Bayes: Homework 7
Leslie Gains-Germain

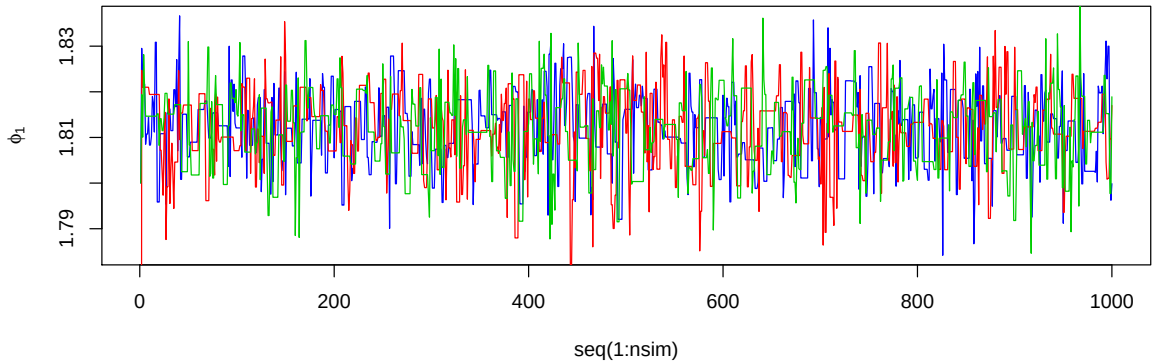
1. (a) To write an independence chain metropolis algorithm, I grabbed the code from the tutorial and changed the mean of the jumping distribution to the posterior mode $(\phi_1, \phi_2, \phi_3) = (1.81, -4.01, -1.06)$, with a variance covariance matrix that is twice that of the posterior. All of the code is shown in the appendix, but below the part of the code that I modified is printed.

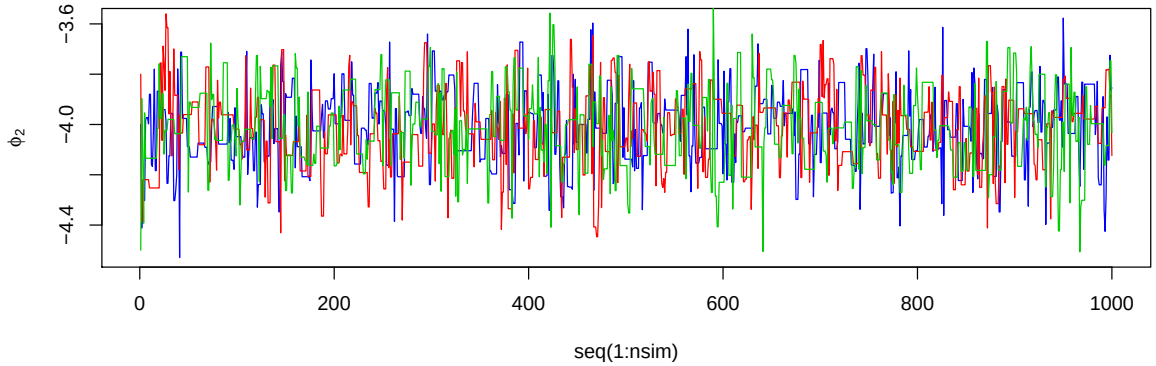
```
phi.cand1 <- rmvnorm(1, mean=mode, sigma=2*VarCov)
phi.cand2 <- rmvnorm(1, mean=mode, sigma=2*VarCov)
phi.cand3 <- rmvnorm(1, mean=mode, sigma=2*VarCov)
```

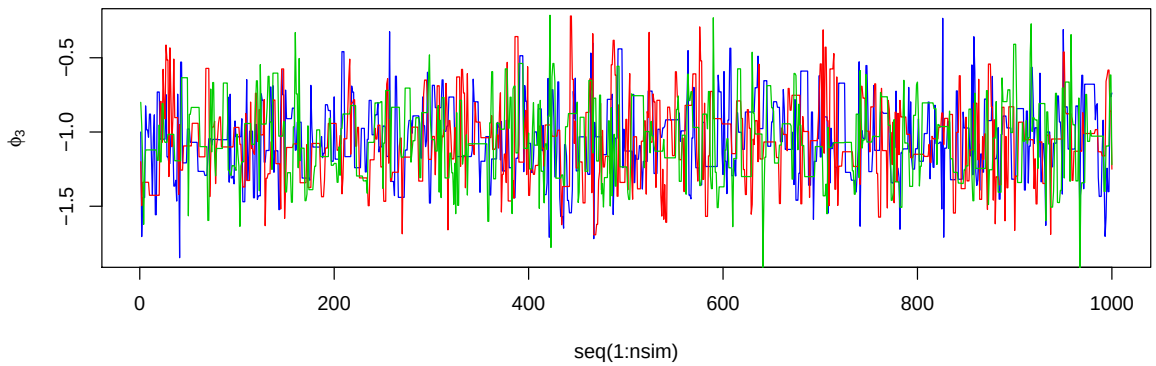
I ran the Metropolis algorithm, and the sample path plots are shown below, for all 1000 iterations. Convergence looks good. All three chains appear to be sampling from the entire parameter space. The candidate acceptance rates are close to 0.4 for all three chains. All $\hat{R}$ values are close to 1, and the effective sample sizes are around 1000. With only 1000 iterations, this shows that there is little dependence among the draws (which we would expect from an independence chain algorithm!). As a result, we don't need tons of iterations for the algorithm to converge as long as we chose and tuned the proposal distribution correctly.

	Point est.	Upper C.I.	eff
var1	1.00	1.00	1310.96
var2	1.01	1.03	837.50
var3	1.00	1.01	1007.11

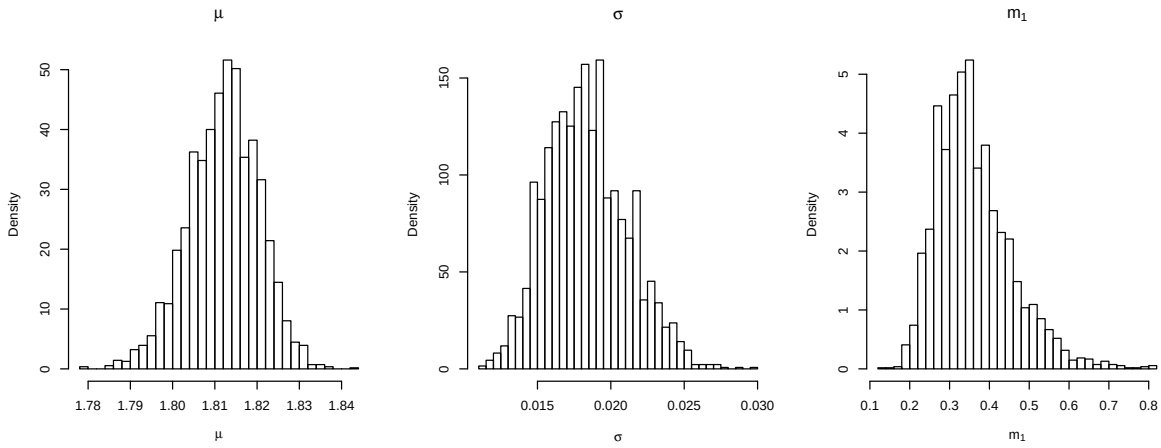
I think a burn in period of 100 iterations is appropriate. My justification for this is the appearance of the sample path plots. For all of the parameters, the draws look like they are coming from the distribution of interest after only the first few iterations. I think 100 burn in iterations might be overkill, but I figured that it is better to throw away some extra draws than to use draws that might not have converged yet.







