Biostatistics 682: Applied Bayesian Inference
Lecture 7: Markov chain Monte Carlo

Jian Kang

Department of Biostatistics
University of Michigan, Ann Arbor

Markov chain Monte Carlo (MCMC)

- Markov Chain: a stochastic process in which future states are independent of past states given the present state
- Monte Carlo: simulation
- Up until now, we have done Monte Carlo simulations to find integrals rather than doing it analytically, a process called Monte Carlo Integration.
- Basically a fancy way of saying we can take quantities of interest of a distribution from simulated draws from the distribution

What is a Markov Chain

Definition: a stochastic process in which future states are independent of past states given the present state

Stochastic process: a *consecutive* set of random (not deterministic) quantities defined on some known state space.

- think of Θ as our parameter space
- *consecutive* implies a time component, indexed by t.

Consider a draw of $\theta^{(t)}$ to be a state at iteration t. The next draw $\theta^{(t+1)}$ is dependent only on the current draw $\theta^{(t)}$, and not on any past draws.

This satisfies the **Markov property**:

$$\pi(\theta^{(t+1)} \mid \theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(t)}) = \pi(\theta^{(t+1)} \mid \theta^{(t)})$$

Markov chain

- A bunch of draws of θ that are each slightly dependent on the previous one.
- The chain wanders around the parameter space, remembering only where it has been in the last period
- What are the rules governing how the chain jumps from one state to another at each period?
- The jumping rules are governed by a **transition kernel**, which is a mechanism that describes the probability of moving to some other state based on the current state.

Transition Kernel

- For discrete state space (k possible states): a $k \times k$ matrix of transition probabilities
- **Example**: Suppose $k = 3$. The 3×3 transition matrix $\mathbf{P}$ would be

$$\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix}$$

  where the subscripts index the three possible values that θ can take.

- The rows sum to one and define a conditional probability mass function, conditional on the current state.
- The columns are the marginal probabilities of being in a certain state in the next period.
- For continuous state space (infinite possible states), the transition kernel is a bunch of conditional probability density functions: $\pi(\theta^{(t+1)} \mid \theta^{(t)})$.

How does a Markov chain work? (Discrete Example)

- Define a starting distribution $\Pi^{(0)}$ (a $1 \times k$ vector of probabilities that sum to one).
- At iteration 1, our distribution $\Pi^{(1)}$ (from which $\theta^{(1)}$ is drawn) is

$$\Pi^{(1)} = \Pi^{(0)} \times \mathbf{P}$$

.....

- At iteration t, our distribution $\Pi^{(t)}$ (from which $\theta^{(t)}$ is drawn) is

$$\Pi^{(t)} = \Pi^{(t-1)} \times \mathbf{P} = \Pi^{(0)} \times \mathbf{P}^t$$

Stationary Distribution

- Define a stationary distribution π to be some distribution Π such that

$$\pi = \pi \mathbf{P}$$

- The typical MCMC algorithm constructs the Markov chain that converges to a stationary distribution regardless of our starting points.

- We can devise a Markov chain whose stationary distribution is our desired posterior distribution $\pi(\theta \mid y)$, then we can run this chain to get draws that are approximately from $\pi(\theta \mid y)$ once the chain has converged.

Burn-in

- Since convergence usually occurs regardless of our starting point, we can usually pick any feasible (for example, picking starting draws that are in the parameter space) starting point.
- However, the time it takes for the chain to converge varies depending on the starting point.
- As a matter of practice, most people throw out a certain number of the first draws, known as the burn-in. This is to make our draws closer to the stationary distribution and less dependent on the starting point.
- However, it is unclear how much we should burn-in since our draws are all slightly dependent and we do not know exactly when convergence occurs.

