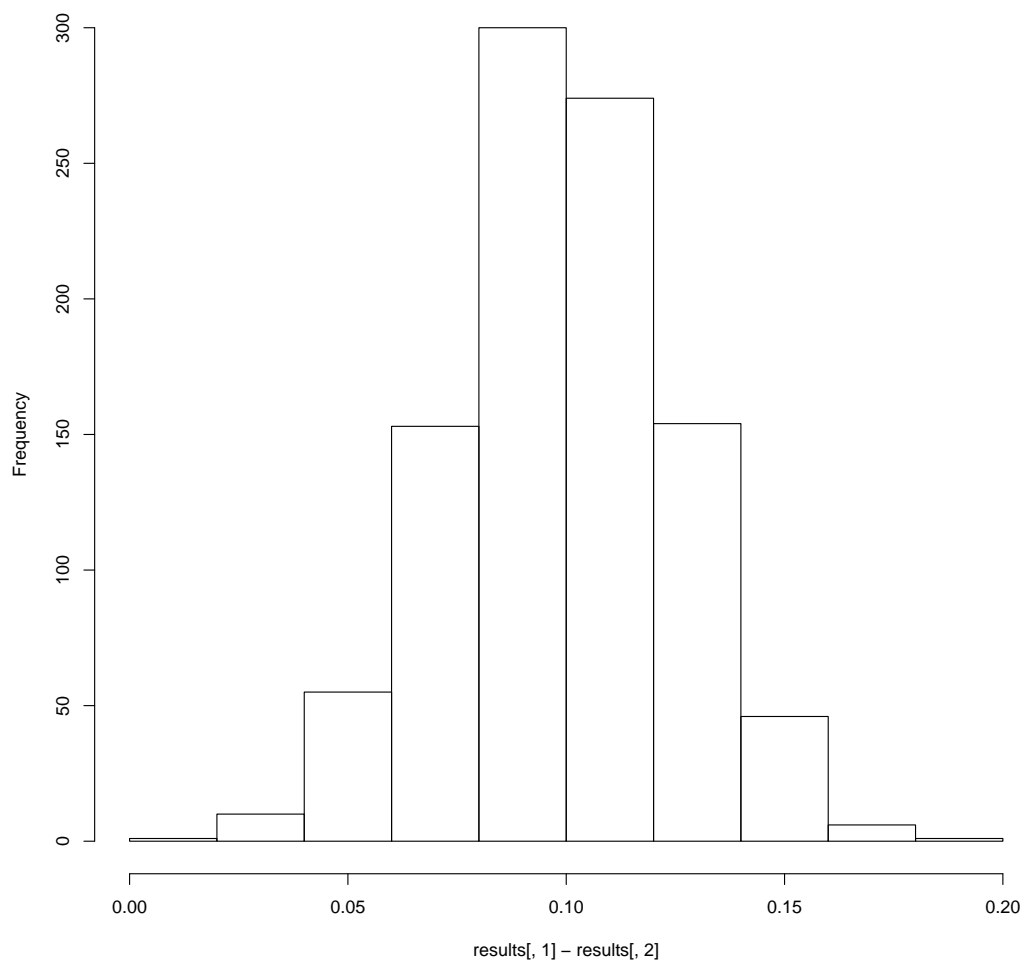
  for(i in 1:nrow(theta)){
    T = sum(theta[i,])

    theta[i,] <- theta[i,]/T
  }

  return(theta)
}

results <- rdirichlet(1000, c(728,584,138))

hist(results[,1] - results[,2], main = "")
```



```
(e) beta.select <- function(quantile1, quantile2){  
  }  
  
  #Example run  
  beta.select(list(p=0.5, x=0.2), list(p=0.9, x=0.5))  
  ## NULL
```

```

(f) logisticpost <- function(beta, data){
  post <- 1

  for(i in 1:nrow(data)){
    p <- exp(beta[1] + beta[2]*data[i, 1])

    p <- p/(1+exp(beta[1] + beta[2]*data[i, 1]))

    y <- data[i, 3]
    n <- data[i, 2]

    post <- post * (p^y * (1-p)^(n-y))
  }

  return(log(post))
}

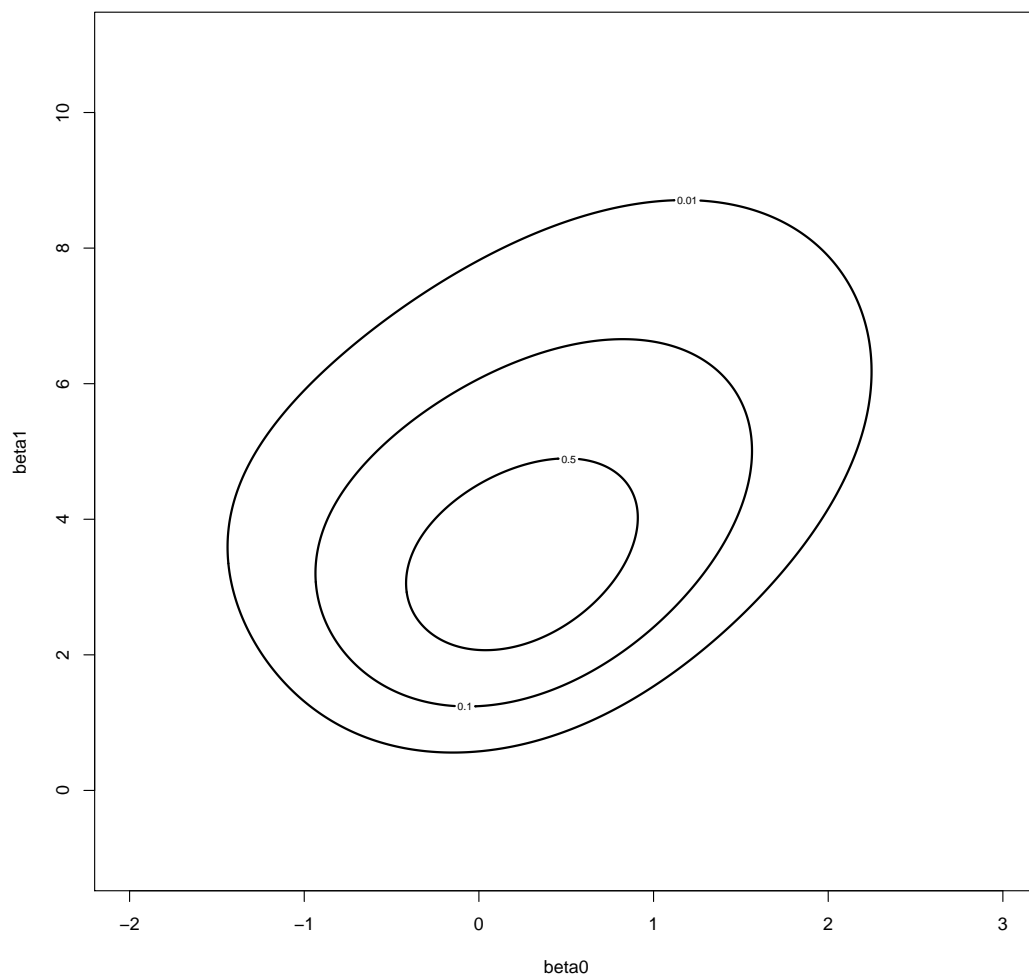
#Example run
x <- c(-0.86, -0.3, -0.05, 0.73)
n <- c(5, 5, 5, 5)
y <- c(0, 1, 3, 5)

data <- cbind(x, n, y)

prior <- rbind(c(-0.7, 4.68, 1.12),
               c(0.6, 2.84, 2.10))
data.new <- rbind(data, prior)

mycontour(logisticpost, c(-2, 3, -1, 11),
           data= data.new,
           size_grid = 1000,
           levels = c(0.5, 0.1, 0.01),
           log_func = TRUE,
           labels = c(0.5, 0.1, 0.01),
           xlab="beta0", ylab="beta1")

```



