Computation and Monte Carlo Techniques

Up to now we have seen conjugate Bayesian analysis:

$$posterior \propto prior \times likelihood$$

$$beta(a+x, b+n-x) \propto beta(a,b) \times binomial(x;n)$$

$$\theta^{a+x-1}(1-\theta)^{b+n-x-1} \propto \theta^{a-1}(1-\theta)^{b-1} \times \theta^{x}(1-\theta)^{n-x}$$

For example:

$beta(10,4) \propto beta(2,2) \times binomial(x=8;n=10)$

Conjugate analyses can often be used for simple problems, but almost all problems with more than two or three parameters no conjugate pairing exists, and finding posterior distributions analytically can become very difficult:

Let's see that even in still simple problems, this can be the case:

Example: Let $\epsilon = \theta_1 - \theta_2$, where θ_1 and θ_2 are two independent binomial parameters.

Suppose in a clinical trial (TIMI [thrombolysis in myocardial infarction], NEJM (1985; 923-936)) we observe

$x_1 = 78$, $n_1 = 118$, $\hat{\theta}_1 = 0.661$, and

$x_2 = 44$, $n_2 = 122$, $\hat{\theta}_2 = 0.361$

Question: What is the posterior distribution of ϵ?

In theory, this can be solved by following three steps:

1. Getting the joint posterior distribution of (θ_1, θ_2) using the usual conjugate analysis for each of θ_1 and θ_2, and multiplying them together, since they are independent.
2. Do a change of variable, letting

$$\begin{aligned} \epsilon &= \theta_1 - \theta_2, \text{ and} \\ \theta_1 &= \theta_1 \end{aligned}$$

   This gives the joint posterior distribution of (θ_1, ϵ).
3. Integrate θ_1 out of the joint distribution in 3 to get the posterior of ϵ.

Problem: While this is sometimes feasible, the exact answer is a high degree polynomial (degree ≈ 250 for TIMI trial data).

Question: What alternatives are there, when exact posterior distributions are difficult or impossible to obtain directly?

Idea: Perhaps it will be sufficient to generate random variables from the posterior distribution. Thus the idea is to replace the posterior density curve by a histogram, and replace exact quantiles of the posterior distribution with sample quantiles, for example to get posterior means and variances, or 95% credible sets. If we can generate a large enough sample, we can attain any accuracy we desire.

We will see four such techniques in this course:

- Monte Carlo integration
- Direct simulation
- Sampling Importance Resampling (SIR)
- The Gibbs Sampler

Monte Carlo Integration

You may not know it, but every time you calculate a sample mean ($\overline{x}$) as an estimator of a mean μ, you are (sort of) using Monte Carlo integration.

Remember the definition of a mean of a continuous variable:

$$\mu = \int_{-\infty}^{\infty} x \times f(x)dx$$

Suppose this integral is too difficult to do analytically. How could we approximate it?

Think of what the formula means: For each value of x, we "weight" it how often it occurs, $f(x)$. If we do the integral exactly, we get the exact average value of x. What happens if we can generate a random sample of x's from $f(x)$?

Take $x_1, x_2, \ldots, x_m \sim f(x)$. If done correctly, the x_i's drawn randomly are *already* weighted according to $f(x)$. Thus, to approximate the weighted average, just average the x_i's! This is a simple form of Monte Carlo integration.

What if we wanted to approximate the variance of a continuous variable? Remember the definition of the variance in this case:

$$\sigma^2 = \int_{-\infty}^{\infty} (x - \mu)^2 \times f(x)dx$$

Note that this is simply the function $(x - \mu)^2$, weighted by $f(x)$. So, the solution is very similar to what we had before:

Take $x_1, x_2, \ldots, x_m \sim f(x)$. Convert each of these sample points to $(x_1 - \mu)^2, (x_2 - \mu)^2, \ldots, (x_m - \mu)^2$. If done correctly, the $(x_i - \mu)^2$'s are again already weighted according to $f(x)$. Thus, to approximate the variance, just average the $(x_i - \mu)^2$'s.