Monte Carlo integration on the Markov chain

- Once we have a Markov chain that has converged to the stationary distribution, then the draws in our chain appear to be like draws from $p(\theta \mid y)$, so it seems like we should be able to use Monte Carlo integration methods to find quantities of interest
- One problem: the draws are not independent, which we required for Monte Carlo integration to work (recall SLLN)
- Solution: Ergodic Theorem

Ergodic Theorem

- Let $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(M)}$ be M values from a Markov chain that is *aperiodic*, *irreducible* and *positive recurrent* (then the chain is ergodic), and $\mathbb{E}[g(\theta)] < \infty$.
- Then with probability 1,

$$\frac{1}{M}\sum_{i=1}^{M} g(\theta_i) \to \int_{\Theta} g(\boldsymbol{\theta})\pi(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}, \text{ as } M \to \infty$$

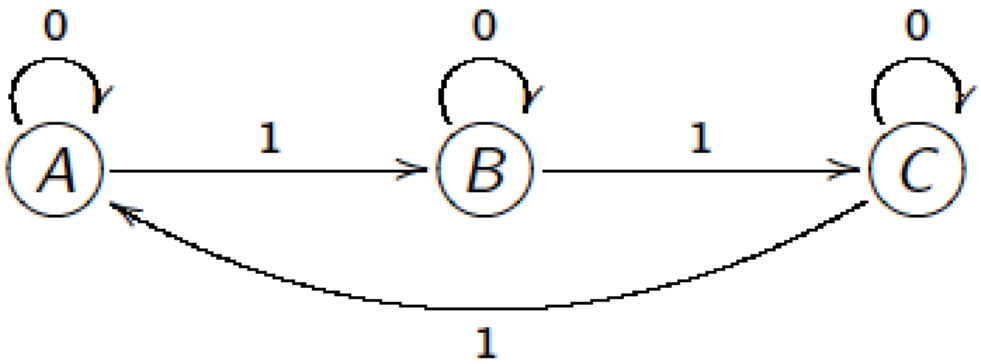
  where π is the stationary distribution

- This is the Markov chain analog to the strong law of large numbers (SLLN), and it allows us to ignore the dependence between draws of the Markov chain when we calculate quantities of interest from the draws.

Aperiodicity

- A Markov chain is **aperiodic** if the only length of time for which the chain repeats some cycle of values is the trivial case with cycle length equal to one

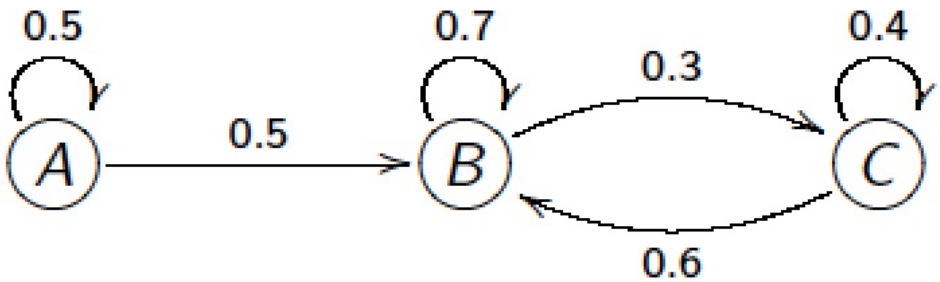
- Let A, B, and C denote the states (analogous to the possible values of θ) in a 3-state Markov chain. The following chain is periodic with period 3, where the period is the number of steps that it takes to return to a certain state.



- As long as the chain is not repeating an identical cycle, then the chain is **aperiodic**

Irreducibility

- A Markov chain is **irreducible** if it is possible go from any state to any other state (not necessarily in one step).
- The following chain is reducible, or not irreducible



- The chain is not irreducible because we cannot get to A from B to C regardless of the number of steps we take.