```

(g) simcontour <- function(fn, limits, data,
                          size_grid, m, log_func = FALSE)
{
  LOGF = function(theta, data) {
    if (is.matrix(theta) == TRUE) {
      val = matrix(0, c(dim(theta)[1], 1))

      for (j in 1:dim(theta)[1]){
        if(log_func == FALSE)
          val[j] = log(fn(theta[j,], data))
        else
          val[j] = fn(theta[j,], data)
      }
    }
    else{
      if(log_func == FALSE)
        val = log(fn(theta, data))
      else
        val = fn(theta, data)
    }
    return(val)
  }

  ng = size_grid
  x0 = seq(limits[1], limits[2], len = ng)
  y0 = seq(limits[3], limits[4], len = ng)
  X = outer(x0, rep(1, ng))
  Y = outer(rep(1, ng), y0)
  n2 = ng^2

  Z = LOGF(cbind(X[1:n2], Y[1:n2]), data)
  Z = matrix(Z, c(ng, ng))

  d = cbind(X[1:n2], Y[1:n2], Z[1:n2])
  prob = d[, 3]
  prob = exp(prob)
  prob = prob/sum(prob)

  i = sample(n2, m, replace = TRUE, prob = prob)

  return(list(x = d[i, 1], y = d[i, 2]))
}

```

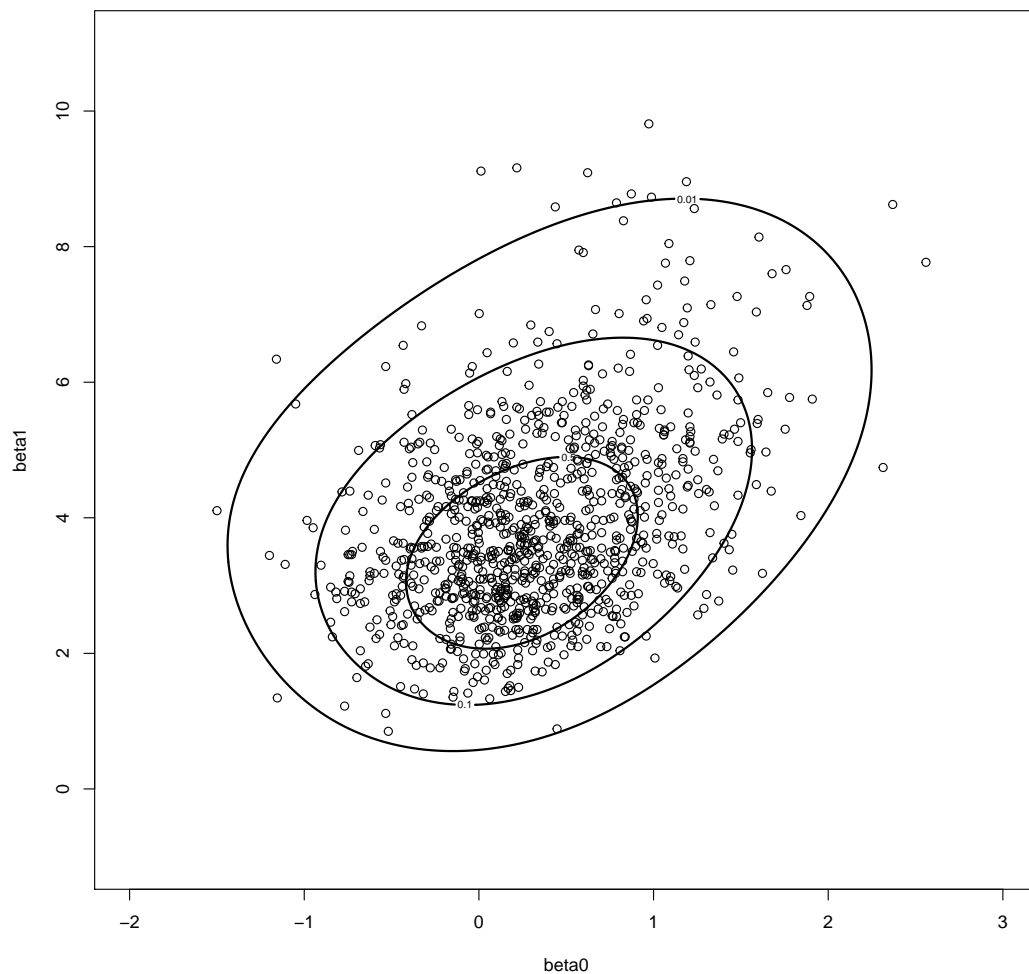
```

mycontour(logisticpost, c(-2, 3, -1, 11),
           data= data.new,
           size_grid = 1000,
           levels = c(0.5, 0.1, 0.01),
           log_func = TRUE,
           labels = c(0.5, 0.1, 0.01),
           xlab="beta0", ylab="beta1")

s <- simcontour(logisticpost, c(-2, 3, -1, 11),
                data.new, 1000, 1000, TRUE)

points(s)

```



```

(h) howardprior <- function(xy, par){
  alpha <- par[1]
  beta <- par[2]
  gamma <- par[3]
  delta <- par[4]
  sigma <- par[5]

  p1 <- xy[1]
  p2 <- xy[2]

  theta1 <- log(p1/(1-p1))
  theta2 <- log(p2/(1-p2))

  mu <- 1/sigma * (theta1 - theta2)

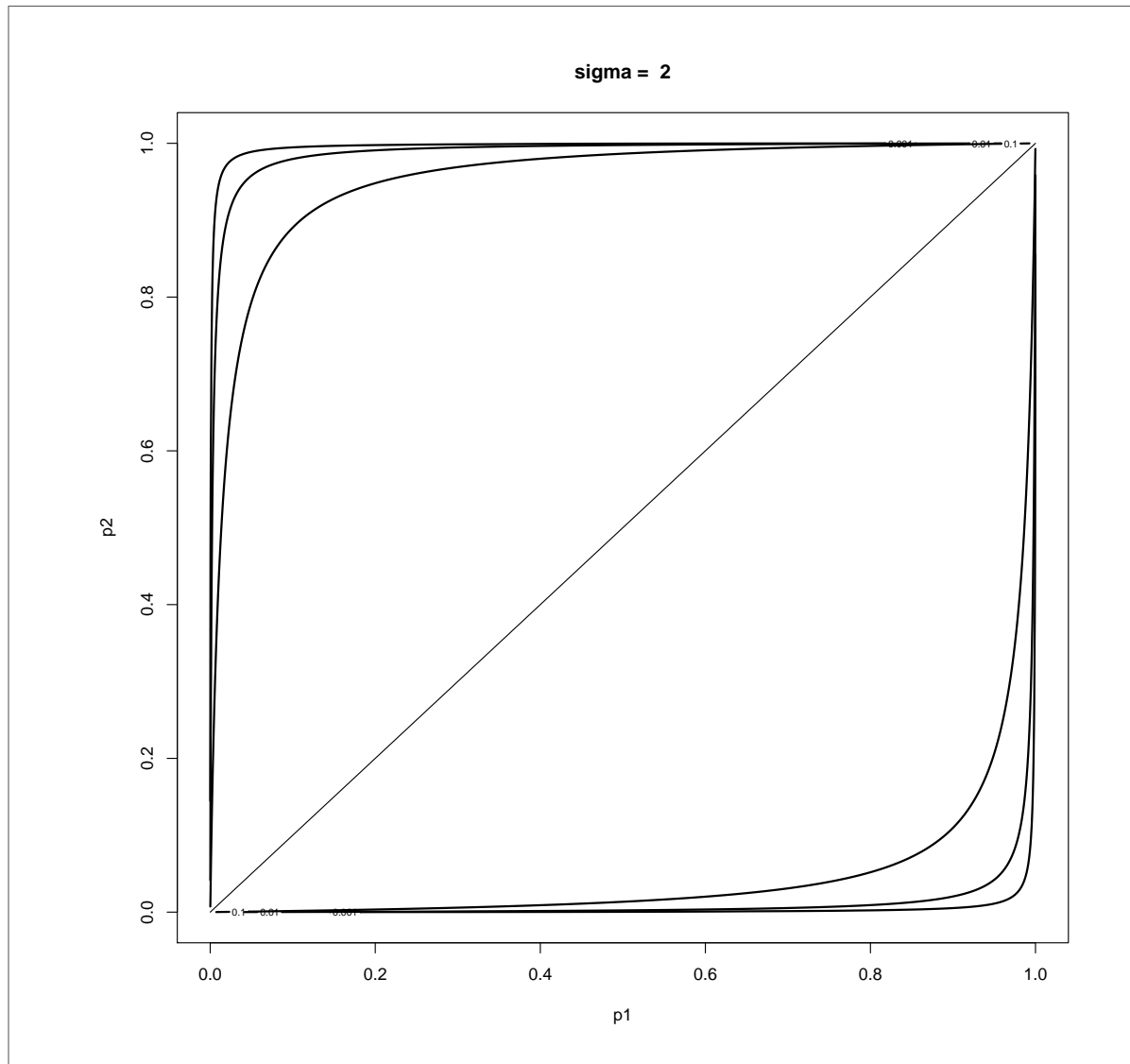
  value <- exp(-1/2 * mu^2) * p1^(alpha-1) * (1-p1)^(beta-1) *
    p2^(gamma-1) * (1 - p2)^(delta-1)

