Stat 532 Assignment 5

Kenny Flagg

October 9, 2015

1. **Guide to Fitting a Logistic Beta-Binomial Model in R**

   The example R script performed a Bayesian analysis to estimate a distribution of mortality rates for a population. The steps taken were:

   (a) Define the model.
   (b) Find the posterior distribution.
   (c) Find the parameters that could be used for a normal approximation.
   (d) Use rejection sampling to simulate the posterior distribution.
   (e) Make inference from the simulated distribution.

   I will first describe the model and then explain the code step-by step.

   The Model

   The problem analyzed in the example code was to estimate the stomach cancer mortality rate for a certain at-risk population in the largest cities in Missouri. The model used was a Beta-Binomial model. This is a hierarchical model where each city had its own mortality rate, and the rates followed a distribution. The parameter of interest was the mean, or overall average, of the mortality rates across all cities.

   It was assumed that each individual in this population in the jth city had a probability π_j of dying of stomach cancer. The population size n_j was known, so the number of mortalities y_j was assumed to follow a Binomial(n_j, π_j) distribution. π_j is the mortality rate, for which the researchers sought a distribution.

   The mortality rates π_j were modeled using a Beta distribution with an average of η and a precision of K. The parameter η is the overall mortality rate for all cities combined. The precision specifies how tightly the the distribution is concentrated around η, so a large K means that there is little variation in the mortality rates between different cities. K can be thought of as the number of individuals in the population who contribute information to the distribution of the π_j. The Beta distribution is usually parameterized in terms of previous "successes" and "failures" so the model used here was Beta$(K\eta, K(1-\eta))$.

   Since the researchers were not interested in the mortality rates for individual cities, so they used the Beta-Binomial$(n_j, K\eta, K(1-\eta))$ distribution for y_j. This distribution incorporates the uncertainty from the Beta distribution into the data model, avoiding extra complexity from estimating the π_j.

A Bayesian model has three components: The probability model (or likelihood), the observed data, and the prior information. The Beta-Binomial distribution is the probability model. The data are the population sizes and mortality counts that were recorded. The remaining piece in the prior information. A prior distribution is needed to describe η and K. These researchers used a non-informative joint prior with density

$$p(\eta, K) \propto \frac{1}{\eta(1-\eta)(1+K)^2}.$$

The Code

(a) Define the model.

The first piece of code defines a function to compute the natural logarithm of the likelihood, or probability model for the data. The logarithmic scale compresses values across different orders of magnitude, so it is used to simplify computation.

The arguments to the function are a vector containing values for η and K, and vectors containing the population sizes n_j and mortality counts y_j.

The Beta-Binomial probability mass function can be found in many textbooks on statistical models. The likelihood is the product of multiplying Beta-Binomial probabilities for each observed (n_j, y_j) pair. On the logarithmic scale, multiplication becomes addition and division becomes subtraction, so this function adds and subtracts the terms. The `lchoose` and `lbeta` functions operate on each element of the vectors and return vector values; the `sum` function adds these up to compute the log-likelihood.

```
BB.loglik.fun <- function(etaK.vec, n.vec, y.vec){
  ll.out <- sum(lchoose(n.vec, y.vec) +
                lbeta((etaK.vec[2]*etaK.vec[1] + y.vec),(etaK.vec[2]*(1-etaK.vec[1])+n.vec-y.vec))-
                lbeta(etaK.vec[2]*etaK.vec[1], etaK.vec[2]*(1-etaK.vec[1])))
  return(ll.out)
}
```

The next section creates the data vectors and defines the prior density function.