The histograms of draws (minus the 100 burn in iterations) are shown for each parameter below. S



(b) Below, I implement a Gibbs sampler to draw from the 3 parameter posterior distribution. At each step of the algorithm, I use a Metropolis Hastings algorithm to sample from the complete condiditional distributions. The code is shown here.

```
##Write function to compute log likelihood for theta=(mu, sig2, m1) given the data
llik.fun <- function(theta.vec, dose.vec, y.vec, n.vec) {
       mu <- theta.vec[1]
       sig <- theta.vec[2]
       m1 <- theta.vec[3]
       x.vec <- (dose.vec - mu)/sig
     llik.vec <- m1*y.vec*(x.vec-log(1+exp(x.vec))) +
       (n.vec - y.vec)*log(1-((exp(x.vec)/(1+exp(x.vec)))^m1))
       out <- sum(llik.vec)
       return(out)
  }

### prior function (transformed parameters)
l.prior.fun2 <- function(phi.vec, a0=0.25, b0=0.25, c0=2, d0=10, e0=2.000004, f0=0.001) {
      phi1 <- phi.vec[1]
      phi2 <- phi.vec[2]
      phi3 <- phi.vec[3]
      log.p.phi1 <- log(dnorm(phi1, mean=c0, sd=d0))
      log.p.phi2 <- -2*e0*phi2 - (f0*(exp(-2*phi2)))
      log.p.phi3 <- phi3*a0 - (b0*exp(phi3))
      log.p <- log.p.phi1 + log.p.phi2 + log.p.phi3 #assuming priors independent
      return(log.p)
    }

#takes transformed params as inputs
#evaluates conditional distribution of a parameter, given values of the other parameters
l.unpost.fun2 <- function(phi.vec, dose.vec, y.vec, n.vec) {
     theta.vec <- c(phi.vec[1], exp(phi.vec[2]), exp(phi.vec[3]))
     llik <- llik.fun(theta.vec, dose.vec=dose.vec, y.vec=y.vec, n.vec=n.vec)
     lp <- l.prior.fun2(phi.vec)
     lout <- llik + lp
```

```
    return(lout)
   }
```

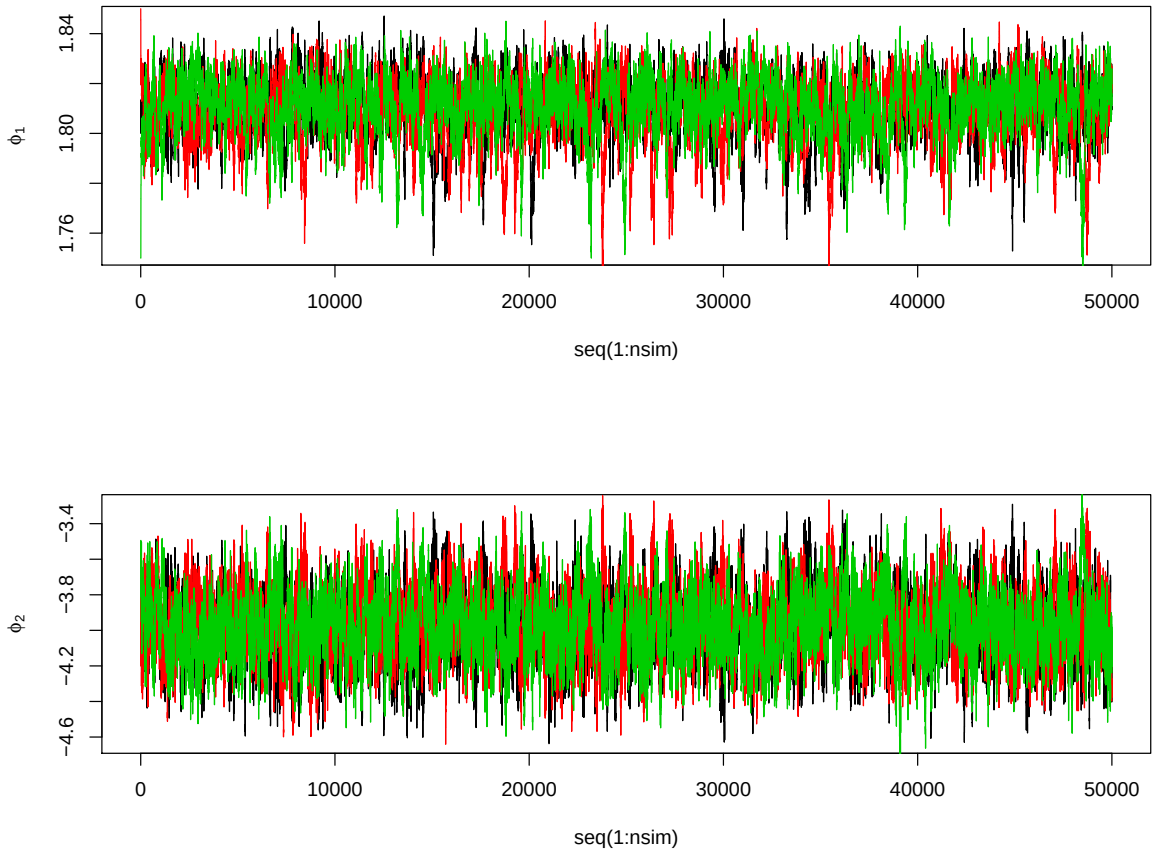
```
#goal - obtain draw from theta1|theta2, theta3
#then plug it in get draw from theta2|theta1,theta3
#then plug it in and get draw from theta3|theta1,theta2

   nchain <- 3
   nsim <- 50000  #number of iteration
   phi.mat <- array(NA, dim=c(nsim, 3, nchain)) #chain 1

   jump.vec1 <- numeric(nsim-1) #keep track of when we jump (accept candidates)
   jump.vec2 <- numeric(nsim-1)
   jump.vec3 <- numeric(nsim-1)

   phi.mat[1, 1:3, 1] <- c(1.8, -4, -1)  #1.543270 -4.874497  2.179276
   phi.mat[1, 1:3, 2] <- c(1.85, -4.2, -0.8)
   phi.mat[1, 1:3, 3] <- c(1.75, -3.8, -1.2)

   sd.scale <- c(.01,.2,.2)

  for(j in 1:nchain){
   for (i in 2:nsim) {
     phi.2 <- phi.mat[i-1, 2, j]
     phi.3 <- phi.mat[i-1, 3, j]
     phi.cur1 <- phi.mat[i-1, 1, j]

     phi.cand1 <- rnorm(1, mean=phi.cur1, sd=sd.scale[1])

     log.r.num1 <- l.unpost.fun2(c(phi.cand1, phi.2, phi.3), dose.vec=dose,
                                 y.vec=killed, n.vec=exposed) +
                   dnorm(phi.cur1, mean=phi.cand1, sd=sd.scale[1], log=TRUE)

     log.r.denom1 <- l.unpost.fun2(c(phi.cur1, phi.2, phi.3), dose.vec=dose,
                                   y.vec=killed, n.vec=exposed) +
                   dnorm(phi.cand1, mean=phi.cur1, sd=sd.scale[1], log=TRUE)

     log.r1 <- log.r.num1 - log.r.denom1

     p.accept1 <- min(1, exp(log.r1))

     u.vec <- runif(3)
     ifelse(u.vec[1] <= p.accept1, phi.mat[i, 1, j]<- phi.cand1,
            phi.mat[i, 1, j] <- phi.cur1)

     jump.vec1[i-1] <- ifelse(u.vec[1] <= p.accept1, 1, 0)

      #now draw theta2 given the values of theta1 and theta3
      phi.1 <- phi.mat[i, 1, j]
      phi.3 <- phi.mat[i-1, 3, j]
      phi.cur2 <- phi.mat[i-1, 2, j]
```

