Lecture 5

Implementing Bayesian models: Introduction to MCMC samplers

WILD6900 (Spring 2020)

Readings

> Hobbs & Hooten 145-180

From the joint distribution to the posterior

Remember from lecture 3 that the posterior distribution of random variable θ conditional on data y is defined by Bayes theorem:

$$\underbrace{[\theta|y]}_{posterior\ distribution} = \frac{\overbrace{[y|\theta]}^{likelihood} \quad \overbrace{[\theta]}^{prior}}{\underbrace{[y]}_{marginal\ distribution}}$$

We spent a fair about of time learning how to define the likelihood $[\theta, y]$ and the prior $[\theta]$

But we gave only given a cursory overview to the denominator of Bayes theorem, the marginal distribution $[y]$ [1]

Markov chain Monte Carlo

Markov chain Monte Carlo

In your own analyses, you will generally write out the model using a programming language understood by one of the common Bayesian software programs and then let the software fit the model

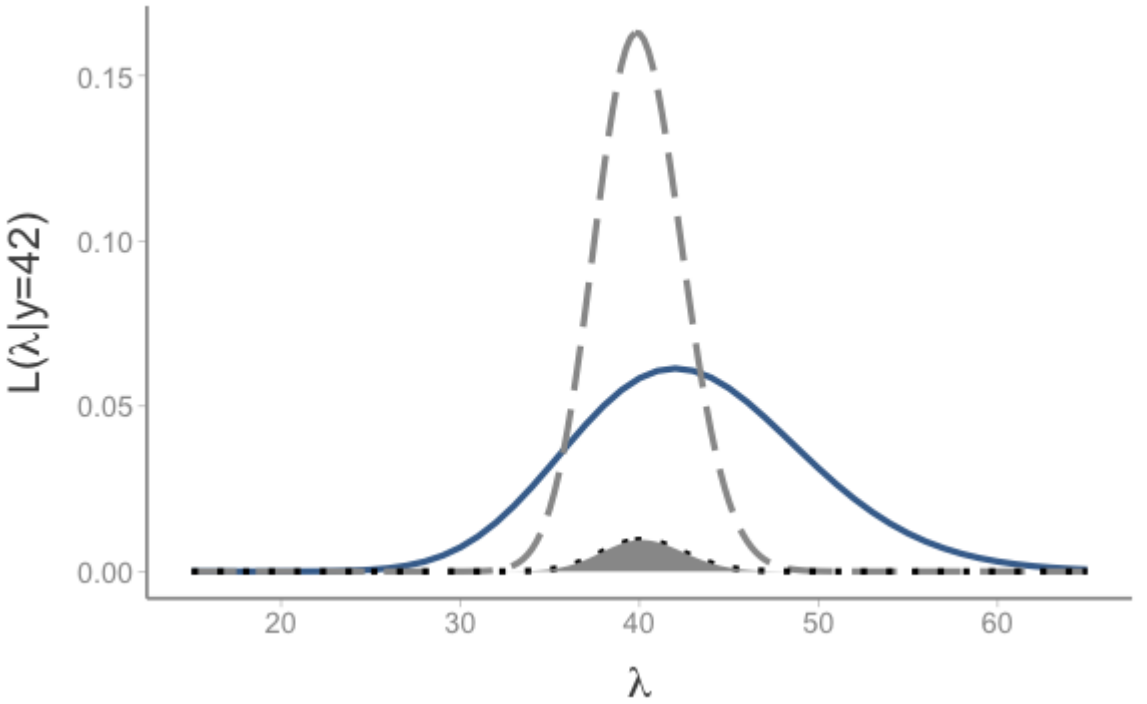
- This can be done without really knowing much of anything about *how* the software fits the model
- After this lecture, that's essentially the way we will operate for the remainder of the semester.