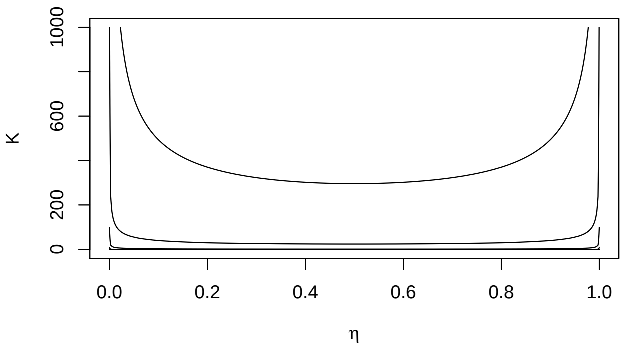
```
#### Applied problem  (Tsutakawa et al. 1985) - Estimate rate of death from
## stomach cancer for at risk males between ages 45-64 for the largest cities in
## Missouri.  Here are the mortality rates for 20 of the cities
##  (number at risk (nj), number of cancer deaths (yj))

nj.vec <- c(1083,855,3461,657,1208,1025, 527, 1668, 583, 582, 917, 857, 680, 917, 53637,
            874, 395, 581, 588, 383)
yj.vec <- c(0,0,2,0,1,1,0,2,1,3,0,1,1,1,54,0,0,1,3,0)

## We need to assign priors for eta and K - let's use the vague prior
##  porportional to  g(eta,K) propto (1/(eta*(1-eta)))*(1/(1+K)^2)

BB.prior.fun <- function(etaK.vec){
  (1/(etaK.vec[1]*(1-etaK.vec[1])))*(1/((1+etaK.vec[2])^2))
}
```

The researchers neglected to plot the prior, so I do that in the contour plot on the next page. This is an improper prior distribution that places infinite mass along the boundary values of $\eta = 0$, $\eta = 1$, and $K = -1$. The effect of this prior on the model was to give the parameters very large variances, so the posterior distribution would be dominated by the data.



(b) Find the posterior distribution.

The program initially finds the distribution of η and K on their original scales. Since their values are bounded, it is useful to transform them before doing further computation, so the program then computes the posterior distribution for an unbounded transformation. After the transformation, it will be relatively simple to characterize the distribution with a simulation.

As a function of η and K, the posterior distribution is proportional to the likelihood times the the prior. In the example, the researchers did the computation on the logarithmic scale, so the log-likelihood and the logarithm of the prior are added.

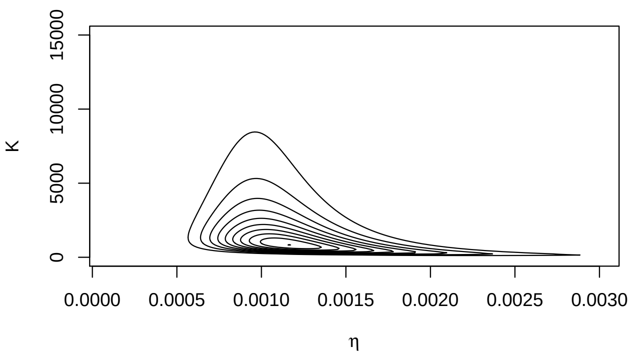
The `ll` piece repeats the code from the log likelihood function. The line with `lprior` computes the logarithm of the prior.

```r
## We could combine them into one function to find the posterior

BB.logpost1<- function(etaK.vec, n.vec, y.vec){
  ll <- sum(lbeta(etaK.vec[2]*etaK.vec[1] + y.vec, etaK.vec[2]*(1-etaK.vec[1])+n.vec-y.vec)-
              lbeta(etaK.vec[2]*etaK.vec[1], etaK.vec[2]*(1-etaK.vec[1])))
  lprior <- -log(etaK.vec[1]) - log(1-etaK.vec[1]) - 2*log(1+etaK.vec[2])
  lpost.out <- ll + lprior
  return(lpost.out)
}

eta.vec <- seq(0.0001,0.003,length=200)
K.vec <- seq(1,15000, length=200)
etaK.grid <- expand.grid(eta.vec, K.vec)
lp.grid <- apply(etaK.grid, 1, BB.logpost1, n.vec=nj.vec, y.vec=yj.vec)
lp.mat <- matrix(lp.grid, nrow=200, ncol=200)
p.mat <- exp(lp.mat - max(lp.mat))
contour(eta.vec, K.vec, p.mat, ylab="K", xlab=expression(eta))
```

The block of code at the end creates a grid of η and K values, computes the log-posterior density, and creates a contour plot.