Positive Recurrence

- A Markov chain is **recurrent** if for any given state i, if the chain starts at i, it will eventually return to i with probability 1.
- A Markov chain is **positive recurrent** if the expected return time to state i is finite; otherwise it is *null recurrent*.
- If our Markov chain is **aperiodic**, **irreducible**, and **positive recurrent**, then it is ergodic and the ergodic theorem allows us to do Monte Carlo integration by calculating quantities of interest from our draws, ignoring the dependence between draws.

Detailed Balance Equation

- A Markov chain is said to be **reversible** if there is a probability distribution over states, $\boldsymbol{\pi}$, such that

$$\pi_i \Pr(X_{n+1} = j \mid X_n = i) = \pi_j \Pr(X_{n+1} = i \mid X_n = j),$$

  for all times n and all states i and j.
- This is also known as **detailed balance equation**
- For reversible Markov chains, π is always a stationary distribution of $\Pr(X_{n+1} = j \mid X_n = i)$.
- Reversible Markov chains are common in MCMC algorithms because the detailed balance equation for a target distribution necessarily implies that the Markov chain has been constructed so that the target distribution is a stationary distribution

Thinning the chain

In order to break the dependence between draws in the Markov chain, some have suggested only keeping every dth draw of the chain.

This is known as **thinning**.

Pros:

- Gets closer to independent identically distributed draws
- Saves memory since you only store a fraction of the draws

Cons:

- Unnecessary with ergodic theorem
- Shown to increase the variance of Monte Carlo estimates

What is MCMC?

- MCMC is a class of methods in which we can simulate draws that are slightly dependent and are approximately from a (posterior) distribution
- We then take those draws and calculate quantities of interest for the (posterior) distribution
- In Bayesian statistics, there are generally two MCMC algorithms: the **Gibbs sampler** and the **Metropolis-Hastings algorithm**

Gibbs Sampling

- Suppose we have a joint distribution $\pi(\theta_1, \ldots, \theta_k)$ as our target density distribution, where $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$.
- We can use the Gibbs sampler to sample from the joint distribution if we knew the **full conditional** distribution for each parameter
- For each parameter, the **full conditional** distribution is the distribution of the parameter conditional on the known information and all the other parameters $\pi(\theta_j \mid \boldsymbol{\theta}_{-j}, y)$, where $\boldsymbol{\theta}_{-j} = (\theta_1, \ldots, \theta_{j-1}, \theta_{j+1}, \ldots, \theta_k)$
- How can we know the joint distribution simply using the full conditional distributions?

The Hammersley-Clifford Theorem (for two blocks)

- Suppose we have a joint density $\pi(x, y)$. The theorem shows that the joint density can be represented in terms of the conditional densities $\pi(x \mid y)$ and $\pi(y \mid x)$:

$$\pi(x, y) = \frac{\pi(y \mid x)}{\int \frac{\pi(y|x)}{\pi(x|y)} \mathrm{d}y}$$

- We can write the denominator as

$$\int \frac{\pi(y \mid x)}{\pi(x \mid y)} \mathrm{d}y = \int \frac{\pi(x, y)/\pi(x)}{\pi(x, y)/\pi(y)} \mathrm{d}y = \int \frac{\pi(y)}{\pi(x)} \mathrm{d}y = \frac{1}{\pi(x)}.$$

- Thus, the right-hand side is

$$\frac{\pi(y \mid x)}{1/\pi(x)} = \pi(y \mid x)\pi(x) = \pi(x, y)$$

  The theorem shows that knowledge of the conditional densities allows us to get the joint density

- This works for more than two blocks of parameters

- But how do we figure out the full conditionals?

Steps to calculating full conditional distributions

Suppose we have a posterior $\pi(\boldsymbol{\theta} \mid \mathbf{y})$. To calculate the full conditionals for each θ_i in $\boldsymbol{\theta}$, do the following steps:

1. Write out the full posterior ignoring constants of proportionality
2. Pick a block of parameters (for example, θ_1) and drop everything that does not depend on θ_1
3. Use your knowledge of distributions to figure out what the normalizing constant is (and thus what the full conditional distribution $\pi(\theta_1 \mid \boldsymbol{\theta}_{-1}, \mathbf{y})$ is).
4. Repeat steps 2 and 3 for all parameter blocks.