But treating these software as a total black box is not a great idea

- Having at least some idea what JAGS (or BUGS or Nimble) is doing to estimate the posterior distributions of parameters in your model will help you better understand if and when these programs are doing what you want [1]

From the joint distribution to the posterior

Remember that the marginal distribution $[y]$ is what normalizes the joint distribution to ensure the posterior is a proper probability distribution [1]



From the joint distribution to the posterior

Remember that the marginal distribution $[y]$ is what normalizes the joint distribution to ensure the posterior is a proper probability distribution [1]

- Without the marginal distribution, parameters cannot be treated as random variables [1]

This issue of estimating the marginal distribution is what limited the application of Bayesian methods to practical problems from it's inception until the 1990's

Progress was only made once statisticians started developing methods to learn about the posterior distribution by sampling *from the posterior distributions*

Learning about the posterior by sampling from it

How can we draw samples from a distribution that we don't know?

Remember from lecture 3 that we do know something about the distribution of each parameter in our model, namely the joint distribution:

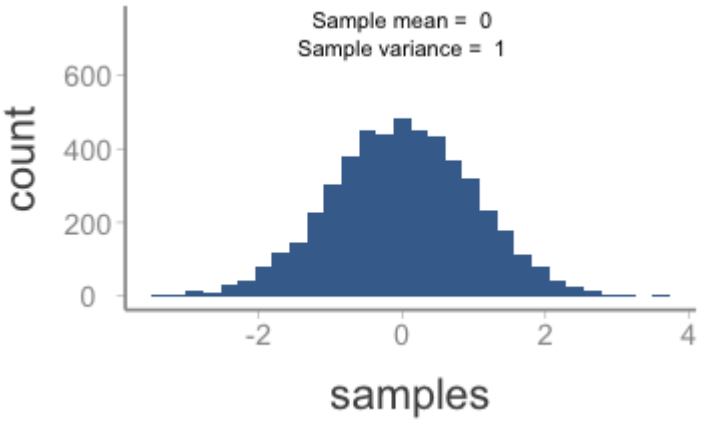
$$[\theta|y] \propto [y|\theta][\theta]$$

By taking many many samples from the joint distribution, we can learn about its shape

Learning about the posterior by sampling from it

If this seems confusing, think about learning about the shape of a probability distribution by taking many samples from it [1]

```
norm_df <- data.frame(samples = rnorm(5000))    # Generate samples fr
```



Learning about the posterior by sampling from it

In modern Bayesian analysis, estimating posterior distributions is done using *Markov chain Monte Carlo* (MCMC) methods

MCMC is an algorithm that uses the joint distribution to to sample values of each random variable in proportion to their to their probability

Essentially, MCMC takes the likelihood profile [1], weights it by the prior [2], and then normalizes the joint distribution so it is a proper probability distribution

Implementing a simple MCMC

If that seems a little confusing, it should be come clear if go through the steps of the MCMC applied to a simple model with a single parameter θ

Let's return to our survival example where we tracked individuals using GPS collars and at the end of the study, recorded the number of individuals that were still alive

- We will expand the example to include 5 study sites and assume we tracked 20 individuals at each site

We want to estimate ϕ, the probability of survival from the beginning to the end of the study [1]

Implementing a simple MCMC

Before we can implement the MCMC algorithm, we have to choose an appropriate likelihood distribution to describe our system

In this case, we want to know the probability of getting n "successes" (alive at the end of the study) out of N "tries" (individuals alive at the beginning of the study)

- A natural choice for this likelihood is the binomial distribution

Simulate data, assuming the true survival probability is 40%:

```
n.sites <- 5 # Number of sites
N <- 20      # Number of individuals tracked at each site
phi <- 0.4   # True survival probability

(n <- rbinom(n = n.sites, size = N, prob = phi))
```

```
## [1]  9  8 10  7  8
```