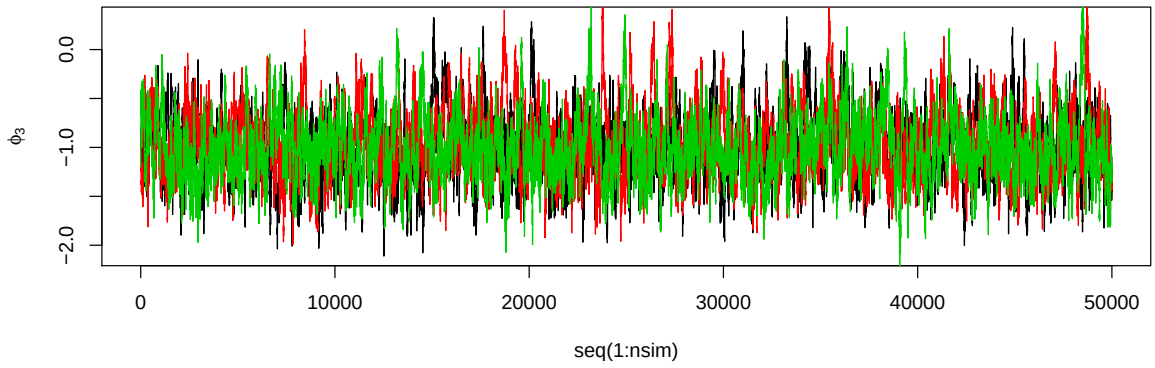
```
     phi.cand2 <- rnorm(1, mean=phi.cur2, sd=sd.scale[2])

    log.r.num2 <- l.unpost.fun2(c(phi.1, phi.cand2, phi.3), dose.vec=dose,
                                y.vec=killed, n.vec=exposed) +
                  dnorm(phi.cur2, mean=phi.cand2, sd=sd.scale[2], log=TRUE)

    log.r.denom2 <- l.unpost.fun2(c(phi.1, phi.cur2, phi.3), dose.vec=dose,
                                  y.vec=killed, n.vec=exposed) +
                  dnorm(phi.cand2, mean=phi.cur2, sd=sd.scale[2], log=TRUE)

    log.r2 <- log.r.num2 - log.r.denom2

    p.accept2 <- min(1, exp(log.r2))

    ifelse(u.vec[2] <= p.accept2, phi.mat[i, 2, j] <- phi.cand2,
           phi.mat[i, 2, j] <- phi.cur2)

    jump.vec2[i-1] <- ifelse(u.vec[2] <= p.accept2, 1, 0)

     #now draw theta3 given the values of theta2 and theta3
     phi.1 <- phi.mat[i, 1, j]
     phi.2 <- phi.mat[i, 2, j]

     phi.cur3 <- phi.mat[i-1, 3, j]

     phi.cand3 <- rnorm(1, mean=phi.cur3, sd=sd.scale[3])

    log.r.num3 <- l.unpost.fun2(c(phi.1, phi.2, phi.cand3), dose.vec=dose,
                                y.vec=killed, n.vec=exposed) +
                  dnorm(phi.cur3, mean=phi.cand3, sd=sd.scale[3], log=TRUE)

    log.r.denom3 <- l.unpost.fun2(c(phi.1, phi.2, phi.cur3), dose.vec=dose,
                                  y.vec=killed, n.vec=exposed) +
                  dnorm(phi.cand3, mean=phi.cur3, sd=sd.scale[3], log=TRUE)

    log.r3 <- log.r.num3 - log.r.denom3

    p.accept3 <- min(1, exp(log.r3))

    ifelse(u.vec[3] <= p.accept3, phi.mat[i, 3, j] <- phi.cand3,
           phi.mat[i, 3, j] <- phi.cur3)

    jump.vec3[i-1] <- ifelse(u.vec[3] <= p.accept3, 1, 0)
   }
  }
```

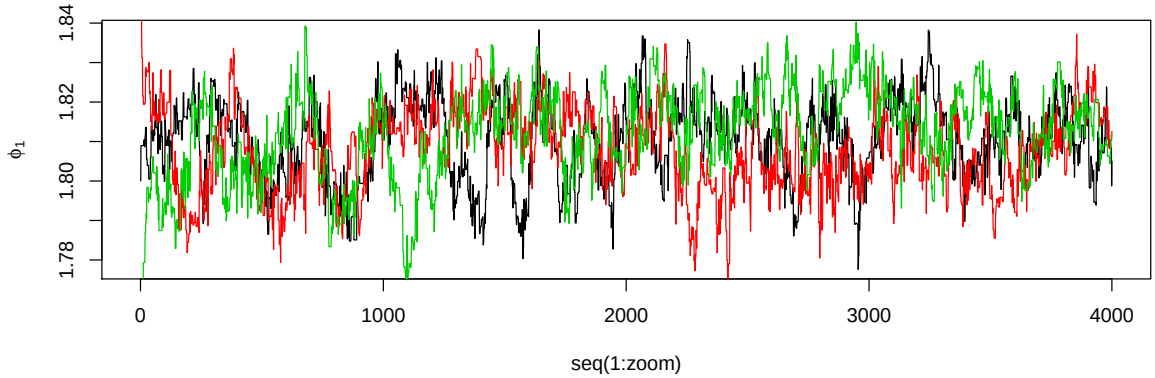
For the jumping distribution for ϕ_1, I chose a standard deviation of 0.01, and for the jumping distributions for ϕ_2 and ϕ_3, I chose standard deviations of 0.2. These gave me acceptance ratios of $0.406, 0.430$, and 0.427 for each parameter, respectively. Before I

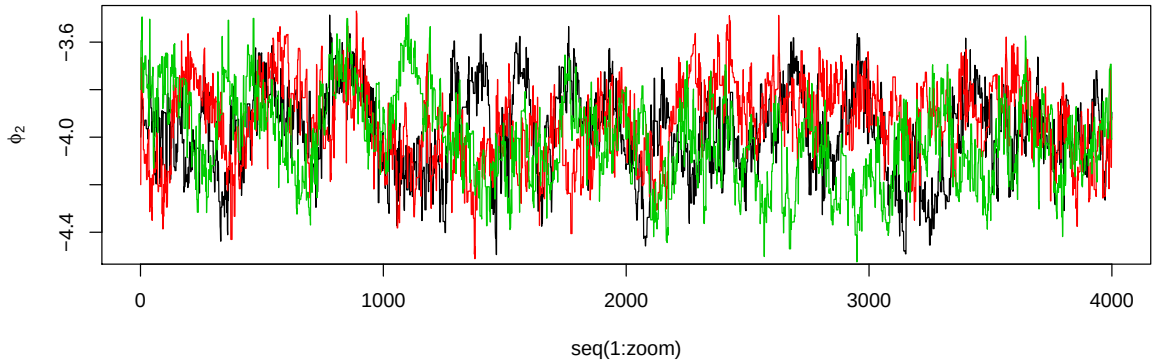
adjusted the standard deviations, the traceplots looked horrible. After I found standard deviations that gave me acceptance ratios near 0.4, the traceplots were improved. The traceplots for 50000 iterations are shown below.