Note the general pattern: We have an integral of the form

$$I = \int_{-\infty}^{\infty} g(x) \times f(x) dx$$

that we wish to evaluate (i.e., find the value of the integral, generically denoted by I here. To solve this integral, take a random sample of x_i's from f(x), and for each, calculate $g(x_i)$. Then take the average of these values, i.e., calculate

$$\frac{\Sigma_{i=1}^{m} g(x_i)}{m}$$

Note that Monte Carlo simulation as discussed here is very general, and it is as useful to pure mathematics as it is to statistics. If you have any integral that looks like

$$\int_{-\infty}^{\infty} g(x) \times f(x) dx$$

where either f or g are probability distributions, this method can be used to approximate the integral. Clearly, the larger the value of m, the better the approximation will be, in general.

Calculating an RR for our two binomial problem

Let's return to our TIMI data. Suppose we wish to calculate the relative risk for the treatment versus control group. We know that

$\theta_1 \sim beta(a_1 + x_1, b_1 + n_1 - x_1)$

and

$\theta_2 \sim beta(a_2 + x_2, b_2 + n_2 - x_2)$,

from conjugacy within each group, where the a's and b's are the prior parameters. Now, from the independence of the two groups, we know that

$$f(\theta_1, \theta_2) \sim beta(a_1 + x_1, b_1 + n_1 - x_1) \times beta(a_2 + x_2, b_2 + n_2 - x_2)$$

Note that we can easily simulate from this *joint* distribution, by simulating from each beta distribution separately.

Now suppose we want to find the mean posterior relative risk. This is defined by:

$$mean(RR) = \int_{-\infty}^{\infty} \frac{\theta_2}{\theta_1} \times f(\theta_1, \theta_2) dx$$

By Monte Carlo integration, we simply need to simulate a sample of $(\theta_{1i}, \theta_{2i})$ samples, calculate $RR = \theta_2/\theta_1$ from each, and average these values.

Here is a quick R program that does this (assumes $a_1 = b_1 = a_2 = b_2 = 1$, i.e., uniform priors:

```
> theta1 <- rbeta(10000, 79, 41)
> theta2 <- rbeta(10000, 45, 79)
> rr <- theta1/theta2
> mean.rr <- mean(rr)
```

Running this program in R produces mean.rr = 1.84.

In solving integrals involved in Bayesian analyses by Monte Carlo simulation, we were able to gets posterior means and variances, for example. However, we usually want to calculate entire posterior distributions, not just univariate summaries of them. There are several ways we will see how to approximate entire posterior distributions. We will first see the most simple way, which is only a very small variation of Monte Carlo integration, which is called "direct sampling" (by some) or simply Monte Carlo simulation (by others).

Direct Simulation

In this problem, we know

$$\theta_1 \sim beta(a_1 + x_1, b_1 + n_1 - x_1)$$

and

$$\theta_2 \sim beta(a_2 + x_2, b_2 + n_2 - x_2).$$

If we have a random number generator that can generate beta random numbers, we can get a sample from the posterior distribution of any quantity that depends only on θ_1 and θ_2. For example, the following steps produce a sample from the correct posterior distribution for $\epsilon = \theta_1 - \theta_2$:

Direct Simulation

1. Simulate θ_1^* from $beta(a_1 + x_1, b_1 + n_1 - x_1)$
2. Simulate θ_2^* from $beta(a_2 + x_2, b_2 + n_2 - x_2)$
3. Form $\epsilon_1^* = \theta_1^* - \theta_2^*$
4. Repeat steps 1, 2, and 3 a large number (M) of times to get $\epsilon_1^*, \epsilon_2^*, \ldots, \epsilon_M^*$. This is a sample from the correct posterior distribution. We can now approximate the posterior distribution by the histogram of ϵ_1^*, $\epsilon_2^*, \ldots \epsilon_M^*$, or get posterior quantiles from the sample quantiles.