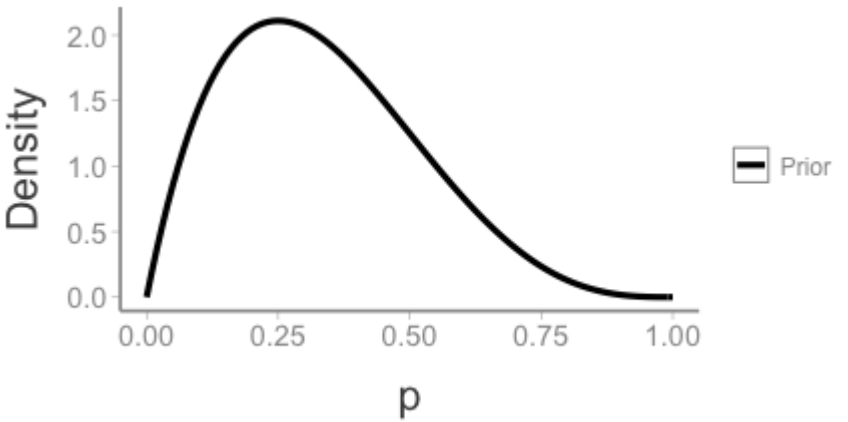
Implementing a simple MCMC

Next, we need to define the prior distribution

We could choose a uniform prior but as we have already learned, that's probably not the best choice

Still, let's assume we don't know much about the survival of our study organism other than that it's probably not less than ~20% and probably not more than ~50%

- A reasonable prior to represent this knowledge might be $beta(2, 4)$



Implementing a simple MCMC

To make our lives easier, let's also define a function to estimate the (log) joint distribution [1] conditional on our data and prior:

```
joint <- function(data, N, p, a, b){
  ll <- sum(dbinom(data, size = N, prob = p, log = TRUE)) +
        dbeta(x = p, shape1 = a, shape2 = b, log = TRUE)
  return(ll)
}
```

Make sure you understand exactly what this function is doing!

MCMC: basic steps

MCMC: basic steps

Once we have our data and defined a likelihood function and a prior distribution, the basics steps of the MCMC algorithm [1] are:

0) Choose an initial value for θ (call it θ^1)

1) Propose a new value θ^* from a proposal distribution

2) Compute probability of accepting θ^* using the joint distributions evaluated at θ^* and θ^{k-1}

3) Accept $\theta*$ (i.e., $\theta^k = \theta^*$) with the probability estimated in step 2, otherwise retain the previous value (i.e., $\theta^k = \theta^{k-1}$)

MCMC: basic steps

Steps 1-3 are repeated many times, resulting in K samples of θ

- This collection of θ values is referred to as a *chain*
- Steps 2 and 3 ensure that the values of θ is our chain are sampled in proportion to their probability in the posterior distribution $[\theta|y]$
- By collecting a large number of samples of θ is proportion to their probability in $[\theta|y]$, we can define the posterior distribution

MCMC: basic steps

To clarify what's happening in each of these steps, we'll work through each one using our example data

MCMC: basic steps

Step 0: Choose an initial value

The first step is to choose an initial value for the chain

The exact value typically isn't too important but should be consistent with the probability distribution that defines the parameter [1]

In some cases, it can be advantageous to choose initial values that are at least reasonable approximations of the parameter value (but for simple models it shouldn't matter much)

MCMC: basic steps

Step 0: Choose an initial value

For our example, we'll just choose a random number between 0 and 1 using `runif(1)`

- This will give different values each time we run the algorithm, which is a good way to investigate whether the posterior estimate is sensitive to the initial values [1]

```
set.seed(123456)
(init <- runif(1))
```

```
## [1] 0.7978
```

MCMC: basic steps

Step 1: Propose a new value

All MCMC algorithms develop chains by proposing a new value of θ (denoted θ^*) from some *proposal distribution*

There are different ways to formulate proposal distributions but in most applications, the proposal θ^* is *dependent* on θ^{k-1}

That is:

$$\left[\theta^* \middle| \theta^{k-1}\right]$$

For example, we could use a normal proposal distribution:

$$\theta^* \sim Normal(\theta^{k-1}, \sigma^2)$$

MCMC: basic steps

Step 1: Propose a new value

Using a normal proposal distribution, θ^* can be any real number but will, on average, tend to be close to θ^{k-1}

How close θ^* is to θ^{k-1} is determined by σ^2, which is referred to as the *tuning parameter*

- σ^2 determines the frequency that proposals are accepted
- Larger values will result in more frequent rejections, smaller values will result in more acceptances.