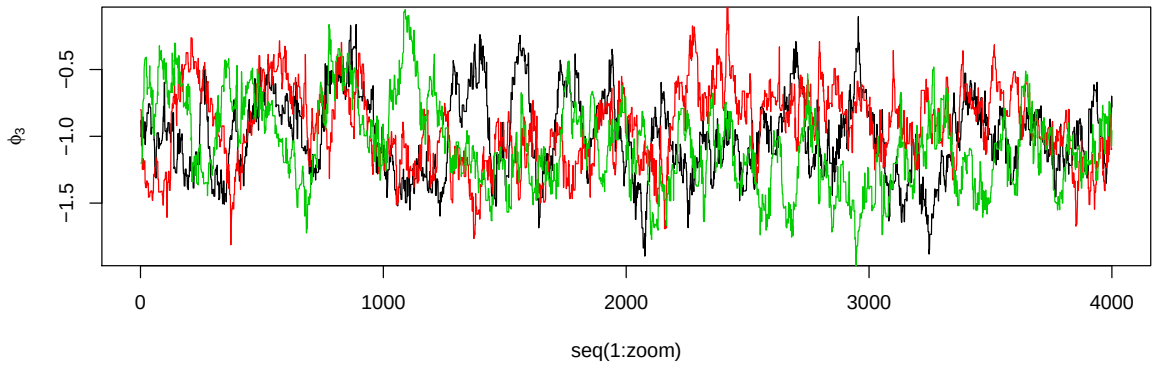
Based on the plots, I think a burn-in period of 500 iterations seems adequate. I zoomed in on the first 4000 iterations to get a better idea of how many burn in iterations should be used. Honestly, it doesn't seem like many burn-in iterations are needed at all. I looked at the output for the `gelman.plot` and `raftery.diag` functions, and they both suggested that the number of burn-in iterations needed is less than 100. I'll use 500 to be safe.




ϕ_3
0.0
−1.0
−2.0
0
10000
20000
30000
40000
50000
seq(1:nsim)


ϕ_1
1.84
1.82
1.80
1.78
0
1000
2000
3000
4000
seq(1:zoom)

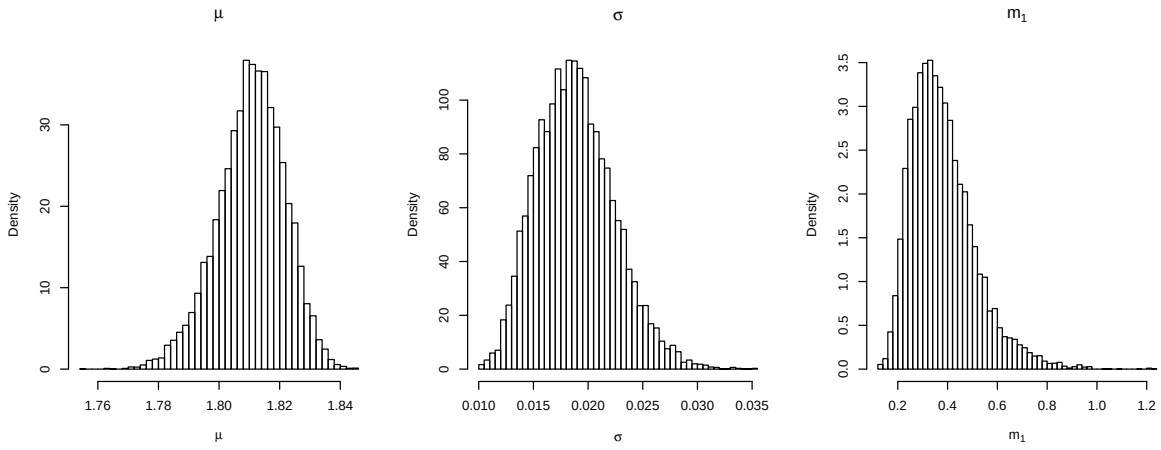

ϕ_2
−3.6
−4.0
−4.4
0
1000
2000
3000
4000
seq(1:zoom)



The $\hat{R}$ values were close to 1, but the effective sample sizes were small (see the table below) considering I ran 50000 iterations. You can see in the zoomed in plots above that each chain does take a while to move around the parameter space, with high corrrelation among the draws. With effective sample sizes near 1000, I wouldn't want to run less than 50000 iterations.

	Point est.	Upper C.I.	eff2
var1	1.01	1.02	1106.06
var2	1.00	1.02	1409.25
var3	1.01	1.02	1066.65

The histograms of draws for parameters m_1, μ, and σ are shown below.



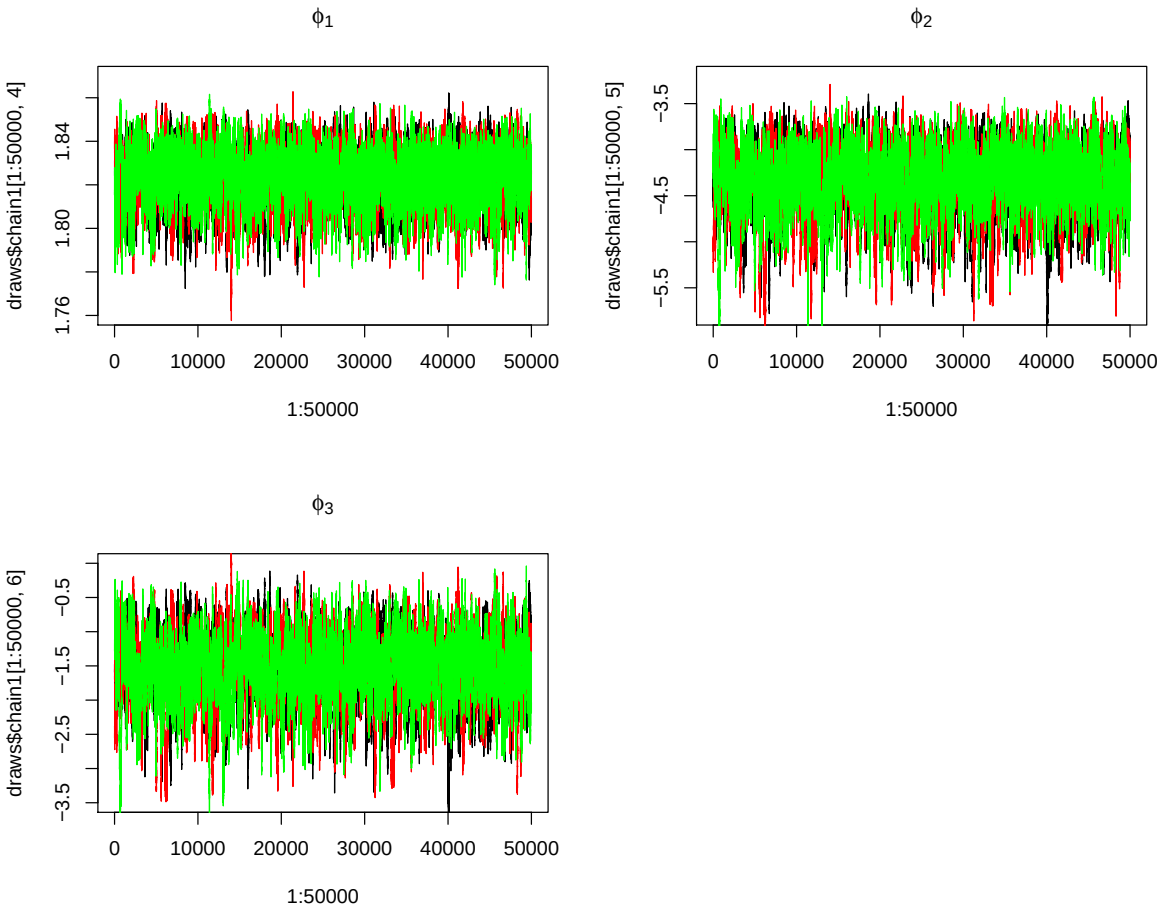
(c) I used JAGs to fit the model. The code for the model is shown below. Note that I started with 0 burn-in iterations, and then I refit the model with an appropriate burn-in after assessing the traceplots.

```
setwd("~/Documents/Stat532/homeworks/bayeshw7")

##write model file first
cat("
model
{
for(i in 1:N)
{
y[i] ~ dbin(p[i], n[i])
p[i] <- (exp((dose[i]-mu)*tau1)/(1+exp((dose[i]-mu)*tau1)))^m1
}

mu~dnorm(c0, tau0)
tau0 <- pow(d0, -2)
m1 ~ dgamma(a0, b0)
tau1 ~ dgamma(e0, f0)
sigma <- pow(tau1, -1)
phi1 <- mu
phi2 <- 1/2*log(sigma^2)
phi3 <- log(m1)

a0 <- 0.25
b0 <- 0.25
c0 <- 2

d0 <- 10
e0 <- 2.000004
f0 <- 0.001
}",
file="jags-beetles.jags")
```

```
##jags call
library(R2jags)
set.seed(52)

dose <- c(1.6907, 1.7242, 1.7552, 1.7842, 1.8113, 1.8369, 1.8610, 1.8839)
killed <- c(6, 13, 18, 28, 52, 53, 61, 60)
exposed <- c(59,60,62,56,63,59,62,60)

data <- list(N=length(killed), y=killed, n=exposed, dose=dose)

inits <- list(list(mu = 1.8, m1 = 0.367, tau1 = 55),
              list(mu = 1.75, m1= 0.6065, tau1 = 60),
              list(mu = 1.85, m1 = 0.5, tau1 = 43))

beetles <-jags(model.file="jags-beetles.jags", data = data,
```