  return(log(value))
}

plo <- 0.0001
phi <- 0.9999
sigma <- 2

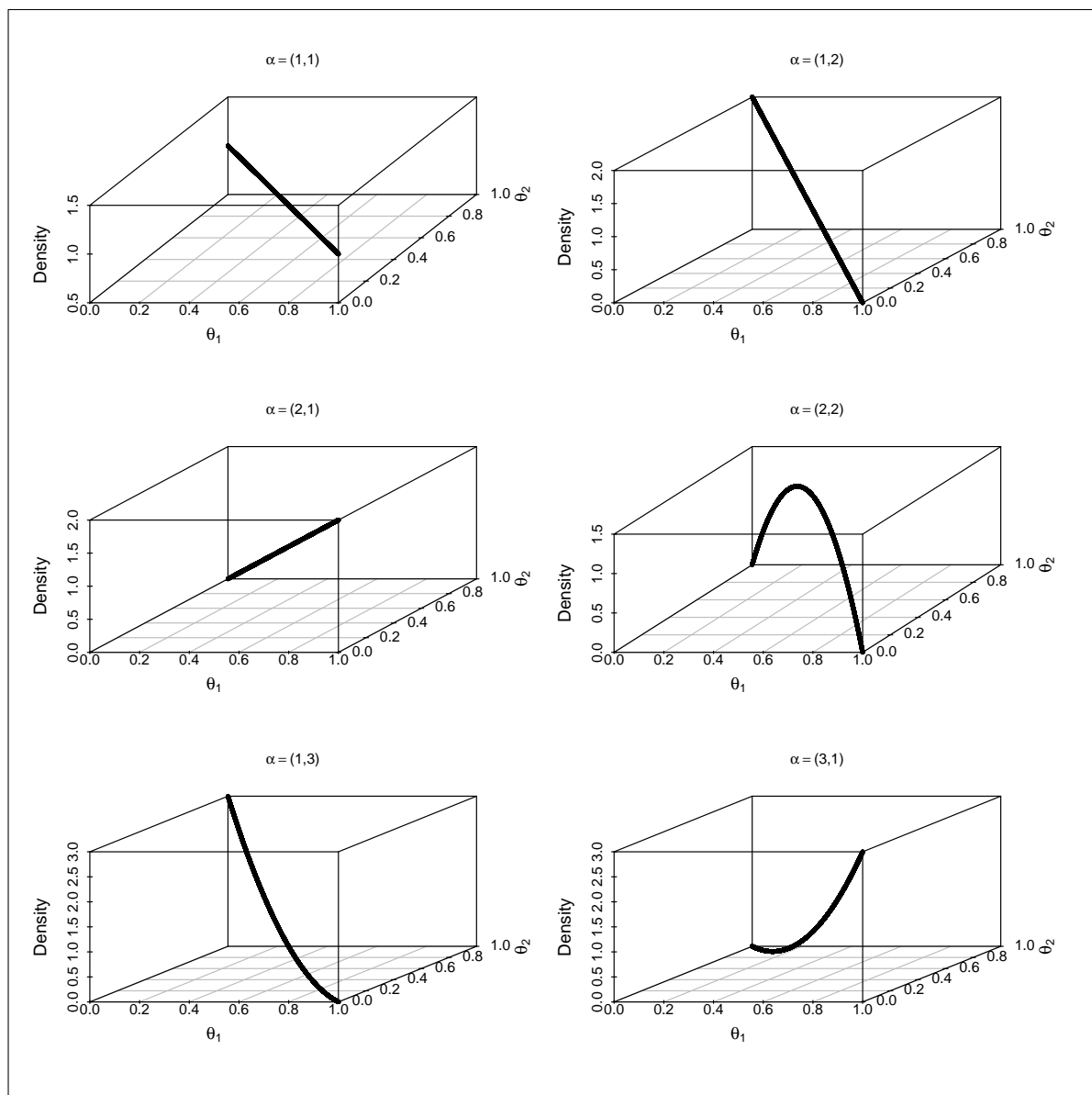
mycontour(howardprior, c(plo, phi, plo, phi),
  c(1, 1, 1, 1, sigma), 1000,
  levels=c(0.1, 0.01, 0.001),
  log_func = TRUE,
  labels = c(0.1, 0.01, 0.001),
  main=paste("sigma = ", as.character(sigma)),
  xlab = "p1", ylab = "p2")
lines(c(0,1),c(0,1))

```



2. **Chapter 4: Multiparameter Models** Pg. 66 introduces the Dirichlet distribution. Draw pdfs of this distribution for various parameter values using the functions in the R package. Find some examples on the internet of where the Dirichlet distribution is used in statistics.

Solution:



3. Do all CH. 4 exercises.

Solution:

```

1. #a)
data <- c(9.0, 8.5, 7.0, 8.5, 6.0, 12.5, 6.0, 9.0, 8.5, 7.5,
          8.0, 6.0, 9.0, 8.0, 7.0, 10.0, 9.0, 7.5, 5.0, 6.5)
n <- length(data)
y_bar <- mean(data)

s_2 <- sum((data-y_bar)^2)*(1/(n-1))

sigma_2 <- rinvchisq(1000, n-1, s_2)

mu <- rnorm(1000, y_bar, sigma_2/n)

#b)
quantile(mu, c(0.05, 0.95))

##          5%          95%
## 7.652164 8.228933

sigma <- sqrt(sigma_2)

quantile(sigma, c(0.05, 0.95))

##          5%          95%
## 1.344385 2.341904

#c)
p_75 <- mu + 0.674*sigma

mean(p_75)

## [1] 9.129789

sd(p_75)

## [1] 0.2714589

```

2. (a) Because we know that the two samples are independent with distributions $N(\mu_1, \sigma_1)$ and $N(\mu_2, \sigma_2)$, the posterior distribution is given by,

$$\begin{aligned}
 p(\theta|y) &= p(\theta)p(y|\theta) = \frac{1}{\sigma_1^2\sigma_2^2} \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}} \\
 &= \frac{1}{\sigma_1^2\sigma_2^2} N(\mu_1, \sigma_1)N(\mu_2, \sigma_2)
 \end{aligned}$$

- (b) Thus, we can sample from the joint distribution by sampling from $N(\mu_1, \sigma_1)$

and $N(\mu_2, \sigma_2)$, multiplying them together, and then multiplying by the prior.

