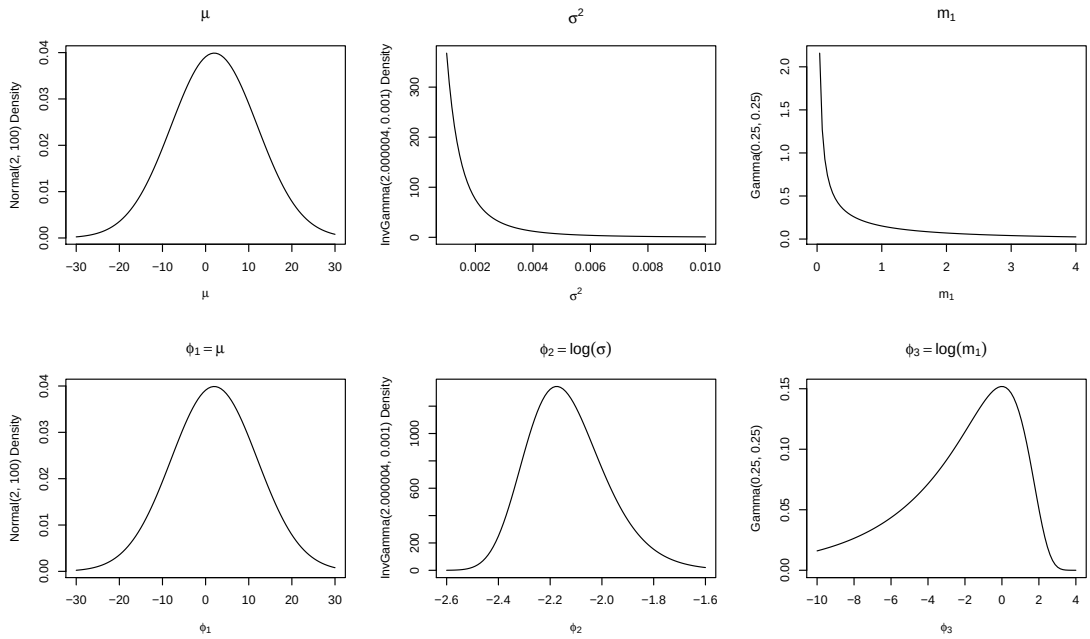
Stat 532 Assignment 7

Kenny Flagg

October 19, 2015

1. (a) For completeness, I plotted the priors on both the original scale and the transformed scale.



I modified the multivariate Metropolis-Hastings sampler into an independence chain. I reused the same code to numerically find the posterior mode and variance-covariance matrix, and then used a multivariate normal proposal distribution with a mean at the posterior mode and a variance-covariance matrix of two times the posterior variance-covariance matrix.

I chose initial values of

$$(\phi_1, \phi_2, \phi_3) = (1.8, -3, -2.5),$$
$$(\phi_1, \phi_2, \phi_3) = (1.7, -3.8, -0.2),$$
$$(\phi_1, \phi_2, \phi_3) = (1.9, -5, 0.3)$$

because these were outside of the center of the posterior distribution.

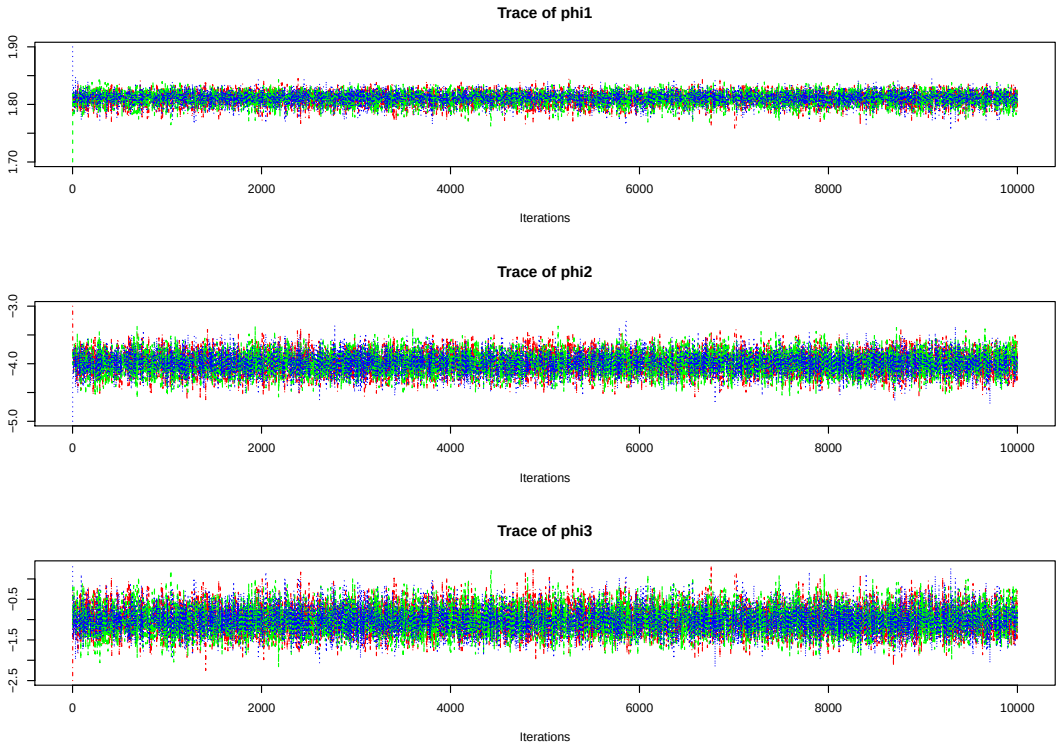
I renamed many of the functions and variables to fit my coding style, and I created a loop to go through the three chains. Since the original multivariate M-H sampler ran

successfully, I started out by running the independence chain for 10,000 iterations. The code occupies the next two pages. Traceplots follow on page 4.