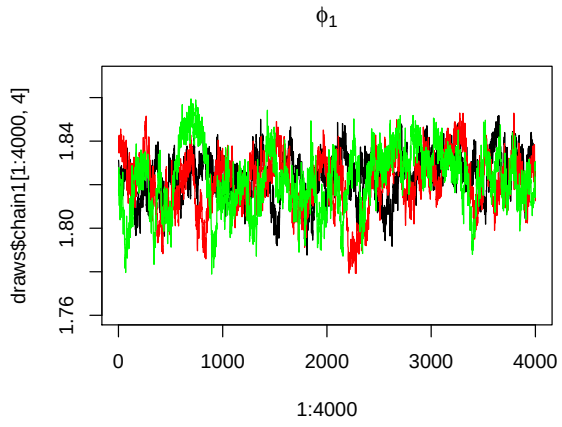
```
                parameters.to.save = c("mu", "m1", "sigma", "phi1", "phi2", "phi3"),
                n.chains=3, n.thin=1, inits=inits, n.burnin=0, n.iter=50000)
```

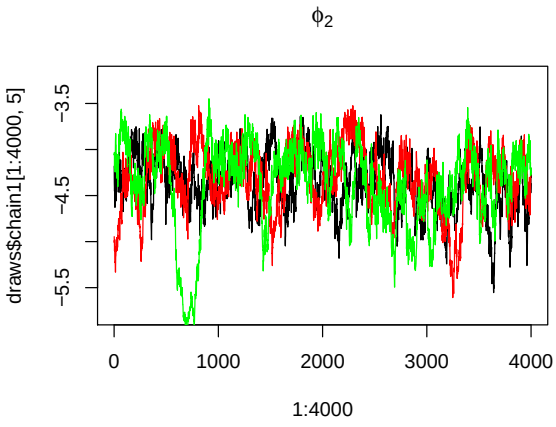
I ran three chains, and specified the starting values myself by looking at the posterior plots from the parts (a) and (b). I ran 50000 iterations and the traceplots are shown below. I show the traceplots on the transformed scale, because the distributions of the transformed parameters appear more symmetric than the distributions of the parameters before transformation. These traceplots look OK, but I do notice that it takes each chain a while to move around the parameter space(see traceplots on the next page for more detail of this). It looks similar to the traceplots in part (b). There are a few spikes in the traceplots, and I saw these spikes in part (b) as well. I don't think the spikes are a concern, however, because in 50000 iterations I would expect to see some unusual draws.

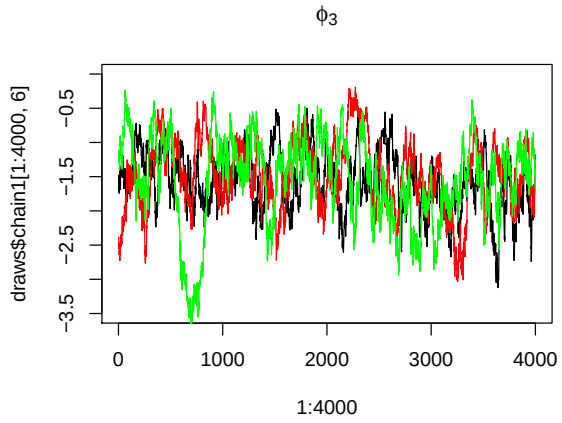


It was hard to decide how many iterations to throw away for the burn in from the above

plots, so I zoomed in the first 4000 iterations in the following plots. It seems like a burn in period of 1000 iterations is adequate for ϕ_1, ϕ_2, and ϕ_3. I also looked at the convergence diagnostic `raftery.diag`, and it suggested that the burn in period could be as low as 100 for some of the parameters. I'm a little suspicious of this, because the green chain shows a large spike in the earlier iterations (see plots on next page).



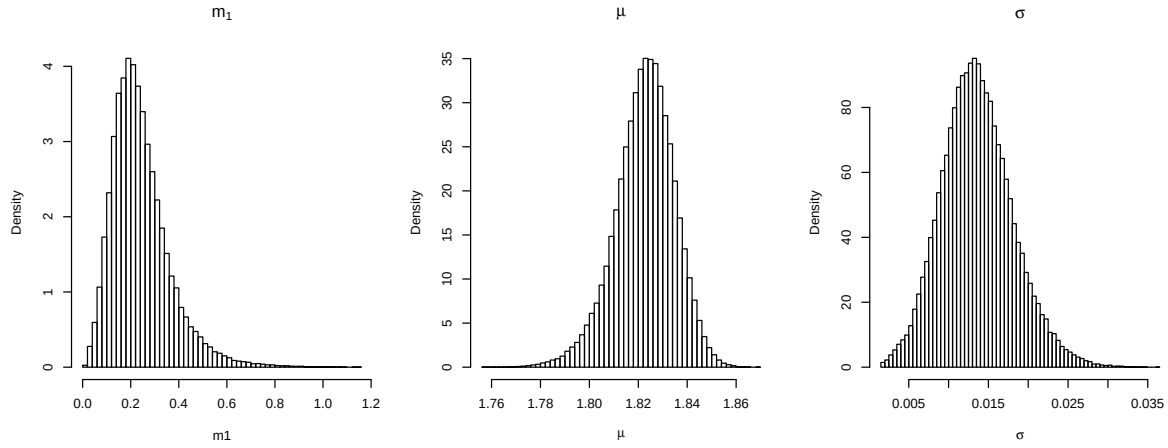





```
Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

          Burn-in  Total  Lower bound  Dependence
          (M)      (N)    (Nmin)       factor (I)
 deviance 12       17256  3746          4.61
 m1       144      148932 3746         39.80
 mu       102      121261 3746         32.40
 phi1     102      121261 3746         32.40
 phi2     105      110810 3746         29.60
 phi3     144      148932 3746         39.80
 sigma    105      110810 3746         29.60
```