```
#c)
males <- c(120, 107, 110, 116, 114, 111, 113, 117, 114, 112)

females <- c(110, 111, 107, 108, 110, 105, 107, 106, 111, 111)

S1 <- var(males)

S2 <- var(females)

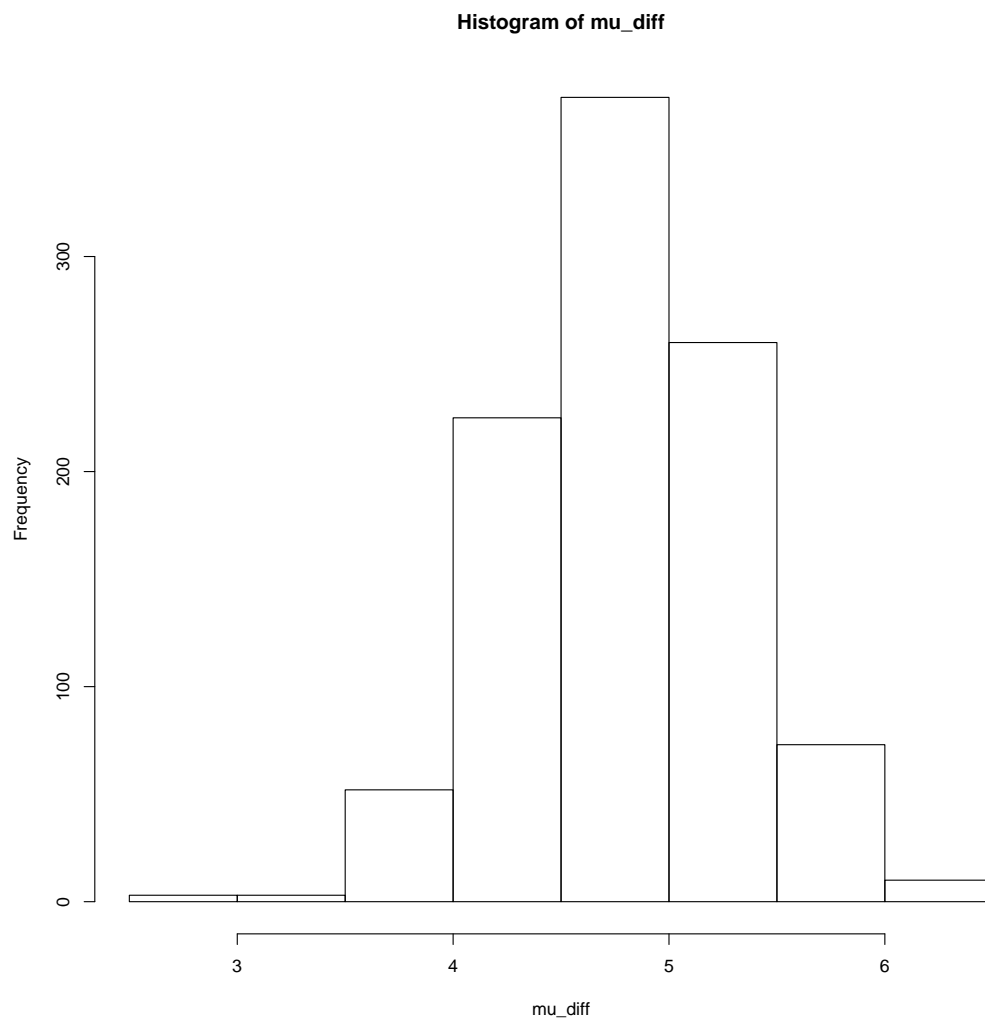
n <- length(males)

sigma2_males <- S1/rchisq(1000,n-1)
sigma2_females <- S2/rchisq(1000,n-1)

mu_males <- rnorm(1000, mean=mean(males),
                 sd = sqrt(sigma2_males)/sqrt(n))
mu_females <- rnorm(1000, mean=mean(females),
                  sd = sqrt(sigma2_females)/sqrt(n))

mu_diff <- mu_males - mu_females

hist(mu_diff)
```



All of the differences between the simulated values of μ for male and female golden jackal mandible lengths are well-above 0. Thus, we can conclude that the mean length of the mandible for males is higher than the mean length for females.

3. (a) Since the prior is uniform, we will assign $p(\theta) = p(p_N, p_S) = 1$ for all

values of p_N and p_S such that $0 \leq p_N \leq 1$ and $0 \leq p_S \leq 1$.

$$\begin{aligned}
 p(p_N, p_S | y_N, y_S, n_N, n_S) &= p(p_N, p_S) p(y_N, y_S | p_N, p_S, n_N, n_S) \\
 &= p(y_N | n_N, p_N) p(y_S | n_S, p_S) \\
 &= \binom{n_N}{y_N} p^{y_N} (1-p)^{n_N-y_N} \binom{n_S}{y_S} p^{y_S} (1-p)^{n_S-y_S} \\
 &= \text{Beta}(y_N + 1, n_N - y_N + 1) \text{Beta}(y_S + 1, n_S - y_S + 1) \\
 &= \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha, \beta)}
 \end{aligned}$$

```

#b)
y_N <- 1601
n_N <- 162527 + 1601

p_N <- rbeta(1000, y_N + 1, n_N - y_N + 1)

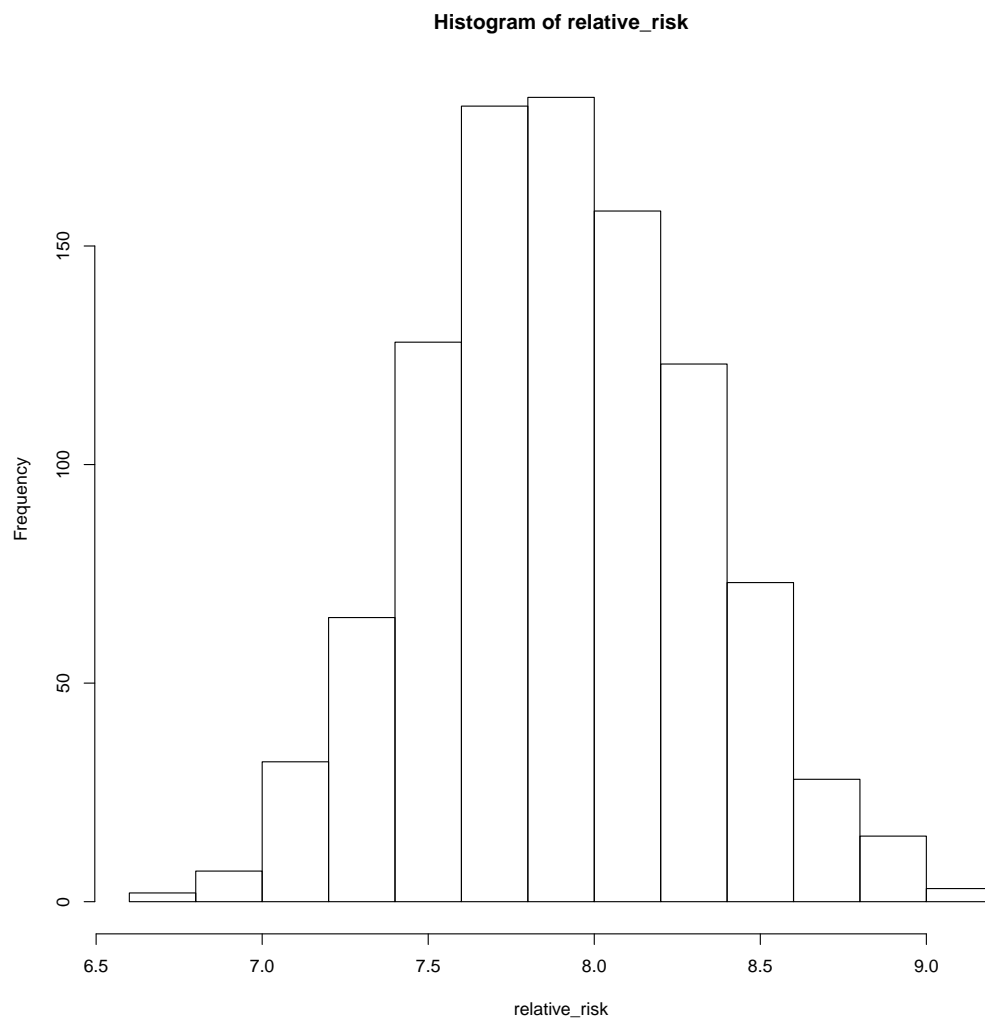
y_S <- 510
n_S <- 412368 + 510

p_S <- rbeta(1000, y_S + 1, n_S - y_S + 1)

#c)
relative_risk <- p_N/p_S

hist(relative_risk)

```



```
relative_risk <- sort(relative_risk)