MCMC: basic steps

Step 1: Propose a new value

The normal distribution implies that θ can be any real number, which makes sense for some parameters but not for our example

We could instead use a beta distribution and use moment matching to define α and β in terms of θ^{k-1} (the mean) and σ^2:

$$\theta^* \sim beta\left(\theta^{k-1}\left(\frac{\theta^{k-1}(1-\theta^{k-1})}{\sigma^2}-1\right),(1-\theta^{k-1})\left(\frac{\theta^{k-1}(1-\theta^{k-1})}{\sigma^2}-1\right)\right)$$

MCMC: basic steps

Step 1: Propose a new value

Again, let's create a function to generate proposals:

```
proposal <- function(p, sigma2){
  alpha <- p * ((p * (1 - p) / sigma2) - 1)
  beta <- (1 - p) * ((p * (1 - p) / sigma2) - 1)

  proposal <- rbeta(n = 1, shape1 = alpha, shape2 = beta)
  return(proposal)
}
```

MCMC: basic steps

Step 2: Estimate acceptance probability

With the proposal θ^*, we have to decide whether to accept this as our new value of θ^k

- If we accept θ^*, $\theta^k = \theta^*$
- If we reject it, $\theta^k = \theta^{k-1}$

We accomplish this using *Metropolis updates*, a very clever way of ensuring that our samples of θ occur in proportion to their probability in the posterior distribution

MCMC: basic steps

Step 2: Estimate acceptance probability

Remember that the support for the proposal conditional on the data is:

$$[\theta^*|y] = \frac{[y|\theta^*][\theta^*]}{[y]}$$

The support for the current value is:

$$[\theta^{k-1}|y] = \frac{[y|\theta^{k-1}][\theta^{k-1}]}{[y]}$$

These, of course, are the posterior probabilities of θ^* and θ^{k-1}

MCMC: basic steps

Step 2: Estimate acceptance probability

At first, this doesn't seem to have gotten us very far

- We started out saying that the hard part was finding $[y]$ and here it is again

But because $[y]$ does not depend on $[\theta]$, it cancels out if we take the ratio of the two probabilities:

$$R = \frac{[y|\theta^*][\theta^*]}{[y|\theta^{k-1}][\theta^{k-1}]} \tag{1}$$

MCMC: basic steps

Step 2: Estimate acceptance probability

Equation 1 tells us the *relative* probability of the proposal (relative to the current value), i.e., which value is more probable [1]

In Metropolis updating we accept θ^* whenever $R \geq 1$, i.e. when θ^* is more probable than θ^{k-1}

At first blush, you might assume that we reject θ^* when $R < 1$

- However, if we did this, we would rarely sample values of θ in the tails [1]
- Instead of automatically rejecting θ^* when $R < 1$, we treat R as a probability and we keep θ^* with probability R

MCMC: basic steps

Step 2: Estimate acceptance probability

We can now use our custom functions to create the chain. First, we will set up the parameters of the sampler:

```
n.samples <- 10000    # Number of samples to draw
sigma2 <- 0.015       # Tuning parameter

posterior <- data.frame(iteration = 1:n.samples,
                        phi = rep(NA, n.samples),
                        accept = rep(0, n.samples)) # Empty vector to

p <- init     # Initial value
```