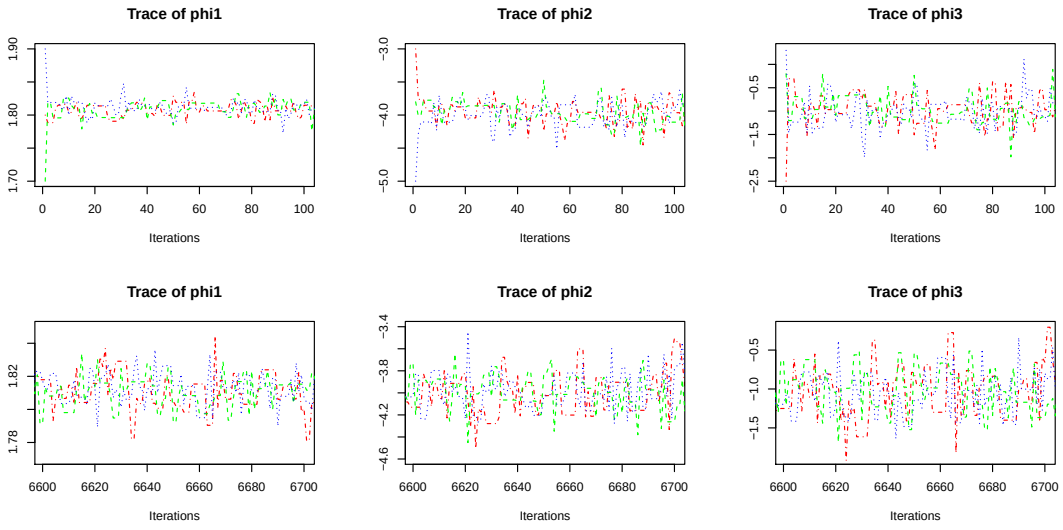
```
# The data
dose <- c(1.6907, 1.7242, 1.7552, 1.7842, 1.8113, 1.8369, 1.8610, 1.8839)
killed <- c(6, 13, 18, 28, 52, 53, 61, 60)
exposed <- c(59,60,62,56,63,59,62,60)

# Log likelihood for phi=(mu, log(sig2), log(m1)) given the data
l.lik <- function(phi, dose, y, n){
  mu <- phi[1]
  sig <- exp(phi[2])
  m1 <- exp(phi[3])
  x <- (dose - mu) / sig
  llik <- m1*y*(x-log(1+exp(x))) + (n-y)*log(1-((exp(x)/(1+exp(x)))^m1))
  out <- sum(llik)
  return(out)
}

# Transformed prior
# I worked out the transformed distributions on paper and got the same result
l.prior <- function(phi, a0 = 0.25, b0 = 0.25, c0 = 2,
                    d0 = 10, e0 = 2.000004, f0 = 0.001){
  log.p.phi1 <- dnorm(phi[1], mean = c0, sd = d0, log = TRUE)
  log.p.phi2 <- -2*e0*phi[2] - f0*(exp(-2*phi[2]))
  log.p.phi3 <- phi[3]*a0 - b0*exp(phi[3])
  log.p <- log.p.phi1 + log.p.phi2 + log.p.phi3 #assuming priors independent
  return(log.p)
}

# Log unnormalized posterior
l.unpost <- function(phi, dose, y, n){
  llik <- l.lik(phi, dose = dose, y = y, n = n)
  lprior <- l.prior(phi)
  out <- llik + lprior
  return(out)
}

# Numerically find the posterior mode
optim.out <- optim(c(1.77, 0.03, 0.35), l.unpost,
                   dose = dose, y = killed, n = exposed,
                   control = list(fnscale=-100),
                   method = 'Nelder-Mead', hessian = TRUE)

# Use the posterior mode and twice the variance-covariance matrix as
# the parameters of the MVN propasal distribution
center <- optim.out$par
sig.matrix <- 2*solve(-optim.out$hessian)

set.seed(457234)

nsims <- 10000
nchains <- 3

# Create a list of empty matrices
phi.chains <- replicate(nchains,
```

```
                        matrix(nrow=nsims, ncol=3,
                               dimnames = list(NULL, c('phi1','phi2','phi3'))),
                        simplify=FALSE)

# Set inital values
phi.chains[[1]][1,] <- c(1.8, -3, -2.5)
phi.chains[[2]][1,] <- c(1.7, -3.8, -0.2)
phi.chains[[3]][1,] <- c(1.9, -5,  0.3)

# Keep track of the jumps
jumps <- matrix(nrow = nsims-1, ncol = nchains)

# Loop for iterations
for (i in 2:nsims){
  # Loop for chains
  for(j in 1:nchains){
    # Get a candidate
    phi.cand <- rmvnorm(1, mean=center, sigma=sig.matrix)

    # Ratio numerator
    log.r.num <- l.unpost(phi.cand, dose=dose, y=killed, n=exposed) +
      dmvnorm(phi.chains[[j]][i-1,], mean=center, sigma=sig.matrix, log=TRUE)

    # Ratio denominator
    log.r.denom <- l.unpost(phi.chains[[j]][i-1,],
                            dose=dose, y=killed, n=exposed) +
      dmvnorm(phi.cand, mean=center, sigma=sig.matrix, log=TRUE)

    # Ratio
    log.r <- log.r.num - log.r.denom

    # Accept with probability min(1, r)
    u <- runif(1)
    if(u<=exp(log.r)){
      # Accept the candidate and note that we jumped
      phi.chains[[j]][i,] <- phi.cand
      jumps[i-1,j] <- 1
    }else{
      # Do not accept the candidate and note that we did not jump
      phi.chains[[j]][i,] <- phi.chains[[j]][i-1,]
      jumps[i-1,j] <- 0
    }
  }
}
```