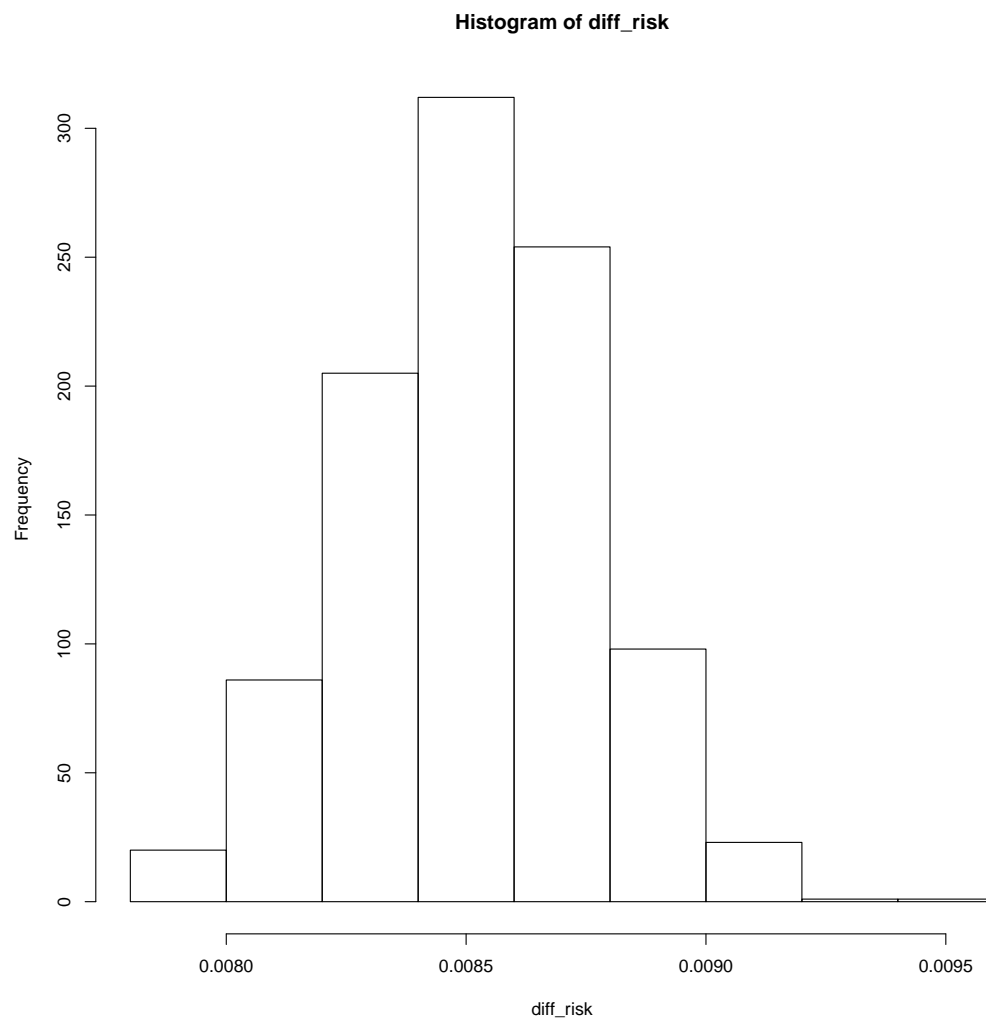
confidence_interval <- relative_risk[26:975]

print(paste0("[", confidence_interval[1], ", ",
             confidence_interval[length(confidence_interval)], "]" ))

## [1] "[7.13534546960749, 8.73023703107285]"

#d)
diff_risk <- p_N - p_S

hist(diff_risk)
```



```
#e)
sum(diff_risk > 0)/length(diff_risk)

## [1] 1
```

```

4. #a)
weights <- c(10, 11, 12, 11, 9)

S <- var(weights)

n <- length(weights)

sigma2 <- S/rchisq(1000, n-1)

mu <- rnorm(1000, mean = mean(weights), sd = sqrt(sigma2)/sqrt(n))

mycontour(normchi2post, c(4, 20, 0, 10), weights,
           xlab="mean", ylab="variance")

## Error in mycontour(normchi2post, c(4, 20, 0, 10), weights, xlab = "mean", :
## argument "size_grid" is missing, with no default

points(mu, sigma2)

## Error in plot.xy(xy.coords(x, y), type = type, ...): plot.new has not been
## called yet

```

Using the approach from Section 6.7 in *Bayesian Computation with R*, we can treat the data as binned data. In that case, the likelihood function is given by,

$$\begin{aligned}
 L(\mu, \sigma) \propto & \Phi(8.5, \mu, \sigma)^0 (\Phi(9.5, \mu, \sigma) - \Phi(8.5, \mu, \sigma))^1 \\
 & X(\Phi(10.5, \mu, \sigma) - \Phi(9.5, \mu, \sigma))^1 (\Phi(11.5, \mu, \sigma) - \Phi(10.5, \mu, \sigma))^2 \\
 & X(\Phi(12.5, \mu, \sigma) - \Phi(11.5, \mu, \sigma))^1 (1 - \Phi(12.5, \mu, \sigma))^0
 \end{aligned}$$

where $\Phi(; \mu, \sigma)$ is the cdf of a normal (μ, σ) random variable.

Using the standard noninformative prior and transforming the standard deviation by $\lambda = \log(\sigma)$ gives us the posterior density,

$$g(\mu, \lambda | \text{data}) \propto L(\mu, \exp(\lambda))$$

```

#b) & c)
d <- list(int.lo=c(0, seq(8.5,12.5,by=1)),
          int.hi=c(seq(8.5,12.5,by=1), Inf),
          f=c(0, 1, 1, 2, 1, 0))

start <- c(ceiling(mean(weights)), ceiling(log(sd(weights))))

fit <- laplace(groupeddatapost,start,d)

modal.sds <- sqrt(diag(fit$var))

proposal <- list(var=fit$var, scale=2)

fit2 <- rwmetrop(groupeddatapost, proposal, start, 10000, d)

post.means <- apply(fit2$par,2,mean)
post.sds <- apply(fit2$par, 2 ,sd)

cbind(c(fit$mode), modal.sds)

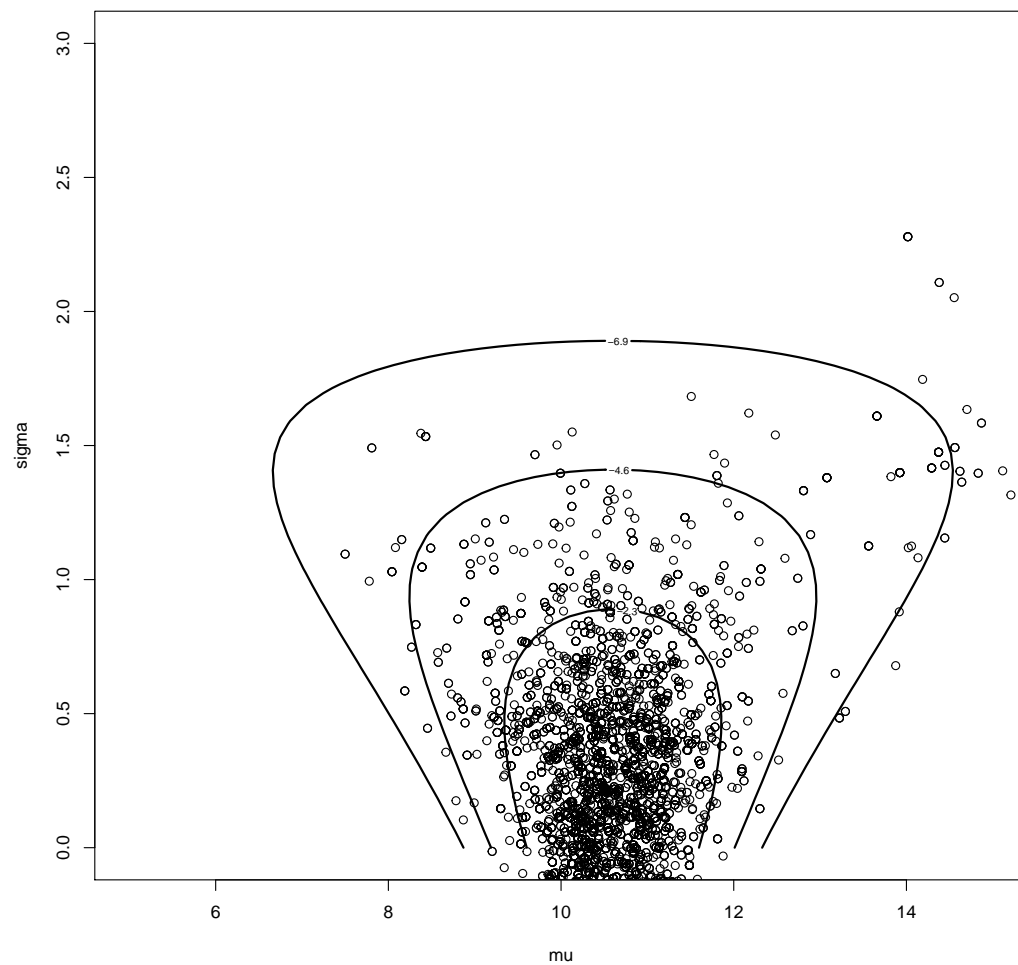
##                modal.sds
## [1,] 10.59952000 0.4553677
## [2,] -0.02373227 0.3445477

cbind(post.means, post.sds)