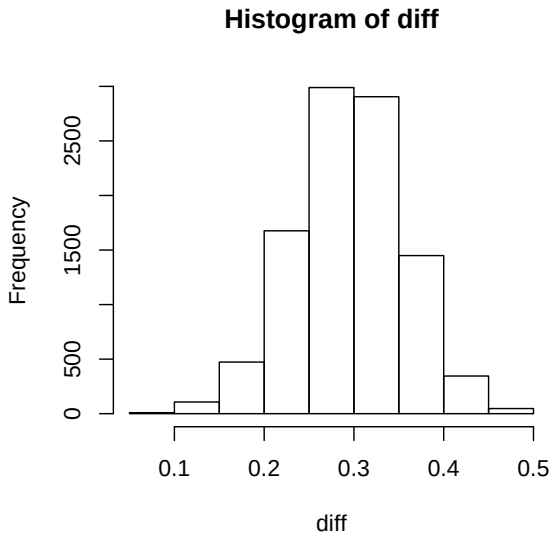
To code this in R, the first few steps are identical to what we had before for Monte Carlo integration, except we replace the RR by the difference in probabilities $\theta_1 - \theta_2$:

```
> theta1 <- rbeta(10000, 79, 41)
> theta2 <- rbeta(10000, 45, 79)
> diff <- theta1-theta2
```

Here the similarity ends, however. Rather than taking the mean of the r quantity "diff", we will graph it. Typing

```
>  hist(diff)
```

in R gives the following graphic:



Similarly, if we want a 95% credible interval, we simply type

```
>  quantile(diff, prob=c(0.025, 0.975))
```

which gives as the result:

```
     2.5%      97.5%
0.1741083 0.4127239
```

Sampling Importance Resampling

Direct simulation requires that the posterior distributions of θ_1 and θ_2 are directly available to be simulated.

What can be done if this is not the case?

Usually we can write down the prior and the likelihood, so that usually the function $prior \times likelihood$ is available. The problem is that the normalizing constant is missing, so direct simulation is not possible.

Idea: Start with a random sample from a guess at the posterior distribution, and modify the sample to be more like the true posterior.

Setup: Let

$$g(\theta)$$

be a first guess approximation to the true posterior distribution. Let

$$f^{'}(\theta|data) = likelihood \ \times \ prior,$$

which we can write down.

Sampling Importance Resampling

The SIR algorithm follows three steps:

1. Sample: Draw a random sample from $g(\theta)$, θ_1^*, $\theta_2^*, \ldots \theta_M^*$.
2. Importance: Attach a weight to each element in the sample,
$$w_i = \frac{f'(\theta_i^*|data)}{g(\theta_i^*)}$$
3. Resample: Draw a sample with replacement from the original sample θ_1^*, $\theta_2^*, \ldots \theta_M^*$, using the weights calculated in step 2.

The sample in step 3 is an approximate random sample from the posterior distribution, without needing to directly simulate from it.

Using SIR in our Example

Letting $\epsilon = \theta_1 - \theta_2$, so that $\theta_2 = \theta_1 - \epsilon$.

The likelihood $\times$ prior is then given by

$$\theta_1^{a_1+x_1-1}(1-\theta_1)^{b_1+n_1-x_1-1}(\theta_1-\epsilon)^{a_2+x_2-1}(1-(\theta_1-\epsilon))^{b_2+n_2-x_2-1}$$

Note that for any given value of θ_1, ϵ is restricted to the region

$$\theta_1 \leq \epsilon \leq 1-\theta_1$$

We will use a uniform distribution over this region for our $g(\theta_1, \epsilon)$ function.

The SIR steps then are:

1. Draw a random sample of size M from $g(\theta_1, \epsilon)$. This is accomplished by first selecting θ_1 from a uniform distribution on the interval [0,1], and then selecting ϵ randomly from a uniform distribution on the interval

$$-\theta_1 \leq \epsilon \leq 1 - \theta_1$$

   where θ_1 is the value just selected.

2. For each $\epsilon_i,\ i = 1, \ldots, M$ selected, assign a weight, given by

$$w_i = \theta_{1i}^{a_1+x_1-1}(1-\theta_{1i})^{b_1+n_1-x_1-1}(\theta_{1i}-\epsilon_i)^{a_2+x_2-1}(1-(\theta_{1i}-\epsilon_i))^{b_2+n_2-x_2-1}$$

3. Resample a sample of size $M^{'}$ (which need not equal M) with replacement from the original (uniform) sample of ϵ's, with probability w_i as the sampling weight. This latter sample is the random sample from the posterior distribution of ϵ that we desire.