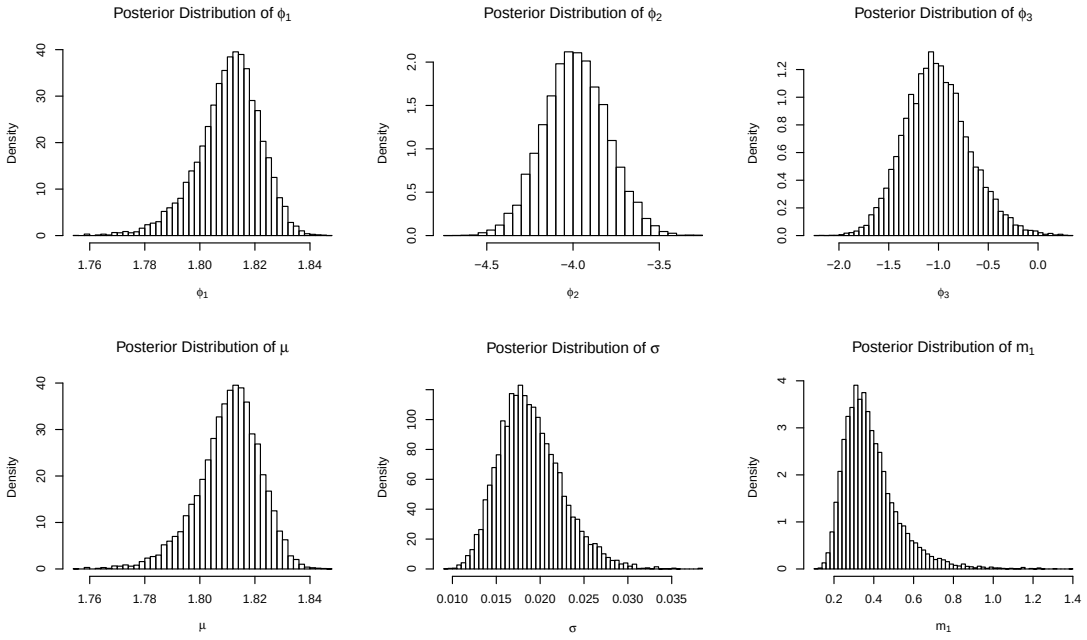
The traceplots of the full sequences of 10,000 iterations show that the chains immediately began to mix together in one region on the parameter space. Otherwise, they are not very informative.



The next set of traceplots show the first hundred iterations, and another hundred iterations from later in the chains. All three chains appear to have converged within 20 iterations. The independence of the candidates is apparent since the plots show many large jumps and little oscillation.



The chains had a multivariate PSRF of 1.0007 which is evidence that that chains converged. The effective sample sizes were computed as $n_{\phi_1} = 11353.5$, $n_{\phi_2} = 13213.2$, and $n_{\phi_3} = 12830.9$, which do not surprise me given that the proposals were independent. Because of the traceplots, I decided to discard the first 20 draws as burn-in.
The distributions of the 9,980 posterior draws appear below.



(b) To construct a Gibbs sampler with Metropolis-Hastings draws, I first found the complete conditional distributions and then modified the independence chain. Since the priors are independent, the complete conditional distributions simplify to

$$p(\mu|\sigma^2, m_1, x, y) \propto \prod_{i=1}^{8} \text{Binomial}\left(y_i|n_i\left(\frac{e^{x_i}}{1+e^{x_i}}\right)^{m_1}\right) \times \text{N}(\mu|c_0, d_0^2),$$

$$p(\sigma^2|\mu, m_1, x, y) \propto \prod_{i=1}^{8} \text{Binomial}\left(y_i|n_i\left(\frac{e^{x_i}}{1+e^{x_i}}\right)^{m_1}\right) \times \text{InvGamma}(\sigma^2|e_0, f_0),$$

$$p(m_1|\mu, \sigma^2, x, y) \propto \prod_{i=1}^{8} \text{Binomial}\left(y_i|n_i\left(\frac{e^{x_i}}{1+e^{x_i}}\right)^{m_1}\right) \times \text{Gamma}(m_1|a_0, b_0).$$

I used the same starting values as before. I centered the proposal distribution at the current value of the parameter, and used the posterior variance as the variance of the proposal distribution. In each iteration, I drew a candidate ϕ_1, accepted or rejected it based on the M-H ratio, and repeated for ϕ_2 and ϕ_3. I ran three chains for 20,000 iterations. The code begins below, and plots appear on pages 8 and 9.