##      post.means post.sds
## [1,] 10.6420378 0.7637662
## [2,]  0.2328623 0.4289336

LearnBayes::mycontour(groupeddatapost, c(5, 15, 0.00001, 3), d,
                      xlab="mu", ylab="sigma")
points(fit2$par[5001:10000,1], fit2$par[5001:10000,2])

```



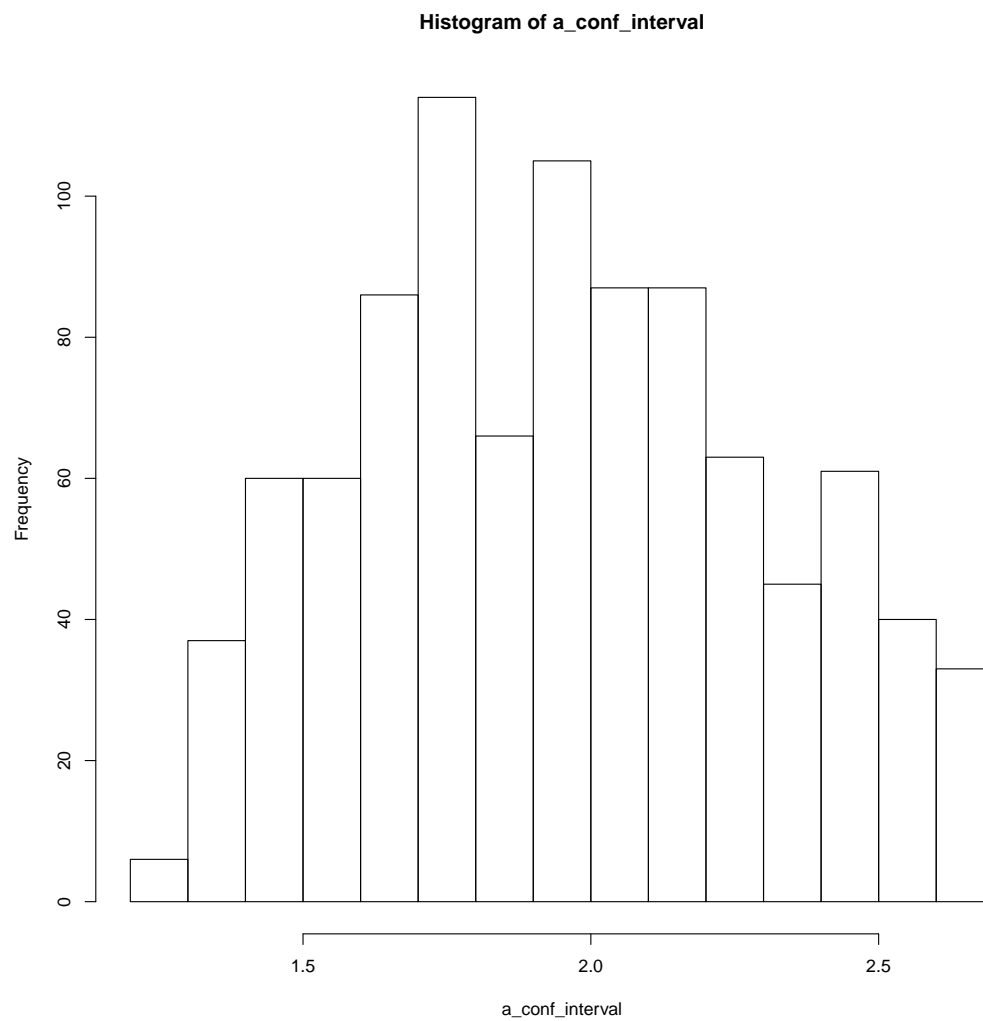
```
#d)
post.means

## [1] 10.6420378  0.2328623

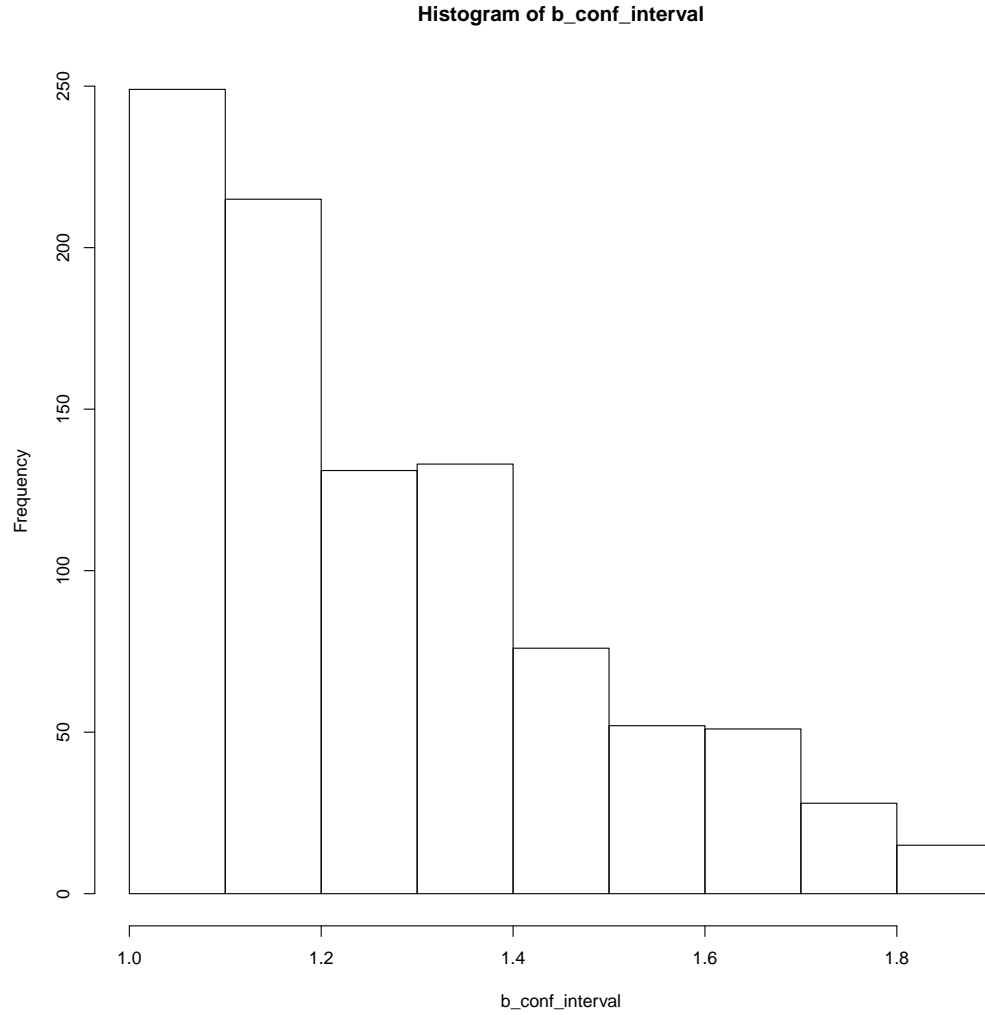
c(mean(mu), sd(mu))

## [1] 10.6090530  0.4018979
```

Both means are quite close together. The standard deviation derived from the correct posterior distribution is smaller than that from the incorrect distribution.