It is possible to get a good picture of the distribution, such as the contour plot, but it may be difficult to actually find a mathematical expression for the distribution function to evaluate its properties. The researchers used simulation instead.
The next section of code computes the transformations

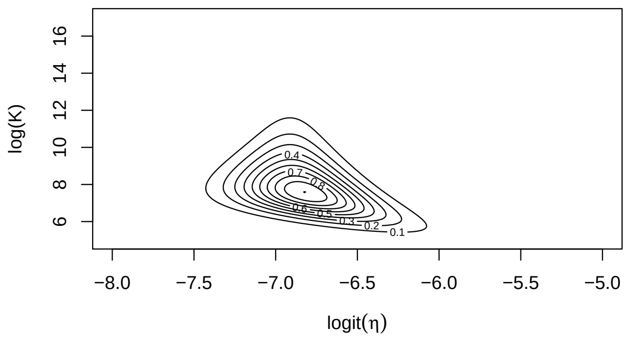
$$\theta_1 = \text{logit}(\eta) = \log\left(\frac{\eta}{1-\eta}\right), \quad \theta_2 = \log(K)$$

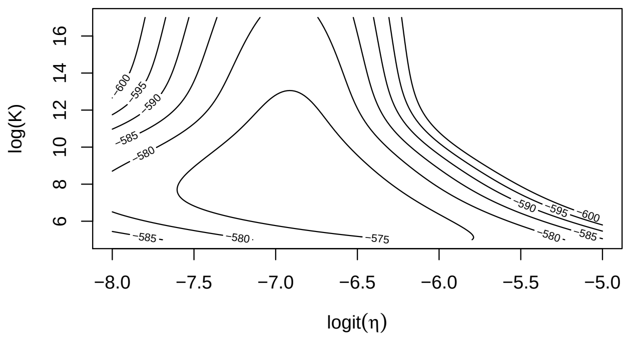
and then finds the posterior distribution of θ_1 and θ_2.
From the commented-out lines of code, it looks like they originally intended to have R do all of the mathematical steps of the transformation, but then they decided to do the derivation by hand to simplify the computation.

```
BB.logpost <- function(theta.vec, n.vec, y.vec){
  eta <- exp(theta.vec[1])/(1+exp(theta.vec[1]))
  K <- exp(theta.vec[2])
  ll <- sum(lbeta(K*eta + y.vec, K*(1-eta) + n.vec - y.vec) -
            lbeta(K*eta, K*(1-eta)))
  #log.prior <- -log(eta) - log(1-eta) - 2*log(1+K)
  #log.Jacob <- (theta.vec[1]+theta.vec[2]) - 2*log(1+exp(theta.vec[1]))
  log.rest <- theta.vec[2] - 2*log(1+exp(theta.vec[2]))
  #trans.log.post <- ll + log.prior + log.Jacob
  trans.log.post <- ll + log.rest
  return(trans.log.post)
}

theta1.vec <- seq(-8,-5,length=200)
theta2.vec <- seq(5,17, length=200)
theta.grid <- expand.grid(theta1.vec, theta2.vec)
logpost.grid <- apply(theta.grid, 1, BB.logpost, n.vec=nj.vec, y.vec=yj.vec)
logpost.mat <- matrix(logpost.grid, nrow=200, ncol=200)
post.mat <- exp(logpost.mat - max(logpost.mat))
contour(theta1.vec, theta2.vec, post.mat, ylab="log(K)",
xlab=expression(logit(eta)))
contour(theta1.vec, theta2.vec, logpost.mat, ylab="log(K)",
        xlab=expression(logit(eta)), levels=seq(-600,-500,by=5))  #not as nice
```

The last part of the above code creates a grid of θ_1 and θ_2 values and then outputs contour plots of the posterior distribution.