The Gibbs Sampler

- Suppose that for some $p > 1$, the random variable $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)$, where θ_i's are either uni-or-multidimensional.
- Suppose that we can sample from the full conditional densities θ_i

$$\pi_i(\theta_i \mid \boldsymbol{\theta}_{-i})$$

where $\boldsymbol{\theta}_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_p)$ for $i = 1, 2, \ldots, p$. The associated Gibbs sampling algorithm is given by the following transition from $\theta^{(t)}$ to $\theta^{(t+1)}$

The Gibbs Sampler

For $t = 1, 2, \ldots, T$, repeat the following steps:
Given $\boldsymbol{\theta}^{(t)} = (\theta_1^{(t)}, \ldots, \theta_p^{(t)})$, generate

1. $\theta_1^{(t+1)} \sim \pi_1(\theta_1 \mid \theta_2^{(t)}, \ldots, \theta_p^{(t)})$
2. $\theta_2^{(t+1)} \sim \pi_2(\theta_2 \mid \theta_1^{(t+1)}, \theta_3^{(t)}, \ldots, \theta_p^{(t)})$

   ......

p. $\theta_p^{(t+1)} \sim \pi_p(\theta_p \mid \theta_1^{(t+1)}, \ldots, \theta_{p-1}^{(t+1)})$

Example

- Suppose we have data of the number of failures y_i for each of 10 pumps in a nuclear plan, denoted $\mathbf{y} = (y_1, \ldots, y_{10})$.
- We also have the times t_i at which each pump was observed, denoted $\mathbf{t} = (t_1, \ldots, t_{10})$.
- We want to model the number of failures with a Poisson likelihood, where the expected number of failure λ_i differs for each pump. Since the time which we observed each pump is different, we need to scale each λ_i by its observed time t_i.
- The likelihood is $\prod_{i=1}^{10} \text{Poisson}(\lambda_i t_i)$
- Prior specifications:

$$\lambda_i \sim \text{G}(\alpha, \beta)$$

$$\beta \sim \text{G}(\gamma, \delta)$$

  where we assume $\alpha = 1.8$, $\gamma = 0.01$ and $\delta = 1$.
- The total number of unknown parameters in the model is 11. (10 λ_i's s and β).

Example

Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)$. The posterior is

$$\begin{aligned}
\pi(\boldsymbol{\lambda}, \beta \mid \mathbf{y}, \mathbf{t}) &\propto \left(\prod_{i=1}^{10} \frac{e^{-\lambda_i t_i} (\lambda_i t_i)^{y_i}}{y_i!} \times \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda_i^{\alpha-1} e^{-\beta\lambda_i} \right) \frac{\delta^\gamma}{\Gamma(\gamma)} \beta^{\gamma-1} e^{-\delta\beta} \\
&\propto \left(\prod_{i=1}^{10} e^{-\lambda_i t_i} (\lambda_i t_i)^{y_i} \times \beta^\alpha \lambda_i^{\alpha-1} e^{-\beta\lambda_i} \right) \times \beta^{\gamma-1} e^{-\delta\beta} \\
&= \left(\prod_{i=1}^{10} \lambda_i^{y_i+\alpha-1} e^{-(t_i+\beta)\lambda_i} \right) \beta^{10\alpha+\gamma-1} e^{-\delta\beta}
\end{aligned}$$

Finding the full conditionals:

$$\begin{aligned}
\pi(\lambda_i \mid \boldsymbol{\lambda}_{-i}, \beta, \mathbf{y}, \mathbf{t}) &\propto \lambda_i^{y_i+\alpha+1} e^{-(t_i+\beta)\lambda_i} \\
\pi(\beta \mid \boldsymbol{\lambda}, \mathbf{y}, \mathbf{t}) &\propto e^{-\beta(\delta+\sum_{i=1}^{10}\lambda_i)} \beta^{10\alpha+\gamma-1}
\end{aligned}$$

