Bayesian Analysis - Final Project

Chris Hayduk

May 10, 2019

1 Implementation, Details, & Function Arguments

This project aims to re-implement the `mycontour()` and `simcontour()` functions from the `LearnBayes` package. This package is associated with the book *Bayesian Computation with R* by Jim Albert.

The `mycontour()` function produces a contour graph for a general two parameter density. The original `mycontour()` had hard-coded values for the level curves in the contour plot, a fixed size for the grid to compute posterior values, only worked with log posterior functions, and computed the max of the log posterior on a grid rather than through an optimization method. My re-implementation of the function seeks to remedy each of these issues.

Arguments:

- **fn** - posterior density function
- **limits** - vector of x and y limit values. eg. limits = c(x_lo, x_hi, y_lo, y_hi)
- **data** - data used to compute the posterior
- **size_grid** - size of the grid that the posterior will be computed on
- **levels** - vector of values that the contours will be drawn at. Each value in the vector represents the desired percent of the mode.
- **log_func** - TRUE if the input function is already a log posterior. FALSE by default
- ... - further arguments to be passed to `contour()`

```
mycontour <- function (fn, limits, data, size_grid,
                       levels, log_func = FALSE, ...)
{
    #This function facilities inputting matrices of parameters
    #into the posterior density function
```

```
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

#Create x, y grid using size specified by user
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

#Compute log posterior on grid
Z = LOGF(cbind(X[1:n2], Y[1:n2]), data)

#Convert Z back from log posterior
Z <- exp(Z)

#Generate start values for optimization function
x_start <- mean(c(limits[1], limits[2]))
y_start <- mean(c(limits[1], limits[2]))

#Tells the optimization function that we want to maximize
#the log posterior
control <- list(fnscale = -1)
```

```
    #Find max of log posterior
    max_z <- optim(par=c(x_start, y_start), fn= LOGF,
                   gr = NULL,
                   data = data,
                   method = "L-BFGS-B",
                   lower = c(limits[1], limits[3]),
                   upper = c(limits[2], limits[4]),
                   control = control)

    #Convert max back from log posterior
    max_z <- exp(max_z$value)

    Z = matrix(Z, c(ng, ng))

    #Create vector to store contour values
    contours <- rep(0, times=length(levels))

    #Get contour values (given by percent of max Z value)
    for(i in 1:length(levels)){
      contours[i] <- levels[i]*max_z
    }

    #Create contour plot
    contour(x0, y0, Z, levels = contours, lwd = 2,
        ...)
}
```

The `simcontour()` function simulates a random sample for a general two-parameter density defined on a grid. The original `simcontour()` had a fixed size for the grid to compute posterior values and only worked with log posterior functions. My re-implementation of the function seeks to remedy each of these issues.

Arguments:

- **fn** - posterior density function
- **limits** - vector of x and y limit values. eg. limits = c(x_lo, x_hi, y_lo, y_hi)
- **data** - data used to compute the posterior
- **size_grid** - size of the grid that the posterior will be computed on
- **m** - size of the random sample you would like to generate
- **log_func** - TRUE if the input function is already a log posterior. FALSE by default

```
simcontour <- function(fn, limits, data,
                       size_grid, m, log_func = FALSE)
{
    #This function facilities inputting matrices of parameters
    #into the posterior density function
    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
```

```
    #Create grid of points
    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

    #Compute log posterior on grid
    Z = LOGF(cbind(X[1:n2], Y[1:n2]), data)
    Z = matrix(Z, c(ng, ng))

    #Create matrix of x, y, and posterior values
    d = cbind(X[1:n2], Y[1:n2], Z[1:n2])

    prob = d[, 3]

    #Convert values back to probabilites from log-probabilities
    prob = exp(prob)

    prob = prob/sum(prob)

    #Sample row indeces based on posterior probability
    i = sample(n2, m, replace = TRUE, prob = prob)

    #Return list of x, y values
    return(list(x = d[i, 1], y = d[i, 2]))
}
```