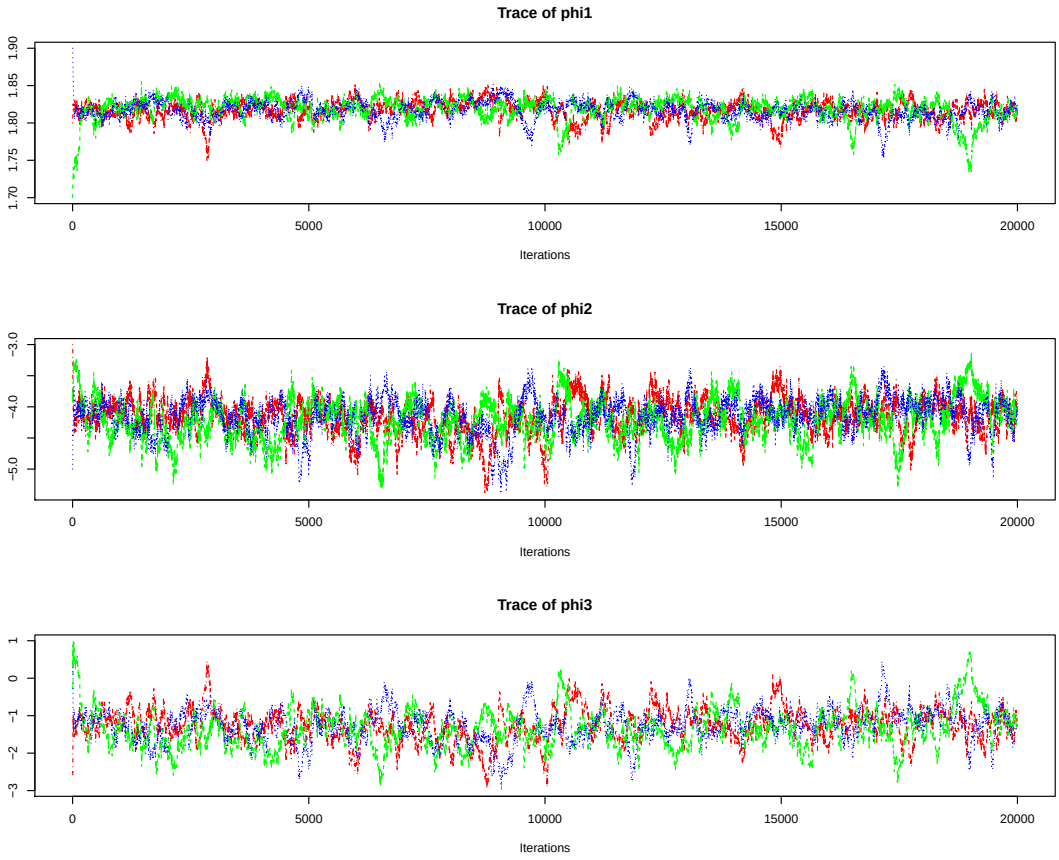
```
# Complete conditional log densities of transformed parameters
l.phi1.cond <- function(phi, c0 = 2, d0 = 10, dose, y, n){
  log.lik <- l.lik(phi, dose = dose, y = y, n = n)
  log.p.phi1 <- dnorm(phi[1], mean = c0, sd = d0, log = TRUE)
```

```
  out <- log.lik + log.p.phi1
  return(out)
}
l.phi2.cond <- function(phi, ph3, e0 = 2.000004, f0 = 0.001, dose, y, n){
  log.lik <- l.lik(phi, dose = dose, y = y, n = n)
  log.p.phi2 <- -2*e0*phi[2] - f0*(exp(-2*phi[2]))
  out <- log.lik + log.p.phi2
  return(out)
}
l.phi3.cond <- function(phi, a0 = 0.25, b0 = 0.25, dose, y, n){
  log.lik <- l.lik(phi, dose = dose, y = y, n = n)
  log.p.phi3 <- phi[3]*a0 - b0*exp(phi[3])
  out <- log.lik + log.p.phi3
  return(out)
}

# sig.matrix is 2 times the posterior variance
sig.gibbs <- sqrt(diag(sig.matrix)*0.5)

set.seed(54673)

nsims <- 20000
nchains <- 3

# Create a list of empty matrices
phi.gibbs <- replicate(nchains,
                       matrix(nrow=nsims, ncol=3,
                              dimnames = list(NULL, c('phi1','phi2','phi3'))),
                       simplify=FALSE)

# Set inital values
phi.gibbs[[1]][1,] <- c(1.8, -3, -2.5)
phi.gibbs[[2]][1,] <- c(1.7, -3.8, -0.2)
phi.gibbs[[3]][1,] <- c(1.9, -5,  0.3)

jumps2 <- matrix(nrow = nsims-1, ncol = nchains)

# Loop for iterations
for (i in 2:nsims){
  # Loop for chains
  for(j in 1:nchains){
    # Get a candidate phi1
    phi.cand <- phi.gibbs[[j]][i-1,]
    phi.cand[1] <- rnorm(1, mean=phi.gibbs[[j]][i-1,1],
                         sd=sig.gibbs[1])

    # Numerator with phi1
    log.r.num <- l.phi1.cond(phi.cand, dose=dose, y=killed, n=exposed) +
      dnorm(phi.gibbs[[j]][i-1,1], mean=phi.cand[1],
            sd=sig.gibbs[1], log=TRUE)

    # Denominator with phi2
    log.r.denom <- l.phi1.cond(phi.gibbs[[j]][i-1,],
                               dose=dose, y=killed, n=exposed) +
      dnorm(phi.cand[1], mean=phi.gibbs[[j]][i-1,1],
```