MCMC: basic steps

Step 2: Estimate acceptance probability

Next, we run the sampler:

```
for(i in 1:n.samples){
  p.cand <- proposal(p = p, sigma2 = sigma2)   ## Proposal

  l.R <- (joint(data = n, N = 20, p = p.cand, a = 2, b = 4) - ## log
            joint(data = n, N = 20, p = p, a = 2, b = 4))

  R <- min(1, exp(l.R))   ## R

  if(runif(1) < R){
    p <- p.cand
    posterior$accept[i] <- 1
  }

  posterior$phi[i] <- p
}
```

MCMC: basic steps

Step 3: Summarizing and visualizing the chain

The `phi` column in the `posterior` data frame now contains 10^{4} samples from the posterior distribution of ϕ

We can use these samples to characterize the posterior distribution without having had to do any integration!

For example, we can plot the posterior distribution:

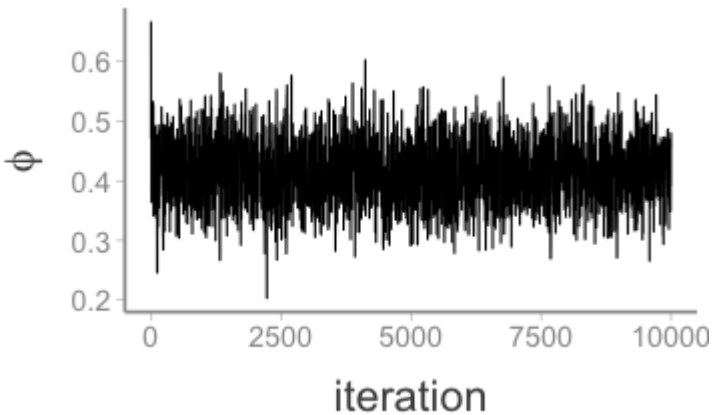
```
ggplot(posterior, aes(x = phi)) + geom_density(size = 1.5) +
  scale_x_continuous(expression(phi))
```

MCMC: basic steps

Step 3: Summarizing and visualizing the chain

We can also create a **trace plot** showing values of the chain at each iteration [1]

```
ggplot(posterior, aes(x = iteration, y = phi)) + geom_path() +
  scale_y_continuous(expression(phi))
```



MCMC: basic steps

Step 3: Summarizing and visualizing the chain

Or find the mean (remember that the true value in our simulation was 0.4):

```
mean(posterior$phi)
```

```
## [1] 0.4139
```

Pretty close (that a good sign that our Metropolis sampler did what we wanted it to do!)

MCMC: basic steps

Step 3: Summarizing and visualizing the chain

We can also estimate quantiles:

```
quantile(posterior$phi)
```

```
##     0%    25%    50%    75%   100%
## 0.2032 0.3814 0.4132 0.4456 0.6665
```

Quantiles are useful for determining the 95% *credible intervals* of the posterior: [1]

```
quantile(posterior$phi, probs = c(0.025, 0.975))
```

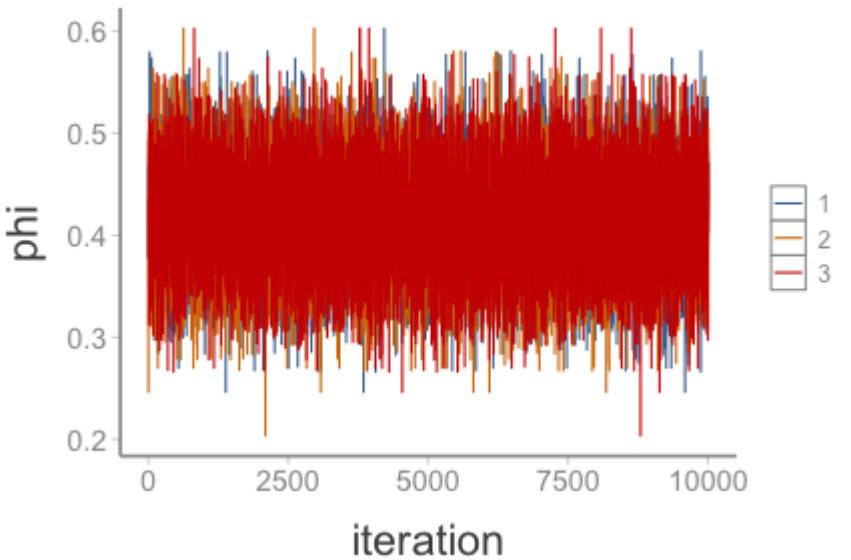
```
##   2.5%  97.5%
## 0.3188 0.5134
```