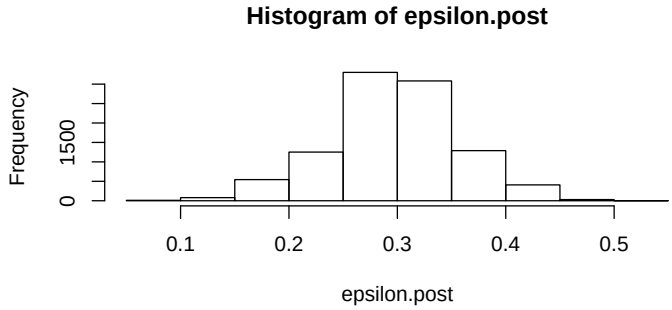
Here is an R program that does this:

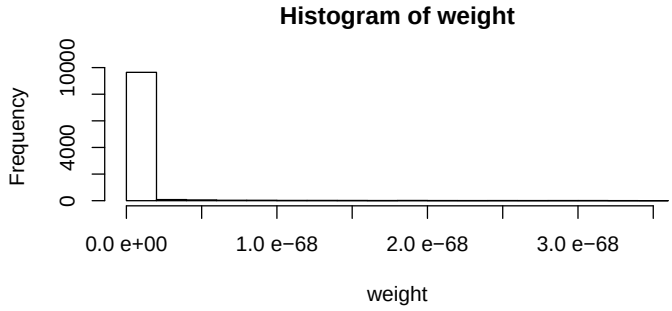
```
SIR.for.epsilon.1 <- function(a1, b1, a2, b2, x1, n1, x2, n2, size=10000)
{
theta1 <- runif(size, min=0,max=1)
epsilon <- runif(size, min=theta1-1, max=theta1)
weight <- theta1^(a1+x1-1)*(1-theta1)^(b1+n1-x1-1)*
    ( theta1 - epsilon)^(a2+x2-1)*(1- (theta1 - epsilon))^(b2+n2-x2-1)
epsilon.post <- sample(epsilon, size, replace=T, prob=weight)
layout(matrix(c(1,2), byrow=T, nrow=2))
hist(epsilon.post)
```

```
hist(weight)
return(quantile(epsilon.post, prob=c(0.025, 0.25, 0.5, 0.75, 0.975)))
}
SIR.for.epsilon.1(1,1,1,1,78, 118, 44, 122, 10000)
     2.5%       25%       50%       75%     97.5%
0.1780713 0.2603438 0.2969588 0.3378202 0.4149166
```

Note that the solutions are identical to what we had before using direct simulation.

Here are the graphs:





There were no problems calculating the weights here, but they can sometimes be problematic, especially in large data sets, where all weights may appear to be zero. A way around this is to calculate all terms in the weights first on a log scale, then convert back. Here is the program, identical expect for the way the weights are calculated:

```
SIR.for.epsilon.2 <- function(a1, b1, a2, b2, x1, n1, x2, n2, size=10000)
{
```

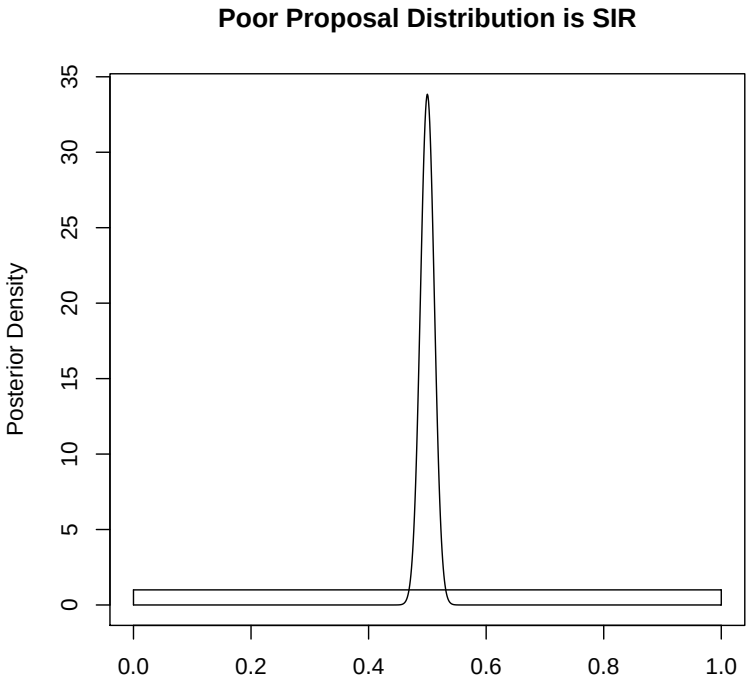
```
theta1 <- runif(size, min=0,max=1)
epsilon <- runif(size, min=theta1-1, max=theta1)
log.weight <- (a1+x1-1)*log(theta1) + (b1+n1-x1-1)* log(1-theta1) +
      (a2+x2-1)*log( theta1 - epsilon) + (b2+n2-x2-1)* log(1- (theta1 - epsilon))
weight <- exp(log.weight)
epsilon.post<-sample(epsilon, size, replace=T, prob=weight)
layout(matrix(c(1,2), byrow=T, nrow=2))
hist(epsilon.post)
hist(weight)
return(quantile(epsilon.post, prob=c(0.025, 0.25, 0.5, 0.75, 0.975)))
}

SIR.for.epsilon.2(1,1,1,1,78, 118, 44, 122, 10000)
     2.5%       25%       50%       75%     97.5%
0.1700592 0.2563583 0.2928192 0.3324690 0.4095839
```