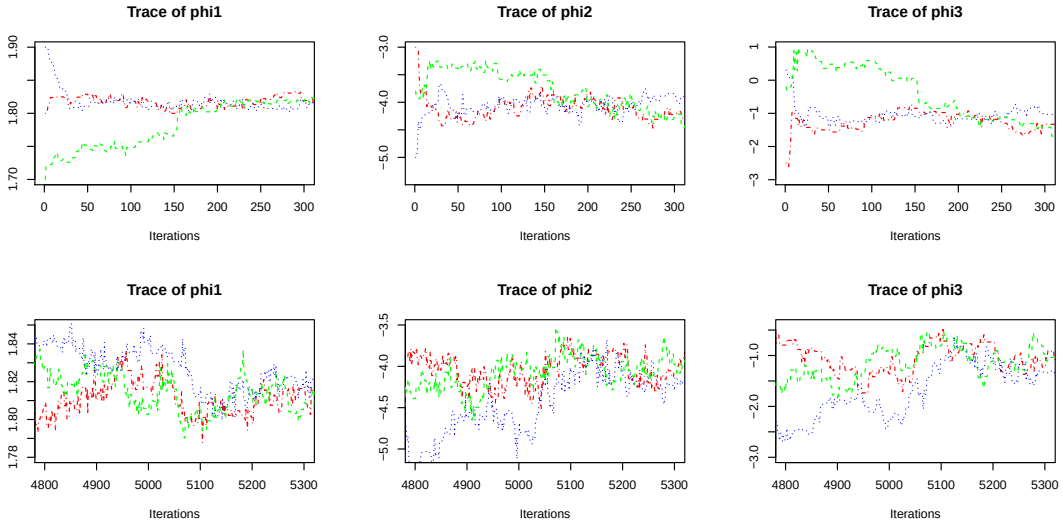
```
        sd=sig.gibbs[1], log=TRUE)

# Ratio
log.r <- log.r.num - log.r.denom

# Accept with probability min(1, r)
u <- runif(1)
if(u<=exp(log.r)){
  phi.gibbs[[j]][i,1] <- phi.cand[1]
  jumps2[i-1, 1] <- 1
}else{
  phi.gibbs[[j]][i,1] <- phi.gibbs[[j]][i-1,1]
  phi.cand[1] <- phi.gibbs[[j]][i-1,1]
  jumps2[i-1, 1] <- 0
}

# Get a candidate phi2
phi.cand[2] <- rnorm(1, mean=phi.gibbs[[j]][i-1,2],
                     sd=sig.gibbs[2])

# Numerator with phi2
log.r.num <- l.phi1.cond(phi.cand, dose=dose, y=killed, n=exposed) +
  dnorm(phi.gibbs[[j]][i-1,2], mean=phi.cand[2],
        sd=sig.gibbs[2], log=TRUE)

# Denominator with phi2
log.r.denom <- l.phi1.cond(phi.gibbs[[j]][i-1,],
                           dose=dose, y=killed, n=exposed) +
  dnorm(phi.cand[2], mean=phi.gibbs[[j]][i-1,2],
        sd=sig.gibbs[2], log=TRUE)

# Ratio
log.r <- log.r.num - log.r.denom

# Accept with probability min(1, r)
u <- runif(1)
if(u<=exp(log.r)){
  phi.gibbs[[j]][i,2] <- phi.cand[2]
  jumps2[i-1, 2] <- 1
}else{
  phi.gibbs[[j]][i,2] <- phi.gibbs[[j]][i-1,2]
  phi.cand[2] <- phi.gibbs[[j]][i-1,2]
  jumps2[i-1, 2] <- 0
}

# Get a candidate phi3
phi.cand[3] <- rnorm(1, mean=phi.gibbs[[j]][i-1,3],
                     sd=sig.gibbs[3])

# Numerator with phi2
log.r.num <- l.phi1.cond(phi.cand, dose=dose, y=killed, n=exposed) +
  dnorm(phi.gibbs[[j]][i-1,3], mean=phi.cand[3],
        sd=sig.gibbs[3], log=TRUE)

# Denominator with phi2
```

```
    log.r.denom <- l.phi1.cond(phi.gibbs[[j]][i-1,],
                               dose=dose, y=killed, n=exposed) +
      dnorm(phi.cand[3], mean=phi.gibbs[[j]][i-1,3],
            sd=sig.gibbs[3], log=TRUE)

    # Ratio
    log.r <- log.r.num - log.r.denom

    # Accept with probability min(1, r)
    u <- runif(1)
    if(u<=exp(log.r)){
      phi.gibbs[[j]][i,3] <- phi.cand[3]
      jumps2[i-1, 3] <- 1
    }else{
      phi.gibbs[[j]][i,3] <- phi.gibbs[[j]][i-1,3]
      jumps2[i-1, 3] <- 0
    }
  }
}
```

The traceplots show a high level of autocorrelation. These chains appeared to reach extreme values more frequently than the independence chains did. However, the Gibbs chains did stay in the same area of the sample space.