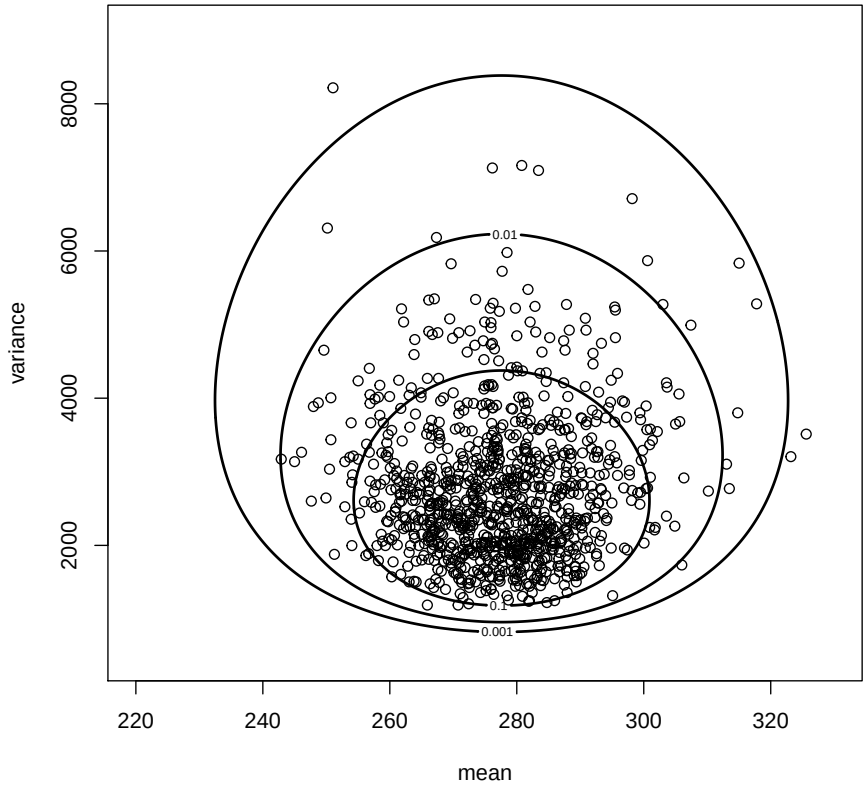
2 Examples

I will now generate some examples from the book *Bayesian Computation with R*. In order to facilitate the first example, I have re-implemented the function `normchi2post()` from the `LearnBayes` pacakge. This version computes the posterior density without taking the log.

```
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))

  return(posterior_density)
}
```

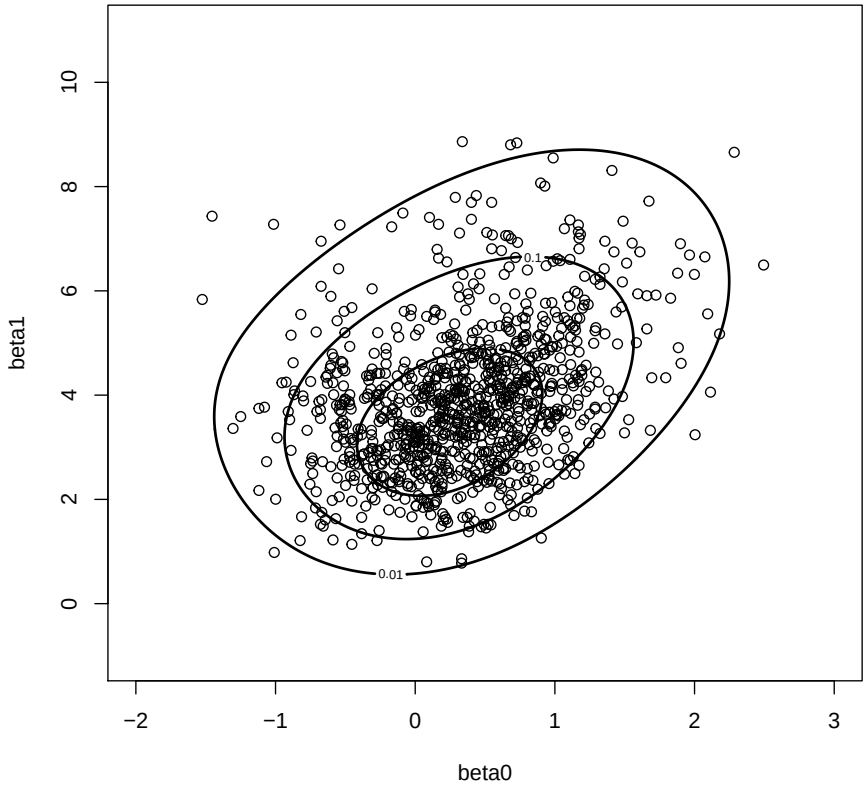
This example is taken from p. 63-65 in *Bayesian Computation with R*. Note the "labels", "xlab", and "ylab" arguments in `mycontour()`. These are all extra arguments that are passed to the contour graph. "Labels" defines the label for each contour level, while "xlab" and "ylab" provide the labels for the x-axis and y-axis respectively. Also note that the argument "log_func" has been omitted because the posterior density has not been log transformed before input.

```
mycontour(normchi2post, limits=c(220, 330, 500, 9000),
               data = time, size_grid = 1000,
               levels = c(0.1, 0.01, 0.001),
               labels = c(0.1, 0.01, 0.001),
               xlab="mean", ylab="variance")

s <- simcontour(normchi2post, limits = c(220, 330, 500, 9000),
           data= time, 1000, 1000)

points(s)
```