The first contour plot is on the logarithmic scale. The second shows the probability density on its usual scale, but makes it more difficult to identify the location with the maximum density.

(c) Find the parameters that could be used for a normal approximation.

One way to make inference from this posterior is to approximate it with a bivariate Normal distribution. This would be done by finding its variance-covariance matrix and the values of θ_1 and θ_2 that give the maximum probability density. The Normal approximation uses this (θ_1, θ_2) pair as its mean and the estimated posterior variance-covariance matrix as its variance-covariance matrix. The code includes an example of finding these numerically with the `optim` function.

The `optim` function can use any of several built-in optimizing algorithms. The details of these are not important, but Nelder-Mead is commonly used because it is robust when optimizing functions that present mathematical difficulties.

The `c(-7,6)` argument sets initial values of (θ_1, θ_2). By default, the function will find a minimum. The `fnscale` specifies a scaling factor; choosing a negative value causes the algorithm to find a maximum. The `hessian=TRUE` argument specifies that the output will include the Hessian, or second derivative matrix evaluated at the maximum. The Hessian is used to find the variance-covariance matrix.

```
optim.out <- optim(c(-7,6), BB.logpost, n.vec=nj.vec, y.vec=yj.vec,
                   control=list(fnscale=-100), method="Nelder-Mead", hessian=TRUE)

optim.out
optim.out$par
Var <- solve(-optim.out$hessian)
Var
```

The desired components of the output are `par`, the parameter values, and `hessian`, the Hessian matrix.

Outputting `par` gives the estimated (θ_1, θ_2):

```
## [1] -6.8197932  7.5761106
```

From the mathematical theory, the variance-covariance matrix is the inverse of the negative Hessian. This is computed by `solve(-optim.out$hessian)` and the result is:

```
##              [,1]        [,2]
## [1,]  0.078965679 -0.14850874
## [2,] -0.148508744  1.34832083
```

Approximate inferences about logit(η) and $\log(K)$ could be made using a

$$\text{MVN}\left(\begin{pmatrix} -6.8197932 \\ 7.5761106 \end{pmatrix}, \begin{pmatrix} 0.078965679 & -0.14850874 \\ -0.148508744 & 1.34832083 \end{pmatrix}\right)$$

distribution. These researchers did not pursue the approximation any further.

(d) Use rejection sampling to simulate the posterior distribution.

The researchers based their analysis on simulated draws from the posterior. These draws can be transformed back to the original scale of η and K, but it is difficult to generate draws without explicitly knowing what the posterior distribution is. They used the method of rejection sampling, where draws are taken from another distribution and then randomly kept or discarded. This other distribution is called the proposal distribution and it must be possible to scale the proposal distribution function so that its value is at least as large as the value of the posterior distribution function for each pair of θ_1 and θ_2 values. The process is:

i. Draw (θ_1, θ_2) pairs from the proposal distribution.
ii. For each draw, compute the posterior density and the proposal density for that (θ_1, θ_2) pair.
iii. For each draw, compute the ratio of the posterior density to the proposal density.
iv. Randomly keep or discard each draw using the draw's ratio as the probability that it is kept.

They chose a bivariate t-distribution as the proposal distribution. The next piece of code loads a library with the bivariate t functions, displays the help file for the library, and defines a function to compute the ratio. Once again, the ratio is calculated on the logarithmic scale, so the division becomes subtraction of logarithms.

```r
library(LearnBayes)
help(dmt)

post.prop.diff <- function(theta, n.vec, y.vec, t.params){
  post.part <- BB.logpost(theta, n.vec=n.vec, y.vec=y.vec)
  proposal.part <- dmt(theta, mean=t.params$m, S=t.params$var,
                       df=t.params$df, log=TRUE)
  d <- post.part - proposal.part
  return(d)
}
```

A scaling factor is needed to ensure that the proposal density is at least as large the posterior density. Without it, the ratio could exceed 1 and then cannot be used as a probability. They used `optim` to find the maximum of the ratio on the log scale.