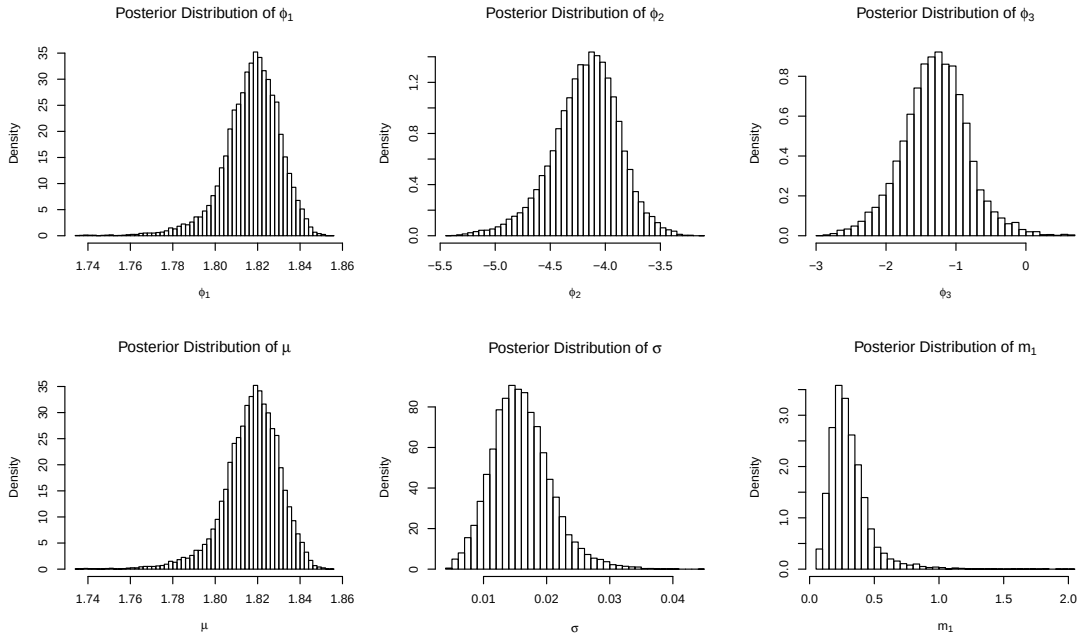


I created zoomed-in plots to further examine the chains. The first row of plots shows that two chains reached the center of the distribution in under 50 iterations, and the third joined them by the 200th iteration. The second illustrates that the draws were strongly correlated within each chain, but that the chains did slowly move about the parameter space.



The multivariate PSRF is 1.02 and the effective sample sizes are low because of the autocorrelation, with $n_{\phi_1} = 269.9$, $n_{\phi_2} = 265.8$, and $n_{\phi_3} = 222.1$. Since the traceplots show that each chain moved around the whole parameter space, it seemed like the algorithm was close to convergence and the results would be usable. I dropped the first 200 iterations as burn-in.

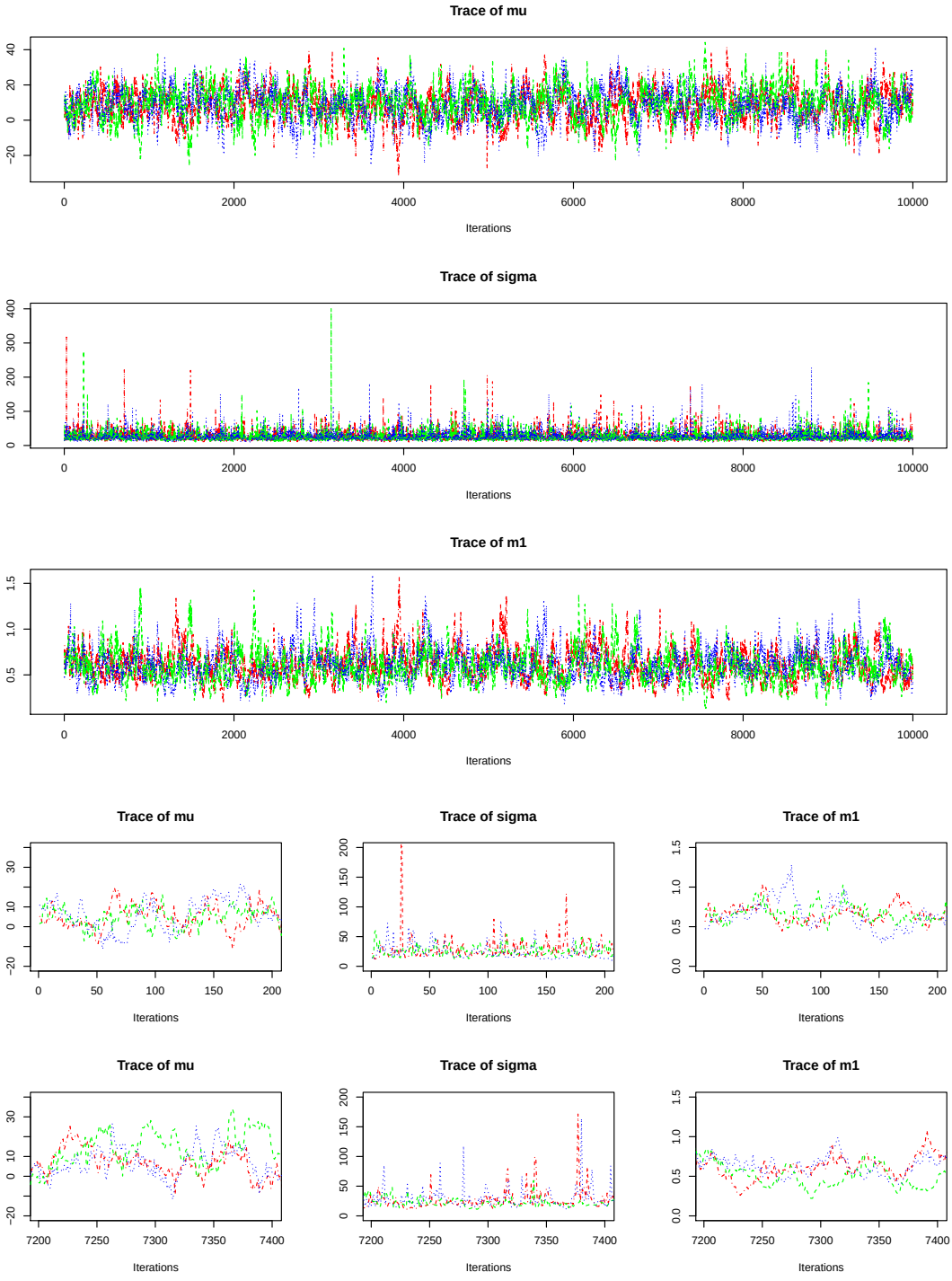
The following plots are based on 19,800 posterior draws.