Note again the identical results. In general, the second method is preferable to the first, it is less "error prone".

There is still another problem, however. We can see from the graphs of the weights that almost all weights are near zero, with just a handful further from zero. This means that in resampling, the same small set of values will be chosen over and over again. This, in turn, can lead to "grainy" histograms, made up of just a few large values used over and over, where we know that the real density is a smooth curve.

This problem arises because of a poor choice of proposal distribution, where most values are far from where the posterior density is located, and so are given very low weight. Here is a picture that illustrates what is happening:



This problem is fixed by using a better proposal. Note that when you change the proposal, you must also change the weights by the same amount.

```
SIR.for.epsilon.3 <- function(a1, b1, a2, b2, x1, n1, x2, n2, size=10000)
{
theta1 <- runif(size, min=0,max=1)
epsilon <- rbeta(size, 30,70)
weight <- theta1^(a1+x1-1)*(1-theta1)^(b1+n1-x1-1)*
    ( theta1 - epsilon)^(a2+x2-1)*
     (1- (theta1 - epsilon))^(b2+n2-x2-1)/dbeta(epsilon, 30, 70)
epsilon.post<-sample(epsilon, length(epsilon), replace=T, prob=weight)
layout(matrix(c(1,2), byrow=T, nrow=2))
hist(epsilon.post)
hist(weight)
return(quantile(epsilon.post, prob=c(0.025, 0.25, 0.5, 0.75, 0.975)))
}
```

```
SIR.for.epsilon.3(1,1,1,1,78, 118, 44, 122, 10000)
     2.5%       25%       50%       75%     97.5%
0.1841660 0.2575468 0.2947968 0.3348847 0.4088054
```

The Gibbs Sampler

Bayes Theorem is:

$$f(\theta|x) = \frac{l(x;\theta)f(\theta)}{\int l(x;\theta)f(\theta)d\theta}$$

Problem: $\int l(x;\theta)f(\theta)d\theta$ may be difficult.

We have seen so far that direct simulation or SIR could be used, but very often may not have distributions to simulate from directly and SIR only works well in low dimensional problems (almost impossible to get good "proposal" distributions in high dimensions). We need a general method that works in high dimensions.

The Gibbs sampler – General Idea

- Similar to Direct Sampling, results in a sample from the desired density or densities, does not give a formula for the density directly.
- Usually used to get samples from marginal densities, but can also be used for joint (that is, $f(\theta_1, \theta_2)$) densities.
- Iterative algorithm, while Direct Sampling was non-iterative
- Gibbs sampler is a special case of a family of similar algorithms called Monte Carlo Markov Chain (MCMC) algorithms.

Example: In two dimensions, may be interested in the marginal posterior density of θ_1:

$$f(\theta_1|x) = \int f(\theta_1, \theta_2|x) d\theta_2$$

Problem: $\int f(\theta_1, \theta_2|x) d\theta_2$ may be difficult.

Idea: Exploit relationships between conditional and marginal densities to simplify the problem, that is, break down one harder step into several easier steps.