So, I discarded the first 1000 iterations, and below I plot and display summary statistics for the posterior draws for each parameter, μ, m_1, and σ. I also looked at the below output to check the convergence diagnostics $\hat{R}$ and the effective sample sizes. The $\hat{R}$ values were close to 1 and the effective sample sizes were between 1000 and 2000. The effective sample sizes affirm what I indicated earlier - it did take a while for each chain to move around the whole parameter space. With high dependence among iterations, I don't think I would want to run less than 50000 iterations.

	mean	sd	2.5%	25%	50%	75%	97.5%	Rhat	n.eff
deviance	33.03	2.93	29.71	30.88	32.25	34.35	40.71	1.00	980.00
m1	0.24	0.12	0.07	0.16	0.22	0.31	0.54	1.00	1400.00
mu	1.82	0.01	1.80	1.81	1.82	1.83	1.84	1.00	3300.00
phi1	1.82	0.01	1.80	1.81	1.82	1.83	1.84	1.00	3300.00
phi2	-4.35	0.35	-5.18	-4.54	-4.31	-4.10	-3.77	1.01	1200.00
phi3	-1.53	0.50	-2.65	-1.83	-1.49	-1.19	-0.63	1.00	1400.00
sigma	0.01	0.00	0.01	0.01	0.01	0.02	0.02	1.01	1200.00



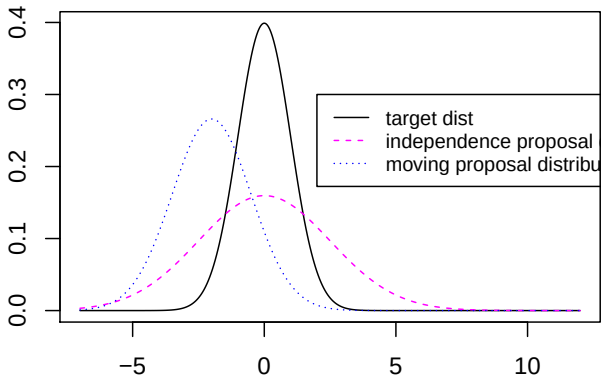
(d) In this section, I'll compare the independence chain Metropolis algorithm to the Metropolis Algorithm with a jumping distribution that does depend on the current value in the chain.

First, I'll address differences between the multivariate Metropolis-Hastings algorithm where each draw is from a multivariate jumping distribution and the Gibbs algorithm where Metropolis-Hastings is used to sample from the univariate complete conditional

distributions at each step.

In terms of understanding, I think the multivariate Metropolis-Hastings algorithm is easier to follow, and easier to program. For me, it's easier to understand the idea of making one multivariate draw at each step in the algorithm, rather than making three univariate draws at each step in the algorithm. When coding the multivariate Metropolis algorithm, you have to specify a variance covariance matrix of the jumping distribution, so I think it's made more explicit that you're accounting for correlation among the parameters. In the Gibbs sampler, the correlation among parameters is accounted for when drawing from the complete conditional distributions, but I think it's less explicit.

I do think that you could get away with a smaller standard deviation of the proposal distribution when the proposal distribution depends on the current parameter value. In the independence chain algorithm, the proposal distribution is not moving, so we want to choose a variance that is large enough that we can be sure that values from the entire parameter space can be proposed. If the proposal distribution moves with the current parameter value, however, the entire parameter space will eventually be sampled even if we choose a variance that is too small. A univariate illustration of my point is shown below. You can see that the spread of the independence proposal distribution seems like it should be larger than the spread of the moving proposal distribution.



The main advantage of the independence chain Metropolis-Hastings algorithm is speed of convergence. The tables below show the effective sample sizes for each method. You can see that we get about the same effective sample sizes for 7000 iterations from the multivariate M-H algorithm, 1000 iterations from the independence chain M-H algorithm, and 50000 iterations from the Gibbs sampler. I think the number of iterations required for convergence of the Gibbs sampler is a disadvantage in this scenario.

Table 1: Effective Sample Sizes for 7000 iterations of the multivariate Metropolis-Hastings algorithm.

	Point est.	Upper C.I.	eff1
var1	1.00	1.00	1033.45
var2	1.00	1.00	1258.33
var3	1.00	1.00	1127.45

Table 2: Effective Sample Sizes for 1000 iterations of the independence chain multivariate Metropolis-Hastings algorithm.

	Point est.	Upper C.I.	eff
var1	1.00	1.00	1310.96
var2	1.01	1.03	837.50
var3	1.00	1.01	1007.11

I think the main disadvantage of the Metropolis-Hastings algorithms is that we have to draw from a multivariate normal proposal distribution to obtain candidate values. For a

Table 3: Effective Sample Sizes for 50000 iterations of the Gibbs sampler (using the Metropolis-Hastings algorithm to sample from the complete conditionals).

	Point est.	Upper C.I.	eff2
var1	1.01	1.02	1106.06
var2	1.00	1.02	1409.25
var3	1.01	1.02	1066.65

three parameter setting, I don't think this is a big deal, but I can imagine that it becomes more difficult as the number of parameters increase. In Gibbs sampling, however, you only need to sample one variable at a time. I think this is a main reason why all these canned software packages use Gibbs sampling rather than Metropolis-Hastings. The canned software packages have to be able to run when the number of parameters is large. Suppose we had 50 parameters. I think it would be difficult to sample from a 50 parameter multivariate normal, so the Gibbs sampler is more adaptable to a variety of models.