It appears that they tried several starting values and found two possible maxima. They checked both of them manually and used the one that was slightly larger. Keeping a record of this in the comments is very good practice.

```r
t.params.set <- list(m=c(-6.8, 7.6), var=2*Var, df=4)

d.out <- optim(c(-7, 7), post.prop.diff, n.vec=nj.vec, y.vec=yj.vec,
               t.params=t.params.set, control=list(fnscale=-10))

d.out$par   #-6.8899, 12.46 So, max value of d occurs at (-6.88, 12.46)
            # -6.886769  7.507773  (for some starting values)

post.prop.diff(c(-5, 7), n.vec=nj.vec, y.vec=yj.vec, t.params=t.params.set)
d.max <- post.prop.diff(c(-6.8899, 12.46), n.vec=nj.vec, y.vec=yj.vec,
                        t.params=t.params.set) #-569.3335
dmax <- post.prop.diff(c(-6.8868, 7.5077),n.vec=nj.vec, y.vec=yj.vec,
                       t.params=t.params.set) #-570.07
```

Now, they did the four steps of rejection sampling. First, they drew 10,000 random pairs of values from the bivariate t-distribution.

```r
#1. Draw 1000 draws from the proposal distribution (a multivariate-t)
n.draws <- 10000
prop.draws <- rmt(n.draws, mean=t.params.set$m, S=t.params.set$var,
                  df=t.params.set$df)
```

Next, they computed the posterior and proposal densities.

```r
#2. Evaluate the posterior and the proposal distribution at all the values
log.post <- apply(prop.draws, 1, BB.logpost, n.vec=nj.vec, y.vec=yj.vec)
log.g.theta <- dmt(prop.draws, mean=t.params.set$m, S=t.params.set$var,
                   df=t.params.set$df, log=TRUE)
```

Third, they computed the ratios to use as acceptance probabilities. The scaling factor was found on the logarithmic scale, so it is subtracted from the logarithms. Exponentiating the logarithms converts them into a probability between 0 and 1.

```r
#3. Calculate the ratio
accept.probs <- exp(log.post - log.g.theta - d.max)
```

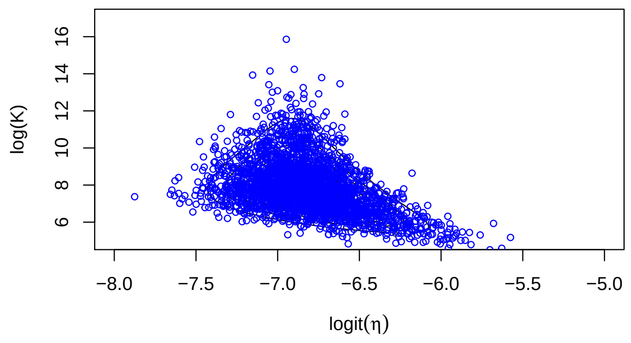
Last, they randomly choose which draws to accept. `runif(n.draws)` generates random numbers between 0 and 1. The `n.draws` argument tells the function to generate the same number of draws as were drawn from the t-distribution. The draws are kept where the random number is less than the ratio.

```
#4. Pick off the draws that we accept
theta.draws <- prop.draws[runif(n.draws) <= accept.probs,]

#What percent did we accept?
length(theta.draws[,1])/n.draws #only 28% inefficient, but easy to find?

contour(theta1.vec, theta2.vec, post.mat, ylab="log(K)",
        xlab=expression(logit(eta)))
points(theta.draws[,1], theta.draws[,2], pch=1, col=4, cex=0.75)
```

The graph overlays the contour plot with the draws that were kept. This is a quick way to check that the simulated draws actually follow the intended distribution.