This implies that

$$\lambda_i \mid \boldsymbol{\lambda}_{-i}, \beta, \mathbf{y}, \mathbf{t} \sim \mathrm{G}(y_i + \alpha, t_i + \beta), \quad \beta \mid \boldsymbol{\lambda}, \mathbf{y}, \mathbf{t} \sim \mathrm{G}(10\alpha + \gamma, \delta + \sum_{i=1}^{10} \lambda_i)$$

Metropolis-Hastings algorithm

Suppose we have a posterior $\pi(\boldsymbol{\theta} \mid \mathbf{y})$ that we want to sample from, but

- the posterior does not look like any distribution we know (no conjugacy)
- the posterior consists of many parameters (grid approximations intractable)
- some (or all) of the full conditionals do not look like any distributions we know (no Gibbs sampling for those whose full conditionals we don't know)

If all else fails, we can try the Metropolis-Hastings algorithm

Metropolis-Hastings Algorithm

1. Choose a starting value $\boldsymbol{\theta}$
2. At iteration t, draw a candidate $\boldsymbol{\theta}^*$ from a jumping distribution $J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})$
3. Compute an acceptance ratio (probability):

$$r = \frac{\pi(\boldsymbol{\theta}^* \mid \mathbf{y}) / J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})}{\pi(\boldsymbol{\theta}^{(t-1)} \mid \mathbf{y}) / J_t(\boldsymbol{\theta}^{(t-1)} \mid \boldsymbol{\theta}^*)}$$

4. Accept $\boldsymbol{\theta}^*$ as $\boldsymbol{\theta}^{(t)}$ with probability $\min(r, 1)$. If $\boldsymbol{\theta}^*$ is not accepted, then $\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)}$.
5. Repeat steps 2–4, M times to get M draws from $\pi(\theta \mid \mathbf{y})$ with optional burn-in and /or thinning.

Step 1: Choose a starting value $\boldsymbol{\theta}^{(0)}$

- This is equivalent to drawing from our initial stationary distribution.
- The important thing to remember is that $\boldsymbol{\theta}^{(0)}$ must have positive probability.

$$\pi(\boldsymbol{\theta}^{(0)} \mid \mathbf{y}) > 0$$

- Otherwise, we are starting with a value that cannot be drawn.

Step 2: Draw $\boldsymbol{\theta}^*$ from $J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})$

- The jumping distribution $J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})$ determines where we move to in the next iteration of the Markov chain (analogous to the transition kernel).
- **The support of the jumping distribution must contain the support of the posterior.**
- The original **Metropolis algorithm** required that $J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})$ be a symmetric distribution (such as the normal distribution), that is $J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})$
- For the Metropolis-Hastings algorithm, symmetry is unnecessary.
- If we have a symmetric jumping distribution that is dependent on $\boldsymbol{\theta}^{(t-1)}$, then we have what is known as **random walk Metropolis sampling**.

Step 2: Draw $\boldsymbol{\theta}^*$ from $J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})$ (Continued)

- If our jumping distribution does not depend on $\boldsymbol{\theta}^{t-1}$,

$$J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)}) = J_t(\boldsymbol{\theta}^*)$$

  then we have what is known as **independent Metropolis-Hastings sampling**

- Basically all our candidate draws $\boldsymbol{\theta}^*$ are drawn from the same distribution, regardless of where the previous draw was.

- This can be extremely efficient or extremely inefficient, depending on how close the jumping distribution is to the posterior.

- Generally speaking, chain will behave well only if the jumping distribution has heavier tails than the posterior.