R code appendix

```
### Based on example 3.7 in Carlin & Louis and data from Bliss(1935)
library(mvtnorm)

### Data: Number of adult flour beetles killed after 5 hours of exposure
##   to various levels of gaseous carbon disulphide (CS2)

dose <- c(1.6907, 1.7242, 1.7552, 1.7842, 1.8113, 1.8369, 1.8610, 1.8839)
killed <- c(6, 13, 18, 28, 52, 53, 61, 60)
exposed <- c(59,60,62,56,63,59,62,60)
```

```
##Write function to compute log likelihood for theta=(mu, sig2, m1) given the data
llik.fun <- function(theta.vec, dose.vec, y.vec, n.vec) {
       mu <- theta.vec[1]
       sig <- theta.vec[2]
       m1 <- theta.vec[3]
       x.vec <- (dose.vec - mu)/sig
     llik.vec <- m1*y.vec*(x.vec-log(1+exp(x.vec))) + (n.vec - y.vec)*log(1-((exp(x.vec)/(1+exp(x.vec)))^m1))
       out <- sum(llik.vec)
       return(out)
        }

###Transformed the prior parameters to the real line
## I worked out the prior distributions for the transformed parameters
##  on paper (you should do the same to check my work!). Transformed parameters:
##  phi1 = mu
```

```
##  phi2 = log(sigma) or (0.5)*log(sigma^2) ##CHECK THIS
##  phi3 = log(m1)
##  Priors are rather vague, but proper (See Carlin & Gelfand (1991b) for more info on specification)

### This first function takes the original params as inputs
l.prior.fun <- function(theta.vec, a0=0.25, b0=0.25, c0=2, d0=10, e0=2.000004, f0=0.001) {
      phi1 <- theta.vec[1]
      phi2 <- log(theta.vec[2])
      phi3 <- log(theta.vec[3])
      log.p.phi1 <- log(dnorm(phi1, mean=c0, sd=d0))
      log.p.phi2 <- (-2*e0*phi2) - (f0*(exp(-2*phi2)))
      log.p.phi3 <- (phi3*a0) - (b0*exp(phi3))
      log.p <- log.p.phi1 + log.p.phi2 + log.p.phi3 #assuming priors independent
      return(log.p)
    }

### This second function takes the transformed params as inputs
l.prior.fun2 <- function(phi.vec, a0=0.25, b0=0.25, c0=2, d0=10, e0=2.000004, f0=0.001) {
      phi1 <- phi.vec[1]
      phi2 <- phi.vec[2]
      phi3 <- phi.vec[3]
      log.p.phi1 <- log(dnorm(phi1, mean=c0, sd=d0))
      log.p.phi2 <- -2*e0*phi2 - (f0*(exp(-2*phi2)))
      log.p.phi3 <- phi3*a0 - (b0*exp(phi3))
      log.p <- log.p.phi1 + log.p.phi2 + log.p.phi3 #assuming priors independent
      return(log.p)
    }


#find posterior prob for given values of parameters
#takes untransformed params as inputs
l.unpost.fun <- function(theta.vec, dose.vec, y.vec, n.vec) {
     llik <- llik.fun(theta.vec, dose.vec=dose.vec, y.vec=y.vec, n.vec=n.vec)
     lp <- l.prior.fun(theta.vec)
     lout <- llik + lp
     return(lout)
    }

#takes transformed params as inputs
l.unpost.fun2 <- function(phi.vec, dose.vec, y.vec, n.vec) {
     theta.vec <- c(phi.vec[1], exp(phi.vec[2]), exp(phi.vec[3]))
     llik <- llik.fun(theta.vec, dose.vec=dose.vec, y.vec=y.vec, n.vec=n.vec)
     lp <- l.prior.fun2(phi.vec)
     lout <- llik + lp
     return(lout)
    }
```

```
#find mode and variance covariance matrix for phi1, phi2, phi3
   optim.out <- optim(c(1.77, log(0.03), log(0.35)), l.unpost.fun2, dose.vec=dose,
                      y.vec=killed, n.vec=exposed, control=list(fnscale=-100),
                      method="Nelder-Mead", hessian=TRUE)
```

```
   mode <- optim.out$par  #MODE [1]  1.812353 -4.010371 -1.056079

   #estimated variance covariance matrix of parameter vector
   VarCov <- solve(-optim.out$hessian)
```

```
### use mode as mean of jumping distribution
### use 2*VarCov as variance covariance matrix

set.seed(25)

   nsim <- 1000  #number of iteration
   phi.mat1 <- matrix(NA, nrow=nsim, ncol=3) #chain 1
   phi.mat2 <- matrix(NA, nrow=nsim, ncol=3) #chain 2
   phi.mat3 <- matrix(NA, nrow=nsim, ncol=3) #chain 3
   jump.vec1 <- numeric(nsim-1) #keep track of when we jump (accept candidates)
   jump.vec2 <- numeric(nsim-1)
   jump.vec3 <- numeric(nsim-1)
   phi.mat1[1,] <- c(1.8, -4, -1)  #1.543270 -4.874497  2.179276
   phi.mat2[1,] <- c(1.5, -3.8, -1.2)
   phi.mat3[1,] <- c(1.8, -4.5,  -0.8)

   for (i in 2:nsim) {
     phi.cur1 <- phi.mat1[i-1,]
     phi.cur2 <- phi.mat2[i-1,]
     phi.cur3 <- phi.mat3[i-1,]
     phi.cand1 <- rmvnorm(1, mean=mode, sigma=2*VarCov)
     phi.cand2 <- rmvnorm(1, mean=mode, sigma=2*VarCov)
     phi.cand3 <- rmvnorm(1, mean=mode, sigma=2*VarCov)

     log.r.num1 <- l.unpost.fun2(phi.cand1, dose.vec=dose, y.vec=killed, n.vec=exposed) +
                   dmvnorm(phi.cur1, mean=phi.cand1, sigma=2*VarCov, log=TRUE)
     log.r.num2 <- l.unpost.fun2(phi.cand2, dose.vec=dose, y.vec=killed, n.vec=exposed) +
                   dmvnorm(phi.cur2, mean=phi.cand2, sigma=2*VarCov, log=TRUE)
     log.r.num3 <- l.unpost.fun2(phi.cand3, dose.vec=dose, y.vec=killed, n.vec=exposed) +
                   dmvnorm(phi.cur3, mean=phi.cand3, sigma=2*VarCov, log=TRUE)

     log.r.denom1 <- l.unpost.fun2(phi.cur1, dose.vec=dose, y.vec=killed, n.vec=exposed) +
                   dmvnorm(phi.cand1, mean=phi.cur1, sigma=2*VarCov, log=TRUE)
     log.r.denom2 <- l.unpost.fun2(phi.cur2, dose.vec=dose, y.vec=killed, n.vec=exposed) +
                   dmvnorm(phi.cand2, mean=phi.cur2, sigma=2*VarCov, log=TRUE)
     log.r.denom3 <- l.unpost.fun2(phi.cur3, dose.vec=dose, y.vec=killed, n.vec=exposed) +
                   dmvnorm(phi.cand3, mean=phi.cur3, sigma=2*VarCov, log=TRUE)

     log.r1 <- log.r.num1 - log.r.denom1
     log.r2 <- log.r.num2 - log.r.denom2
     log.r3 <- log.r.num3 - log.r.denom3

     p.accept1 <- min(1, exp(log.r1))
     p.accept2 <- min(1, exp(log.r2))
     p.accept3 <- min(1, exp(log.r3))

     u.vec <- runif(3)
```

```
     ifelse(u.vec[1] <= p.accept1, phi.mat1[i,]<- phi.cand1, phi.mat1[i,] <- phi.cur1)
     ifelse(u.vec[2] <= p.accept2, phi.mat2[i,]<- phi.cand2, phi.mat2[i,] <- phi.cur2)
     ifelse(u.vec[3] <= p.accept3, phi.mat3[i,]<- phi.cand3, phi.mat3[i,] <- phi.cur3)

     jump.vec1[i-1] <- ifelse(u.vec[1] <= p.accept1, 1, 0)
     jump.vec2[i-1] <- ifelse(u.vec[2] <= p.accept2, 1, 0)
     jump.vec3[i-1] <- ifelse(u.vec[3] <= p.accept3, 1, 0)
    }
```

```
   ##Acceptance rates?
     mean(jump.vec1)
     mean(jump.vec2)
     mean(jump.vec3)

   ## Use coda to help
    library(coda)
    phi.post1 <- mcmc(phi.mat1)
    phi.post2 <- mcmc(phi.mat2)
    phi.post3 <- mcmc(phi.mat3)
#use to put all draws from the 3 chains in one list
    phi.mcmc <- mcmc.list(list(phi.post1, phi.post2, phi.post3))