variance
8000
6000
4000
2000
0.01
0.1
0.001
220
240
260
280
300
320
mean

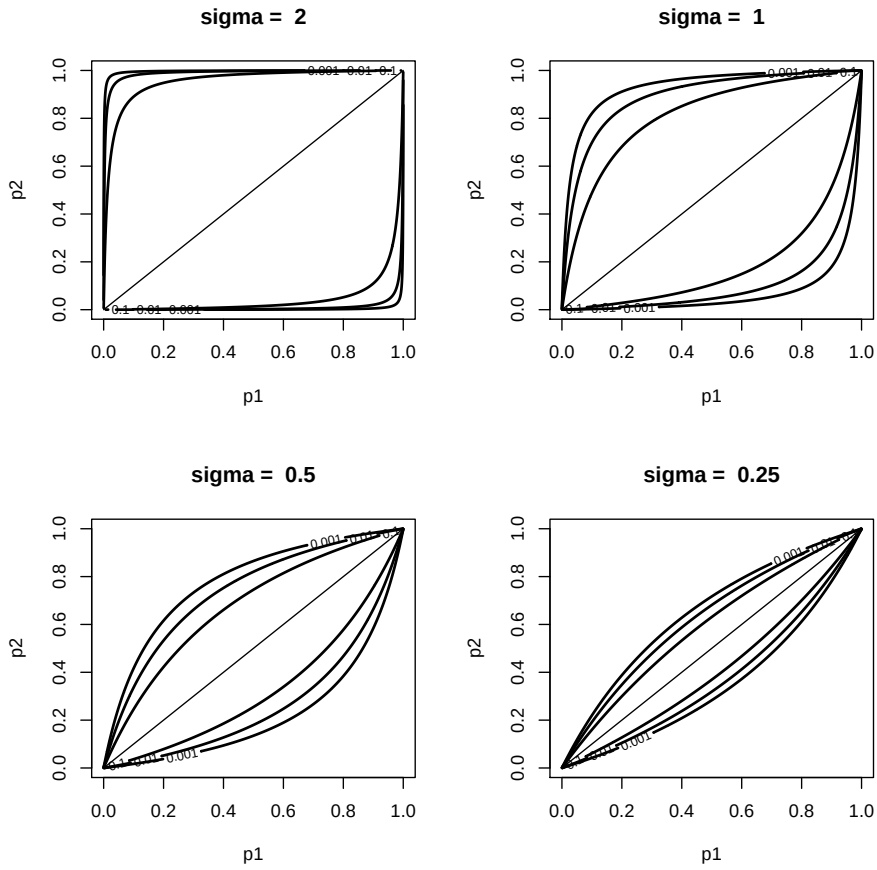
This example is taken from p. 70-74 in *Bayesian Computation with R*. Note that in this case, the posterior density has been log transformed before input, so the argument "log_func" has been included and set to TRUE.

```
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")

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

points(s)
```


beta1
beta0
0.1
0.01
10
8
6
4
2
0
−2
−1
0
1
2
3

This example p.77-78 in *Bayesian Computation with R*. Note the argument "main" in `mycontour()`, which supplies a title for the contour graph.

```
sigma <- c(2, 1, 0.5, 0.25)
plo <- 0.0001
phi <- 0.9999

par(mfrow=c(2,2))
for(i in 1:length(sigma)){
  mycontour(howardprior, c(plo, phi, plo, phi),
          c(1, 1, 1, 1, sigma[i]), 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[i])),
          xlab = "p1", ylab = "p2")
  lines(c(0,1),c(0,1))
}
```


sigma = 2
sigma = 1
sigma = 0.5
sigma = 0.25
p1
p2
0.0
0.2
0.4
0.6
0.8
1.0
0.001
0.01
0.1