Mathematical Statistical Background

Useful identities concerning probability densities:

1.

$$\begin{aligned} f(\theta_1, \theta_2) &= f(\theta_1|\theta_2) f(\theta_2) \\ &= f(\theta_2|\theta_1) f(\theta_1) \end{aligned}$$

2.

$$\begin{aligned} f(\theta_1) &= \int f(\theta_1, \theta_2) d\theta_2 \\ &= \int f(\theta_1|\theta_2) f(\theta_2) d\theta_2 \end{aligned}$$

3. Similarly,

$$\begin{aligned} f(\theta_2) &= \int f(\theta_2, \theta_1) d\theta_1 \\ &= \int f(\theta_2|\theta_1) f(\theta_1) d\theta_1 \end{aligned}$$

Densities $\Longleftrightarrow$ Samples

1. Can replace knowing $f(\theta)$ by a sample $\theta_1^*, \theta_2^*, \ldots \theta_M^*$ from $f(\theta)$.

2. From previous page,

$$f(\theta_1) = \int f(\theta_1|\theta_2) f(\theta_2) d\theta_2$$

   To get a sample from $f(\theta_1)$, a "two-stage" process can be used:

   (a) Draw θ_2^* from $f(\theta_2)$

   (b) Draw $\theta_1^*|\theta_2^*$ from $f(\theta_1|\theta_2^*)$

This is helpful if $f(\theta_1|\theta_2)$ is simpler than $f(\theta_1)$.

Gibbs sampler algorithm – two dimensions

Situation: Want to generate random samples from posterior marginal densities of θ_1 and θ_2. The joint density $f(\theta_1, \theta_2)$ is difficult to work with, hence the marginal densities $f(\theta_1)$ and $f(\theta_2)$ are not available for sampling from, either directly or using SIR. However, the conditional densities $f(\theta_1|\theta_2)$ and $f(\theta_2|\theta_1)$ are easier to work with, and are "available" to sample from.

If the likelihood $\times$ prior can be written down (as is usually the case), the conditional densities are usually derivable by "simplifying" this function.

Example

For the problem of a difference between two binomial densities, we had the likelihood $\times$ prior for ϵ, θ_1 as:

$$f(\theta_1, \epsilon) \propto \theta_1^{a_1+x_1-1}(1-\theta_1)^{b_1+n_1-x_1-1}(\theta_1-\epsilon)^{a_2+x_2-1}(1-(\theta_1-\epsilon))^{b_2+n_2-x_2-1}$$

Then, to get $f(\theta_1|\epsilon)$, assume ϵ is a constant, to get:

$$f(\theta_1|\epsilon) \propto \theta_1^{a_1+x_1-1}(1-\theta_1)^{b_1+n_1-x_1-1}(\theta_1-\epsilon)^{a_2+x_2-1}(1-(\theta_1-\epsilon))^{b_2+n_2-x_2-1}$$

where ϵ is treated as constant. Then, to get $f(\epsilon|\theta_1)$, assume θ_1 is a constant, to get:

$$\begin{aligned} f(\epsilon|\theta_1) &\propto \theta_1^{a_1+x_1-1}(1-\theta_1)^{b_1+n_1-x_1-1}(\theta_1-\epsilon)^{a_2+x_2-1}(1-(\theta_1-\epsilon))^{b_2+n_2-x_2-1} \\ &\propto (\theta_1-\epsilon)^{a_2+x_2-1}(1-(\theta_1-\epsilon))^{b_2+n_2-x_2-1} \end{aligned}$$

The last reduction is because θ_1 is a constant.

Gibbs sampler Algorithm – Two Dimensions

1. Start with any $\theta_2^{(0)}$. Set $i = 0$.
2. Draw $\theta_1^{(i+1)}$ from $f(\theta_1|\theta_2^{(i)})$
3. Draw $\theta_2^{(i+1)}$ from $f(\theta_2|\theta_1^{(i+1)})$
4. Set $i = i + 1$. Repeat steps 2 and 3, M times to get samples

$$\theta_1^{(1)}, \theta_1^{(2)}, \ldots, \theta_1^{(M)} \sim f(\theta_1)$$

$$\theta_2^{(1)}, \theta_2^{(2)}, \ldots, \theta_2^{(M)} \sim f(\theta_2)$$

Note: We always sample from the conditional distributions rather than marginal densities, which we assume are simpler. In fact, we saw how conditional densities are directly available from the likelihood $\times$ prior, so they can always be written down from there.