    summary(phi.mcmc)
    plot(phi.mcmc)
```

```
library(coda)
require(xtable)
    #Look at some other convergence diagnostics
    eff <- effectiveSize(phi.mcmc)
    gd <- gelman.diag(phi.mcmc)  #Can't do b/c need at least 2 chains
    #gelman.plot(phi.mcmc)
    #geweke.diag(phi.mcmc)
    #heidel.diag(phi.mcmc)
    #raftery.diag(phi.mcmc)
table <- cbind(gd$psrf, eff)
names(table) <- c("Rhat", "Upper CI Rhat", "Eff")
xtable(table)
```

```
   ##Look at chains
     plot(seq(1:nsim), phi.mat1[,1], type="l", ylab=expression(phi[1]), col=4)
        lines(seq(1:nsim), phi.mat2[,1], col=2)
        lines(seq(1:nsim), phi.mat3[,1], col=3)
     plot(seq(1:nsim), phi.mat1[,2], type="l", ylab=expression(phi[2]), col=4)
        lines(seq(1:nsim), phi.mat2[,2], col=2)
        lines(seq(1:nsim), phi.mat3[,2], col=3)
     plot(seq(1:nsim), phi.mat1[,3], type="l", ylab=expression(phi[3]), col=4)
        lines(seq(1:nsim), phi.mat2[,3], col=2)
        lines(seq(1:nsim), phi.mat3[,3], col=3)
```

```
par(mfrow=c(1,3))

burnin <- 100

mu.draws <- c(phi.mat1[,1], phi.mat2[(burnin+1):nsim,1],
              phi.mat3[(burnin+1):nsim,1])
sigma.draws <- sqrt(exp(2*c(phi.mat1[(burnin+1):nsim,2],
                            phi.mat2[(burnin+1):nsim,2],
                            phi.mat3[(burnin+1):nsim,2])))
m1.draws <- exp(c(phi.mat1[(burnin+1):nsim,3], phi.mat2[(burnin+1):nsim,3],
                  phi.mat3[(burnin+1):nsim,3]))

hist(mu.draws, nclass=40, xlab=expression(mu), freq=FALSE,
     main = expression(mu))

hist(sigma.draws, nclass=40, xlab=expression(sigma), freq=FALSE,
     main = expression(sigma))

hist(m1.draws, nclass=40, xlab=expression(m[1]), freq=FALSE,
     main = expression(m[1]))
```

```
require(xtable)
xtable(summary(phi.mcmc)$statistics)
xtable(summary(phi.mcmc)$quantiles)
```

```
   ##Look at chains
     plot(seq(1:nsim), phi.mat[,1,1], type="l", ylab=expression(phi[1]))
        lines(seq(1:nsim), phi.mat[,1,2], col=2)
        lines(seq(1:nsim), phi.mat[,1,3], col=3)
   plot(seq(1:nsim), phi.mat[,2,1], type="l", ylab=expression(phi[2]))
        lines(seq(1:nsim), phi.mat[,2,2], col=2)
        lines(seq(1:nsim), phi.mat[,2,3], col=3)
    plot(seq(1:nsim), phi.mat[,3,1], type="l", ylab=expression(phi[3]))
        lines(seq(1:nsim), phi.mat[,3,2], col=2)
        lines(seq(1:nsim), phi.mat[,3,3], col=3)
```

```
zoom <- 4000
##Look at chains
     plot(seq(1:zoom), phi.mat[1:zoom,1,1], type="l", ylab=expression(phi[1]))
        lines(seq(1:zoom), phi.mat[1:zoom,1,2], col=2)
        lines(seq(1:zoom), phi.mat[1:zoom,1,3], col=3)
   plot(seq(1:zoom), phi.mat[1:zoom,2,1], type="l", ylab=expression(phi[2]))
        lines(seq(1:zoom), phi.mat[1:zoom,2,2], col=2)
        lines(seq(1:zoom), phi.mat[1:zoom,2,3], col=3)
    plot(seq(1:zoom), phi.mat[1:zoom,3,1], type="l", ylab=expression(phi[3]))
        lines(seq(1:zoom), phi.mat[1:zoom,3,2], col=2)
        lines(seq(1:zoom), phi.mat[1:zoom,3,3], col=3)
```

```
mean(jump.vec1)
mean(jump.vec2)
mean(jump.vec3)
```

```
burnin <- 500
    phi1.post <- mcmc(phi.mat[(burnin+1):nsim,1:3,1])
    phi2.post <- mcmc(phi.mat[(burnin+1):nsim,1:3,2])
    phi3.post <- mcmc(phi.mat[(burnin+1):nsim,1:3,3])
#use to put all draws from the 3 chains in one list
    phi.mcmc <- mcmc.list(list(phi1.post, phi2.post, phi3.post))

library(coda)
require(xtable)
    #Look at some other convergence diagnostics
    eff2 <- effectiveSize(phi.mcmc)
    gd2 <- gelman.diag(phi.mcmc)  #Can't do b/c need at least 2 chains
    #gelman.plot(phi.mcmc)
    #geweke.diag(phi.mcmc)
    #heidel.diag(phi.mcmc)
   # raftery.diag
table2 <- cbind(gd2$psrf, eff2)
names(table2) <- c("Rhat", "Upper CI Rhat", "Eff")
xtable(table2)
```

```
par(mfrow=c(1,3))

mu.draws <- phi.mat[,1,]
sigma.draws <- sqrt(exp(2*phi.mat[,2,]))
m1.draws <- exp(phi.mat[,3,])

hist(c(mu.draws[(burnin+1):10000,1:3]), nclass=50, xlab=expression(mu),
     freq=FALSE, main = expression(mu))

hist(c(sigma.draws[(burnin+1):10000, 1:3]), nclass=60, xlab=expression(sigma),
     freq=FALSE, main = expression(sigma))

hist(c(m1.draws[(burnin+1):10000, 1:3]), nclass=60, xlab=expression(m[1]),
     freq=FALSE, main = expression(m[1]))
```

```
require(R2jags)
draws <- as.mcmc(beetles)
names(draws) <- c("chain1", "chain2", "chain3")
#traceplot(beetles, mfrow=c(1,2), varname=c("phi1", "phi2", "phi3"), ask=FALSE)
par(mfrow=c(1,2))
plot(1:50000, draws$chain1[1:50000,4], main=expression(phi[1]), type="l",
     ylim=c(1.76, 1.87))
lines(1:50000, draws$chain2[1:50000,4], col="red")
lines(1:50000, draws$chain3[1:50000,4], col="green")

plot(1:50000, draws$chain1[1:50000,5], main=expression(phi[2]), type="l",
```

```
     ylim=c(-5.8, -3.2))
lines(1:50000, draws$chain2[1:50000,5], col="red")
lines(1:50000, draws$chain3[1:50000,5], col="green")

plot(1:50000, draws$chain1[1:50000,6], main=expression(phi[3]), type="l",
     ylim=c(-3.5, 0))
lines(1:50000, draws$chain2[1:50000,6], col="red")
lines(1:50000, draws$chain3[1:50000,6], col="green")
```

```
par(mfrow=c(1,2))
plot(1:4000, draws$chain1[1:4000,4], main=expression(phi[1]), type="l",
     ylim=c(1.76, 1.87))
lines(1:4000, draws$chain2[1:4000,4], col="red")
lines(1:4000, draws$chain3[1:4000,4], col="green")

plot(1:4000, draws$chain1[1:4000,5], main=expression(phi[2]), type="l",
     ylim=c(-5.8, -3.2))
lines(1:4000, draws$chain2[1:4000,5], col="red")
lines(1:4000, draws$chain3[1:4000,5], col="green")

plot(1:4000, draws$chain1[1:4000,6], main=expression(phi[3]), type="l",
     ylim=c(-3.5, 0))
lines(1:4000, draws$chain2[1:4000,6], col="red")
lines(1:4000, draws$chain3[1:4000,6], col="green")
```

```
library(R2jags)
beetles <-jags(model.file="jags-beetles.jags", data = data,
               parameters.to.save = c("mu", "m1", "sigma", "phi1", "phi2", "phi3"),
               n.chains=3, n.thin=1, inits=inits, n.burnin=2000, n.iter=50000)
```

```
require(xtable)
xtable(beetles$BUGSoutput$summary)
```

```
draws <- as.mcmc(beetles)
par(mfrow=c(1,3))
names(draws) <- c("chain1", "chain2", "chain3")
hist(as.matrix(draws)[,2], main=expression(m[1]), xlab=expression(m1), freq=FALSE, nclass=70)
hist(as.matrix(draws)[,3], main=expression(mu), xlab=expression(mu), freq=FALSE, nclass=50)
hist(as.matrix(draws)[,7], main=expression(sigma), xlab=expression(sigma), freq=FALSE, nclass=50)
```