This allows us to make statements like "there is a 95% chance that ϕ is between 0.32 and 0.51" [2]

Multiple chains

So far, we have assumed that our MCMC is composed of just a single chain

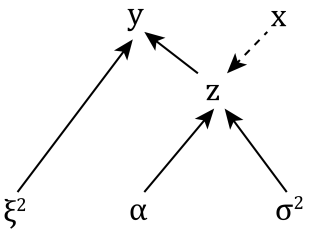
In most applications, we run multiple chains for each parameter [1]



Multiple parameters

What happens if our model contains multiple parameters?

- In most models, we need to find the marginal distributions of parameters that are part of multivariate joint distributions
- To do this, we rely on the conditional relationships defined by the Bayesian network [1]



$$[z, \alpha, \sigma^2, \xi^2 | y] \propto \underbrace{[y|z, \xi^2]}_{a} \underbrace{[z|g(\alpha, x), \sigma^2]}_{b} \underbrace{[\alpha]}_{c} \underbrace{[\sigma^2]}_{d} \underbrace{[\xi^2]}_{e}$$

Multiple parameters

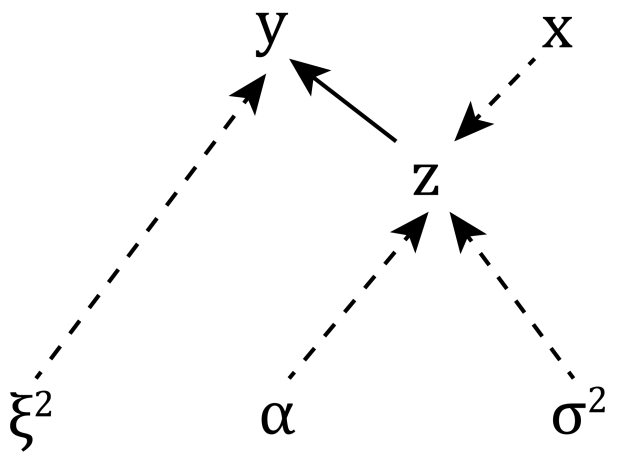
How do we develop a sampler for all of the random variables in this model?

We start by determining the conditional posterior distribution of each parameter

- the conditional posterior is defined by all of the nodes going to/from the parameter of interest
- all other nodes are ignored
- all parameters other than the parameter of interest are treated as **known constants**

Multiple parameters

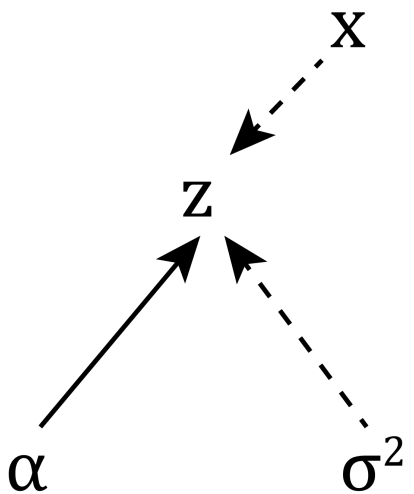
For z, the conditional posterior is: [1]



$$[z|.] \propto \underbrace{[y|z,\xi^2]}_{a}\underbrace{[z|g(\alpha,x),\sigma^2]}_{b}$$