Gibbs sampler Algorithm – Three Dimensions

1. Start with any $\theta_2^{(0)}$ and $\theta_3^{(0)}$. Set $i = 0$.
2. Draw $\theta_1^{(i+1)}$ from $f(\theta_1|\theta_2^{(i)}, \theta_3^{(i)})$
3. Draw $\theta_2^{(i+1)}$ from $f(\theta_2|\theta_1^{(i+1)}, \theta_3^{(i)})$
4. Draw $\theta_3^{(i+1)}$ from $f(\theta_3|\theta_1^{(i+1)}, \theta_2^{(i+1)})$
5. Set $i = i+1$. Repeat steps 2, 3 and 4, M times to get samples

$$\theta_1^{(1)}, \theta_1^{(2)}, \ldots, \theta_1^{(M)} \sim f(\theta_1)$$
$$\theta_2^{(1)}, \theta_2^{(2)}, \ldots, \theta_2^{(M)} \sim f(\theta_2)$$
$$\theta_3^{(1)}, \theta_3^{(2)}, \ldots, \theta_3^{(M)} \sim f(\theta_3)$$

Gibbs sampler algorithm for $\epsilon = \theta_1 - \theta_2$

1. Start with any $\theta_1^{(0)}$. Set $i = 0$.
2. Draw $\epsilon^{(i+1)}$ from $f(\epsilon|\theta_1^{(i)})$
3. Draw $\theta_1^{(i+1)}$ from $f(\theta_1|\epsilon^{(i+1)})$
4. Set $i = i + 1$. Repeat steps 2, and 3 M times to get samples

$$\theta_1^{(1)}, \theta_1^{(2)}, \ldots, \theta_1^{(M)} \sim f(\theta_1)$$

$$\epsilon^{(1)}, \epsilon^{(2)}, \ldots, \epsilon^{(M)} \sim f(\epsilon)$$

Recall that:

$$f(\theta_1|\epsilon) \propto \theta_1^{a_1+x_1-1}(1-\theta_1)^{b_1+n_1-x_1-1}(\theta_1-\epsilon)^{a_2+x_2-1}(1-(\theta_1-\epsilon))^{b_2+n_2-x_2-1}$$

and

$$f(\epsilon|\theta_1) \propto (\theta_1 - \epsilon)^{a_2+x_2-1}(1 - (\theta_1 - \epsilon))^{b_2+n_2-x_2-1}$$

Both of these may be sampled from using SIR.

Gibbs Program and 2 needed subroutines

```
> gibbs.eps <-
function(x1, n1, x2, n2, size1, size2) {
        p.samp <- c()
        eps.samp <- c()
        p.samp[1] <- x1/n1
        for(i in 1:(size2 - 1)) {
                eps.samp[i] <- nexteps(p.samp[i], x1, n1, x2, n2, size1)
                p.samp[i + 1] <- nextpi(eps.samp[i], x1, n1, x2, n2, size1)
        }
         eps<-eps.samp[1:(size2 - 1)]
        return(eps)
}
> nexteps <-
function(p, x1, n1, x2, n2, size) {
        eps <- runif(size, min =  - p, max = 1 - p)
        w <- p^(x1) * (1 - p)^(n1 - x1) * (p + eps)^(x2) * (1 - p - eps)^(n2 -
                x2)
        eps.samp <- sample(eps, 1, replace = T, prob = w)
        return(eps.samp)
}
> nextpi <-
function(eps, x1, n1, x2, n2, size) {
        p <- runif(size, min = max( - eps, 0), max = min(1, 1 - eps))
        w <- p^(x1) * (1 - p)^(n1 - x1) * (p + eps)^(x2) * (1 - p - eps)^(n2 -
                x2)
        p.samp <- sample(p, 1, replace = T, prob = w)
        return(p.samp)
}
```

To run the program, type (-a is used since the program was written
for theta2-theta1 rather than theta1-theta2):

```
> a<-gibbs.eps(78,118,44,122,1000,5000)
> hist(-a)
> quantile(-a,prob=c(0.025,0.5,0.975))
      2.5%     50.0%     97.5%
 0.1661262 0.2981026 0.4157432
```