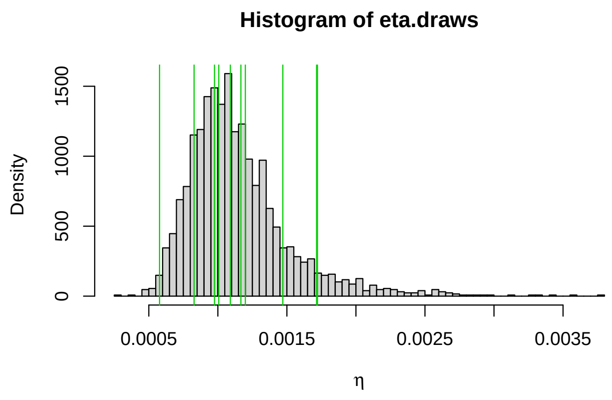


(e) Make inference from the simulated distribution.

The final section of R code transforms the θ_1 and θ_2 draws back to the scale of η and K, and then outputs some summary information about the posterior distribution of η.

```
##Inference about the mean, eta?
eta.draws <- exp(theta.draws[,1])/(1+exp(theta.draws[,1]))
hist(eta.draws, nclass=100, col="lightgray", freq=FALSE,
     xlab=expression(eta))
median(eta.draws) #0.001081160
abline(v=yj.vec/nj.vec, col=3) # Add the observed proportions
quantile(eta.draws, c(.025, .975)) #0.0006410274 0.0020984802
                                   #may need more draws to better capture the right-hand
                                   #  tail of the distn
```

The histogram is skewed, so the median is a more reliable measure of the center than the average. The code also generates a 95% posterior interval; the interpretation of the interval is that there is a 95% probability that a stomach cancer mortality rate for the population of interest in any large Missouri city is between the reported values.



2. Most of my thoughts regard the comments about assumptions and priors near the beginning of the script. I find it hard to believe that animals would ever truly have an "equal probabilities of death" although lab animals with identical genes should come close. I suppose that more realistic versions of this situation are why overdispersed models exist.

   I also take issue with the comment that the flat prior has a "serious flaw." In that case the group receiving each dose will have a separate posterior distribution, $\theta_i \sim \text{Beta}(y_i + 1, n_i - y_i + 1)$. This uses the information that the groups had different doses by treating the groups separately; it's only flaw is that it does not treat the dose as a continuous predictor. This is not a problem if the researchers are not interested in the structure of the relationship between dose and death, but is inapplicable in this case because the linear relationship must be estimated in order to make inference about the LD50.

   On the subject of the R script, it was nice to compare plots of draws from the discrete grid with plots of the jittered draws. The grid was fine enough that I couldn't see a difference, so I don't foresee myself actually using the jittering method!

   The most interesting thing I saw was the technique of computing everything on the log scale and subtracting off the maximum. Since exponentiating these values will result in values in the interval $(0, 1)$, I had an immediate worry that unnormalized densities could be mistaken for probabilities. However, this really isn't any more of a problem than it is with unlogged densities. The person writing the code will know that the values need to be normalized anyway.

   I only have one big question. In practice, is it really necessary to subtract away the maximum? Taking logarithms will put most values on a reasonable scale, but there is still a range of values. If logged values are extreme enough to cause overflow/underflow problems, I would think shifting them would just turn the safe numbers into hard-to-deal-with extreme numbers.

R Code Appendix

My code for the prior contour plot in problem 1:

```r
eta.vec <- c(seq(0.0001, 0.1999, length = 80),
             seq(0.2, 0.8, length = 40),
             seq(0.8001, 0.9999, length = 80))
K.vec <- c(seq(-0.9999, 9.9999, length = 100), seq(10, 1000, length = 100))
etaK.grid <- expand.grid(eta.vec, K.vec)
prior.grid <- apply(etaK.grid, 1, BB.prior.fun)
lprior.mat <- matrix(log(prior.grid), nrow=200, ncol=200) # Use log to compress the range
contour(eta.vec, K.vec, lprior.mat, ylab="K", xlab=expression(eta), drawlabels = FALSE)
```