Multiple parameters

For α, the conditional posterior is:



$$[\alpha|.] \propto \underbrace{[z|g(\alpha, x), \sigma^2]}_{b} \underbrace{[\alpha]}_{c}$$

Multiple parameters

The full-conditionals gives us a set of univariate distributions for sampling each random variable:

$$[z|.] \propto \underbrace{[y|z,\xi^2]}_{a}\underbrace{[z|g(\alpha,x),\sigma^2]}_{b}$$

$$[\alpha|.] \propto \underbrace{[z|g(\alpha,x),\sigma^2]}_{b}\underbrace{[\alpha]}_{c}$$

$$[\sigma^2|.] \propto \underbrace{[z|g(\alpha,x),\sigma^2]}_{b}\underbrace{[\sigma^2]}_{d}$$

$$[\xi^2|.] \propto \underbrace{[y|z,\xi^2]}_{a}\underbrace{[\xi^2]}_{e}$$

Multiple parameters

To construct an MCMC for this model, we use the same steps as for the single-parameter model, except at each iteration, we sample values from each conditional distribution in sequence

- As we move through the set, we use the updated parameters from the previous conditionals in the sequence in the RHS of subsequent conditionals [1,2]

Evaluating Markov chains

Evaluating Markov chains

MCMC is an enormously useful algorithm for finding the posterior distributions of random variables

But it does not always work :(

For any model, it is **critical** to evaluate the chains to determine whether you can trust inferences from the model

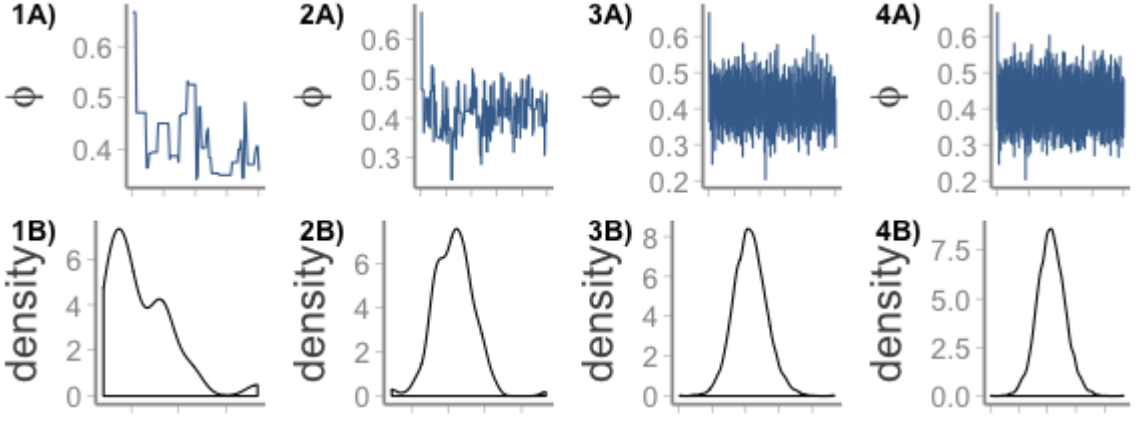
Evaluating Markov chains

The most important characteristic of MCMC chains is **convergence**

- Convergence occurs when the accumulated samples accurately characterize the posterior distribution of a parameter

Once a chain has converged, adding more samples will not meaningfully change the shape or moments of the posterior distribution

- At this point, a chain is *stationary* because more samples will not cause it to "move" [1]