```
#95% confidence interval for b  
b_conf_interval <- b[26:975]  
  
hist(b_conf_interval)
```



6. (a)

$$\begin{aligned}
 p(\lambda_1, \lambda_2 | y_1, y_2) &= \Gamma(144, 2.4) \text{Poisson}(y_1 | \lambda_1) \Gamma(100, 2.5) \text{Poisson}(y_2 | \lambda_2) \\
 &= \Gamma(144, 2.4) e^{-t\lambda_1} \frac{(t\lambda_1)^{y_1}}{y_1!} \Gamma(100, 2.5) e^{-t\lambda_2} \frac{(t\lambda_2)^{y_2}}{y_2!} \\
 &=
 \end{aligned}$$

#b)

#c)

```

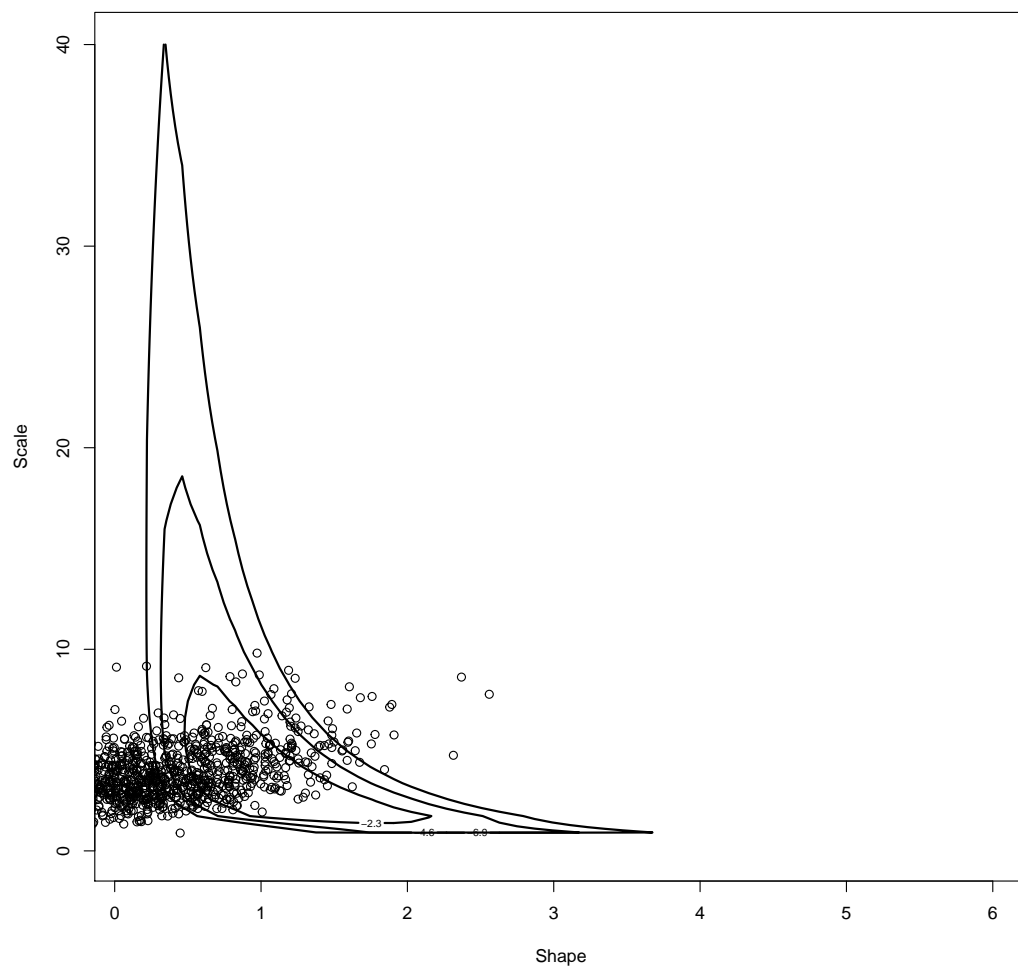
7. #a)
gamma.sampling.post <- function(theta, y){
  return(sum(dgamma(y, shape=theta[1], scale = theta[2], log = TRUE)))
}

data <- c(12.2, 0.9, 0.8, 5.3, 2, 1.2, 1, 0.3, 1.8, 3.1, 2.8)

LearnBayes::mycontour(gamma.sampling.post, c(0.1, 6, 0.1, 40), data,
  xlab="Shape", ylab="Scale")

s <- simcontour(gamma.sampling.post, c(0.1, 6, 0.1, 40), data, 1000)
## Error in sample.int(x, size, replace, prob): NA in probability vector
points(s)

```



```

mu_vals <- sort(s$y*s$x)

s_conf_interval <- mu_vals[26:975]

print(paste0("[", s_conf_interval[1], ", ",
              s_conf_interval[length(s_conf_interval)], "]" ))

## [1] "[-2.51881911941972, 8.5900915930946]"

#b)
gamma.sampling.post.rate <- function(theta, y){
  return(sum(dgamma(y, shape=theta[1], rate = theta[2], log = TRUE)))
}

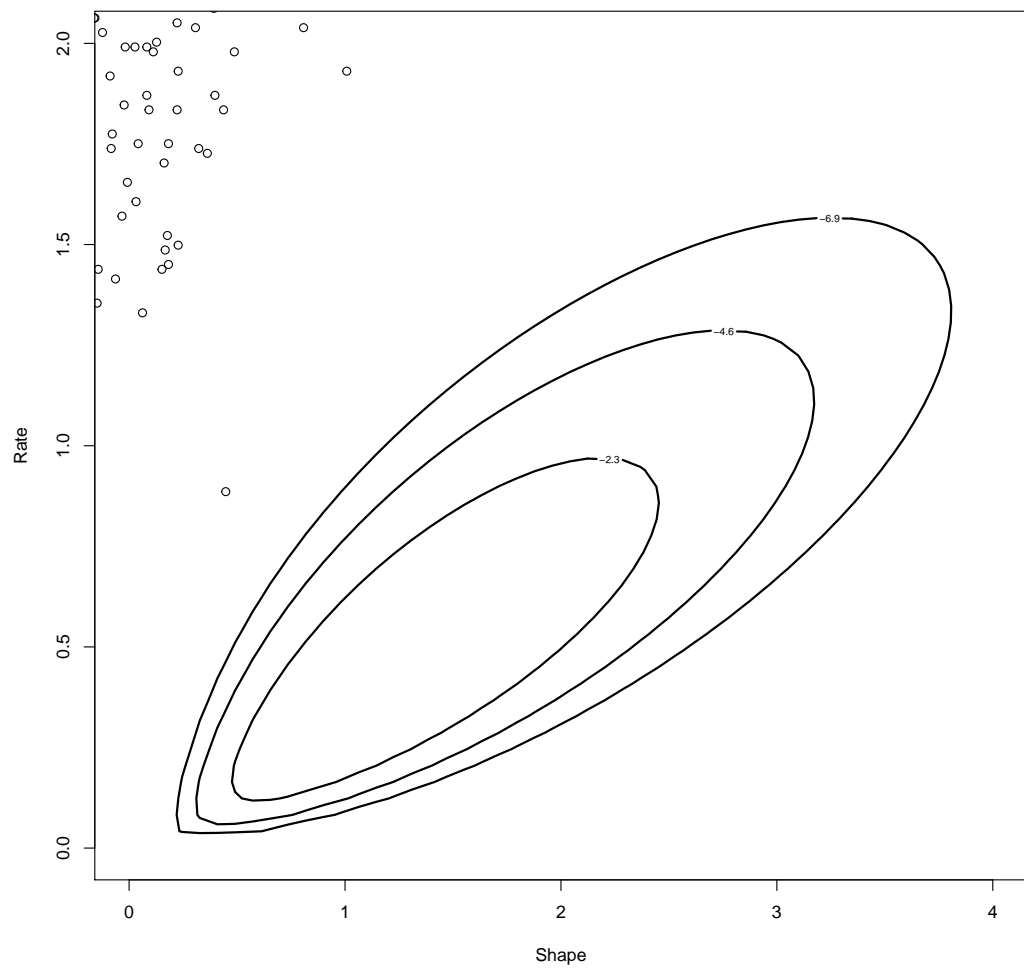
LearnBayes::mycontour(gamma.sampling.post.rate, c(0.001, 4, 0.001, 2), data,
                      xlab="Shape", ylab="Rate")

s <- simcontour(gamma.sampling.post.rate, c(0.001, 4, 0.001, 2), data, 1000)