(c) I constructed a JAGS model for the un-transformed parameters. I let JAGS choose the starting values and ran three chains for 10,000 iterations, instructing JAGS not to do any thinning or burn-in. The JAGS model appears below, and fits nicely on a single page.

```
# Variables:
#  m1     Dispersion (parameter of interest)
#    a0     Hyperparameter - specified in data argument
#    b0     Hyperparameter - specified in data argument
#  mu     Location (parameter of interest)
#    c0     Hyperparameter - specified in data argument
#    d0     Hyperparameter - specified in data argument
#  sigma  Scale (parameter of interest)
#  tau    Precision  tau = 1/sigma^2
#    e0     Hyperparameter - specified in data argument
#    f0     Hyperparameter - specified in data argument
#  p      Probability of death  p = (invlogit(x))^m1
#  y      Number killed (response) - specified in data argument
#  n      Number exposed - specified in data argument
#  w      Dose - specified in data argument
#  x      Standardized dose  x = (w - mu) / sigma
jags.model <- function(){
  # Loop for each row
  for(i in 1:length(y)){
    # Model
    y[i] ~ dbinom(p[i], n[i])
    p[i] <- pow(exp(x[i]) / (1+exp(x[i])), m1)
    x[i] <- (w[i] - mu) / sigma
  }

  # Priors
  m1 ~ dgamma(a0, b0)
  mu ~ dnorm(c0, pow(d0, -2))
  tau ~ dgamma(e0, f0)
  sigma <- 1/sqrt(tau)
}
jags.data <- list('y' = killed,
                  'n' = exposed,
                  'w' = dose,
                  'a0' = 0.25,
                  'b0' = 4,
                  'c0' = 2,
                  'd0' = 10,
                  'e0' = 2.000004,
                  'f0' = 1000)
jags.params <- c('m1', 'mu', 'sigma', 'p', 'x')
```

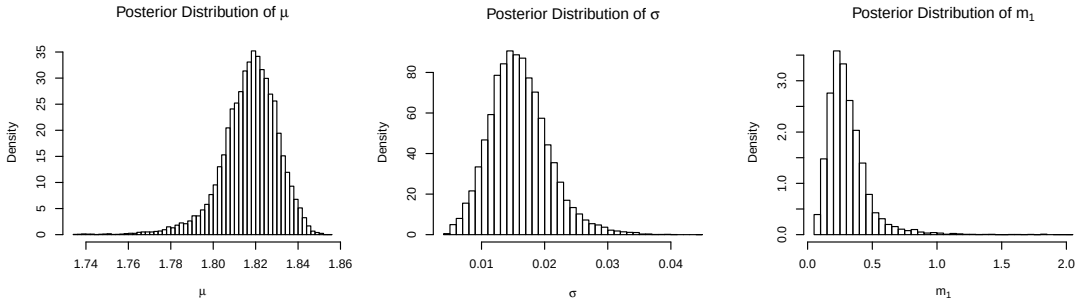
JAGS apparently chose starting values near the center of the posterior. From the plots on the following page, it looks like the algorithm was converged from the beginning. The beginnings of the sequences did not look any different from later sections, so I did not discard any draws.



The JAGS draws had a multivariate PSRF of 1.01 which suggests that the chains nearly converged, but this value may be inaccurate since the distributions of σ and m_1 are skewed. The effective sample sizes were $n_\mu = 781.2$, $n_\sigma = 6208.8$, and $n_{m_1} = 690.6$. I don't think any of these are problematically small, but I don't know why n_{σ^2} was so much larger than the others. The black-box gave nice results with little effort, but given