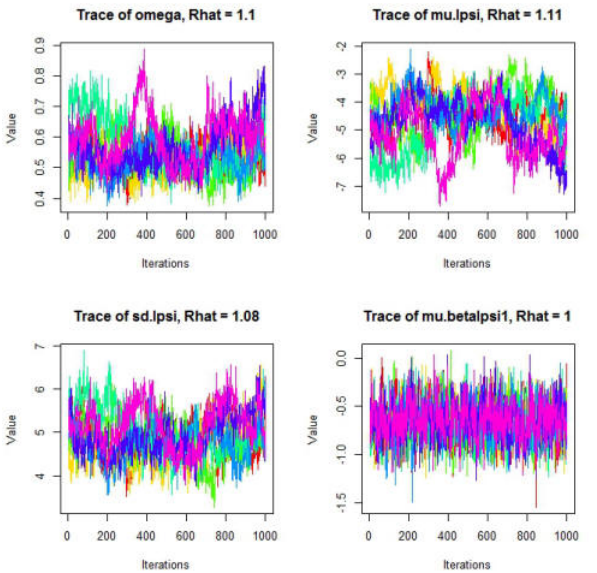


Evaluating convergence

There are two main ways to assess convergence: visually and using formal diagnostic tests

Visual evaluation is done using trace plots

- Chains that have not converged will "wander"



Evaluating convergence

There are two main ways to assess convergence: visually and using formal diagnostic tests

Formal diagnostic testing is commonly done using the **Gelman-Rubin diagnostic**, denoted $\hat{r}$

- when chains have not converged, variance among chains $>$ variance within chains
- when chains have converged, variance among chains $=$ variance within chains
- $\hat{r}$ is essentially the ratio of among:within chain variance so $\hat{r} \approx 1$ indicates convergence
- values much greater than 1 indicate lack of convergence (usually $\hat{r} \leq 1.1$ is used as a threshold)

Improving convergence

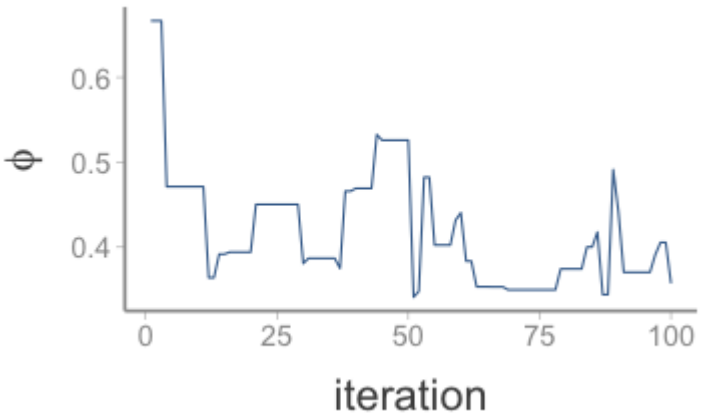
Two issues can cause chains to not converge:

1) Too few samples

2) Lack of identifiability

Improving convergence

If too few samples have been accumulated, the chain will not accurately represent the full posterior

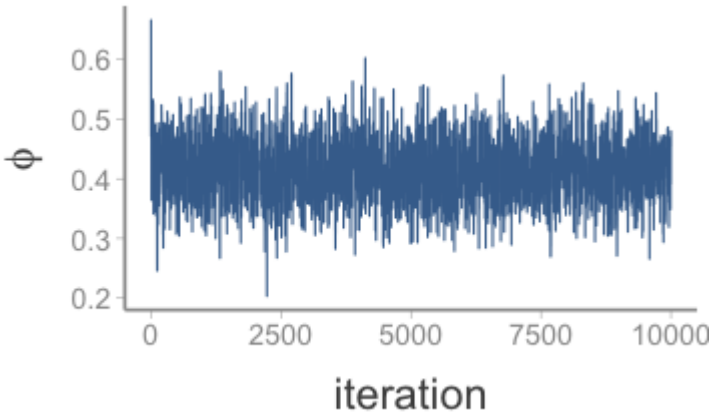


Solution - continuing sampling!

Improving convergence

If too few samples have been accumulated, the chain will not accurately represent the full posterior

Solution - continuing sampling!



Improving convergence

Solving lack of identifiability is harder

- reduce model complexity
- collect more data
- more informative priors [1]
- increase adaptation phase [2]
- transformations can sometimes help