## Error in sample.int(x, size, replace, prob): NA in probability vector

points(s)

```



```

mu_vals <- sort(s$x*(1/s$y))

s_conf_interval <- mu_vals[26:975]

print(paste0("[", s_conf_interval[1], ", ",
              s_conf_interval[length(s_conf_interval)], "]" ))

## [1] "[-0.249830278343517, 0.350148367952522]"

#c)
gamma.sampling.post.mean <- function(theta, y){
  lambda <- theta[2]/theta[1]
  return(sum(dgamma(y, shape=theta[1], scale = lambda, log = TRUE)))
}

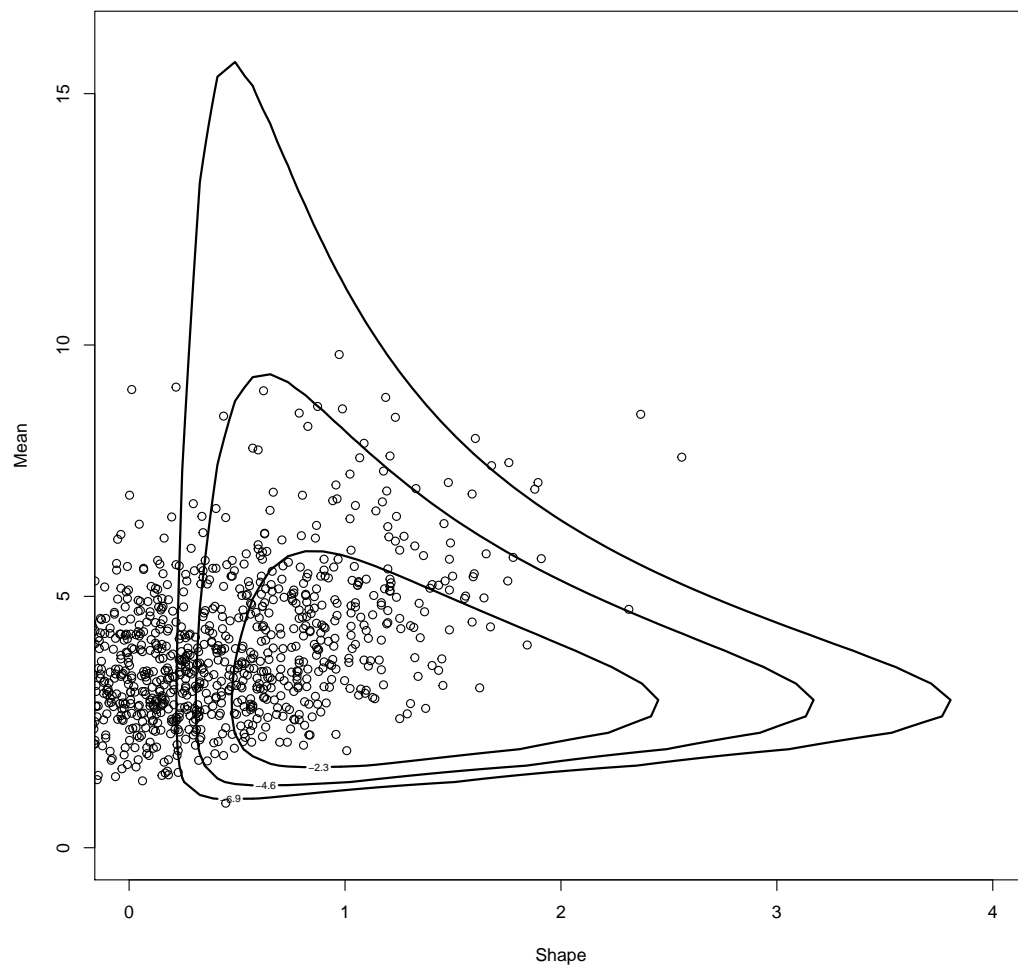
LearnBayes::mycontour(gamma.sampling.post.mean, c(0.001, 4, 0.001, 16), data,
                      xlab="Shape", ylab="Mean")

s <- simcontour(gamma.sampling.post.mean, c(0.001, 4, 0.001, 16), data, 1000)

## Error in sample.int(x, size, replace, prob): NA in probability vector

points(s)

```



```
mu_vals <- sort((s$y/s$x))

s_conf_interval <- mu_vals[26:975]

print(paste0("[", s_conf_interval[1], ", ",
             s_conf_interval[length(s_conf_interval)], "]" ))

## [1] "[-111.710526315789, 94.0961538461536]"
```

I believe the third computational method is the best for computing the 95% interval estimate for μ , as it directly includes μ in the calculations.

```

8. #a)
quantile1 <- list(p=0.25, x=0.15)
quantile2 <- list(p=0.75, x=0.35)

beta1 <- beta.select(quantile1, quantile2)

print(beta1)

## NULL

quantile1 <- list(p=0.25, x=0.75)
quantile2 <- list(p=0.75, x=0.95)

beta2 <- beta.select(quantile1, quantile2)

print(beta2)

## NULL

#b)
data <- cbind(c(16, 18, 20, 22, 24, 26, 28),
              c(0, 0, 6, 12, 7, 9, 3),
              c(2, 7, 14, 26, 13, 14, 3))
#prior <- c(beta1, beta2)
#data.new <- rbind(data, prior)
mycontour(logisticpost, c(-3, 3, -1, 9), data, xlab="beta0",
           ylab = "beta1")

## Error in mycontour(logisticpost, c(-3, 3, -1, 9), data, xlab = "beta0", : argument
"size_grid" is missing, with no default

#c)

#d)

```

4. **CH4, Q8 extra:** fit model using a frequentist logistic regression model; figure out how to make the equivalent confidence interval for part (d). (Could use bootstrap, probably BC_a and its approximation ABC, try package *bootBCa*; include a histogram of your BC_a bootstrap resampled probability values and normal quantile plot of those values. Let number of bootstrap samples be $B = 9,999$.)

Solution:

```

act <- c(rep(16, times = 2), rep(18, times = 7),
        rep(20, times = 14), rep(22, times = 26),
        rep(24, times = 13), rep(26, times = 14),
        rep(28, times = 3))

success <- c(rep(0, times = 9), rep(1, times = 6), rep(0, times = 14-6),
            rep(1, times = 12), rep(0, times = 26-12),
            rep(1, times = 7), rep(0, times = 13-7),
            rep(1, times = 9), rep(0, times = 14-9),
            rep(1, times = 3))

data <- data.frame(act = act, success = success)

model <- glm(success ~ act, family="binomial", data = data)

summary(model)

##
## Call:
## glm(formula = success ~ act, family = "binomial", data = data)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.6008  -1.0638  -0.6339   1.0363   1.5701
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -7.03444    2.23042  -3.154  0.00161 **
## act          0.30732    0.09839   3.123  0.00179 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 109.201  on 78  degrees of freedom
## Residual deviance:  97.413  on 77  degrees of freedom
## AIC: 101.41
##
## Number of Fisher Scoring iterations: 4

```

```

new.data <- data.frame(act = c(20))
predict(model, newdata = new.data, type="response")

##          1
## 0.2915336

#Using the endpoint transformation method detailed here:
#https://stats.stackexchange.com/questions/354098/calculating-confidence-intervals-for-a-logistic-reg

beta <- c(model$coefficients[[1]], model$coefficients[[2]])
x <- c(1, 20)

standard_error <- predict(model, newdata = new.data,
                           type="response", se.fit=TRUE)$se.fit[[1]]

response <- x%*%beta

lower_interval <- exp(response - 1.645*standard_error)/
  (1+exp(response - 1.645*standard_error))
upper_interval <- exp(response + 1.645*standard_error)/
  (1+exp(response + 1.645*standard_error))

print(paste0("[", lower_interval, ", ", upper_interval, "]"))

## [1] "[0.267708775106425, 0.316562129898188]"

```