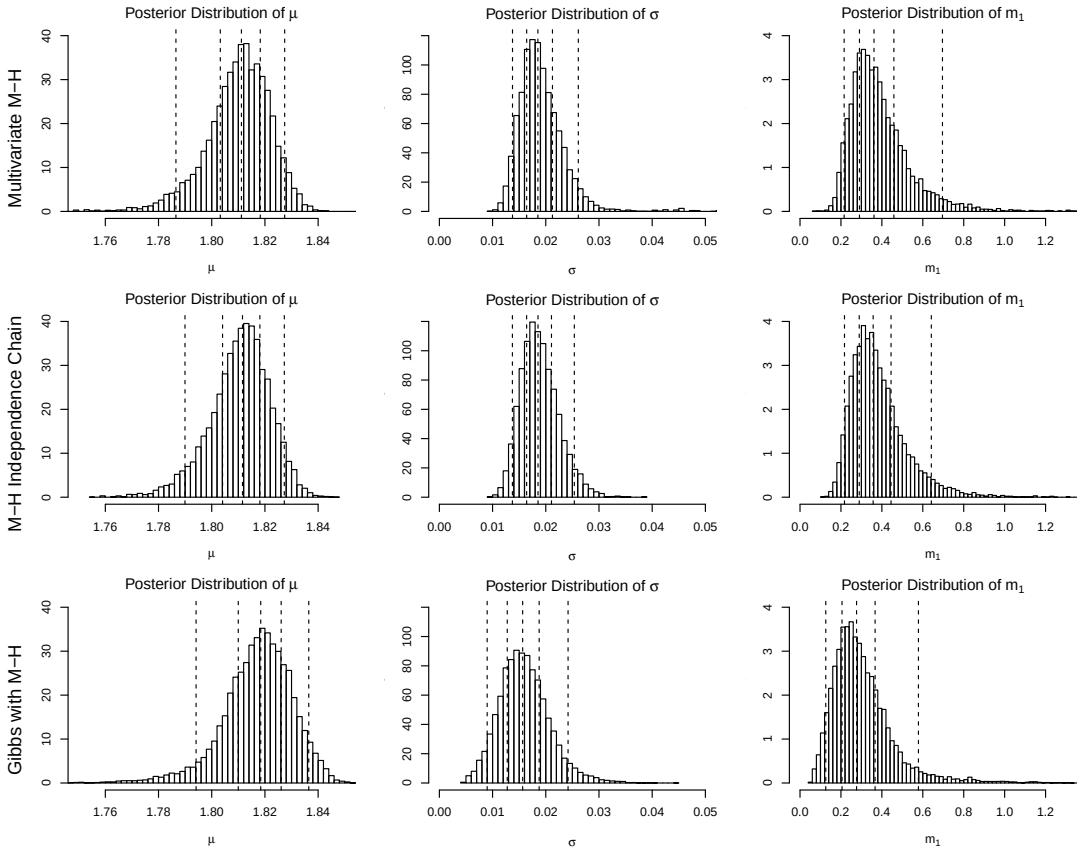
all the work I put into my Gibbs sampler for the previous part, I am curious what JAGS did behind-the-scenes.

The plots below are based on 10,000 posterior draws.



(d) For comparison, I re-ran the multivariate Metropolis-Hastings sampler with the same starting values that I used for the independence chain and the Gibbs sampler. I threw out 200 iterations of burn-in and kept 9,800 draws. The effective sample sizes were $n_{\phi_1} = 1193.8$, $n_{\phi_2} = 1901.8$, and $n_{\phi_3} = 1541.3$.

The plots on this page show the results from all three algorithms, with vertical lines marking the 5th, 25th, 50th, 75th, and 95th percentiles. All three samplers produced very similar posterior distributions.



For further comparison, I have tabulated the effective sample sizes and acceptance rates.

Algorithm	Draws	n_μ	n_σ	n_{m_1}	μ Rate	σ Rate	m_1 Rate
Metropolis-Hastings	10,000	1193.8	1901.8	1541.3	0.299	0.306	0.311
Independence Chain	10,000	11353.5	13213.2	12830.9	0.582	0.577	0.577
Gibbs with M-H	20,000	269.9	265.8	222.1	0.394	0.471	0.298

All three algorithms have strengths and weaknesses that make them useful for different situations. There is no single one that is best for everything.

The Metropolis-Hastings algorithm has the strength of being versatile and easy to understand. I imagine the chain walking through the parameter space, and the proposal distribution controls how it looks around the vicinity of its current location. It is easy to choose a reasonable proposal distribution and then tune it to work very well. The only downside is that some effort must be put into the tuning process.

In this example, the independence chain was certainly the most efficient. It converged almost immediately and had the highest acceptance rates. Since the candidates are independent of the chain, there is very little autocorrelation, so it moves around the parameter space quickly. As a result, a small sample size is acceptable. It does have one major weakness in that the proposal distribution must be chosen very carefully. I think a big part of the reason it did well here is that quite a bit of effort went into studying the target distribution, and creating a proposal distribution that approximated the proposal distribution near its mode. I see a problem in that the chain will not thoroughly cover the space because the candidates will keep pulling it back toward the mode. If the target distribution is multimodal or has an irregular shape, I would not expect an independence chain to yield accurate results.

The biggest advantage of a Gibbs sampler is that, in the algorithm's purest form, a proposal distribution is not necessary. When all of the complete conditional distributions can be easily sampled from, a Gibbs sampler would be quick and easy to program. However, it has many drawbacks. Since the draws come directly from the complete conditional distributions, they are highly autocorrelated. As a result, the sampler can be slow to explore the parameter space, so more draws may be needed than when using other algorithms. Additionally, when another method of sampling is be nested within the Gibbs sampler, I find the problem to be conceptually confusing. In that case, proposal distributions must be chosen, and each iteration requires a candidate for each parameter to be drawn and either accepted or rejected. At that point, it seems more straightforward to use a Metropolis-Hastings sampler.

The independence chain and Gibbs sampler are each suited to a certain type of problem. The independence chain performs excellently when the target distribution is well-understood or can be approximated. Gibbs seems most useful when there are many parameters so that the joint target distribution is complicated, but the complete conditionals are easy to work with. Metropolis-Hastings is a general-purpose fall-back for other situations.