Step 3: Compute acceptance ratio r

$$r = \frac{\pi(\boldsymbol{\theta}^* \mid \mathbf{y}) / J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})}{\pi(\boldsymbol{\theta}^{(t-1)} \mid \mathbf{y}) / J_t(\boldsymbol{\theta}^{(t-1)} \mid \boldsymbol{\theta}^*)}$$

- In the case where our jumping distribution is symmetric

$$r = \frac{\pi(\boldsymbol{\theta}^* \mid \mathbf{y})}{\pi(\boldsymbol{\theta}^{(t-1)} \mid \mathbf{y})}$$

- If our candidate draw has a higher probability than our current draw, then our candidate is better so we definitely accept it. Otherwise, our candidate is accepted according to the ratio of the probabilities of the candidate and current draws

- Note that since r is a ratio, we only need $\pi(\boldsymbol{\theta} \mid \mathbf{y})$ up to a constant of proportionality since $\pi(\mathbf{y})$ cancels out in both the numerator and denominator.

Step 3: Compute acceptance ratio r (Continued)

- In the case where our jumping distribution is not symmetric, we need to weight our evaluations of the draws at the posterior densities by how likely we are to draw each draw.

$$r = \frac{\pi(\boldsymbol{\theta}^* \mid \mathbf{y}) / J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})}{\pi(\boldsymbol{\theta}^{(t-1)} \mid \mathbf{y}) / J_t(\boldsymbol{\theta}^{(t-1)} \mid \boldsymbol{\theta}^*)}$$

- For example, if we are very likely to jump to some $\boldsymbol{\theta}^*$, then $J_t(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(t-1)})$ is likely to be high, so we should accept less of them than some other $\boldsymbol{\theta}^*$ that we are less likely to jump to.
- In the case of independent Metropolis-Hastings sampling,

$$r = \frac{\pi(\boldsymbol{\theta}^* \mid \mathbf{y}) / J_t(\boldsymbol{\theta}^*)}{\pi(\boldsymbol{\theta}^{(t-1)} \mid \mathbf{y}) / J_t(\boldsymbol{\theta}^{(t-1)})}$$

Step 4: Decide whether to accept $\boldsymbol{\theta}^*$

Accept $\boldsymbol{\theta}^*$ as $\boldsymbol{\theta}^{(t)}$ with probability $\min(r, 1)$. If $\boldsymbol{\theta}^*$ is not accepted, then $\theta^{(t)} = \theta^{(t?1)}$

1. For each $\boldsymbol{\theta}^*$, draw a value u from the $\mathrm{U}(0, 1)$ distribution.

2. If $u \leq r$, accept $\boldsymbol{\theta}^*$ as $\boldsymbol{\theta}^{(t)}$. Otherwise, use $\boldsymbol{\theta}^{(t-1)}$ as $\boldsymbol{\theta}^{(t)}$.

Candidate draws with higher density than the current draw are always accepted.

Unlike in rejection sampling, each iteration always produces a draw, either $\boldsymbol{\theta}^*$ or $\boldsymbol{\theta}^{(t-1)}$.

Acceptance Rates

- It is important to monitor the *acceptance rate* (the fraction of candidate draws that are accepted) of your Metropolis-Hastings algorithm.
- If the acceptance rate is too high, the chain is probably not mixing well (not moving around the parameter space quickly enough).
- If the acceptance rate is too low, the algorithm is too inefficient (rejecting too many candidate draws).
- What is too high and too low depends on your specific algorithm, but generally
  - Random walk: between 0.25 and 0.50 is recommended
  - Independent: close to one is preferred
  - High-dimensional problem: between 0.10 and 0.20

Tuning the MCMC algoritm

- MCMC is beautiful because it can handle virtually any statistical model and it is usually pretty easy to write functional code
- However, for hard problems great care must be taken to ensure that the algorithm has converged
- There are three main decisions:
  - Selecting the initial values
  - Determining if/when the chain(s) has converged
  - Selecting the number of samples needed to approximate the posterior

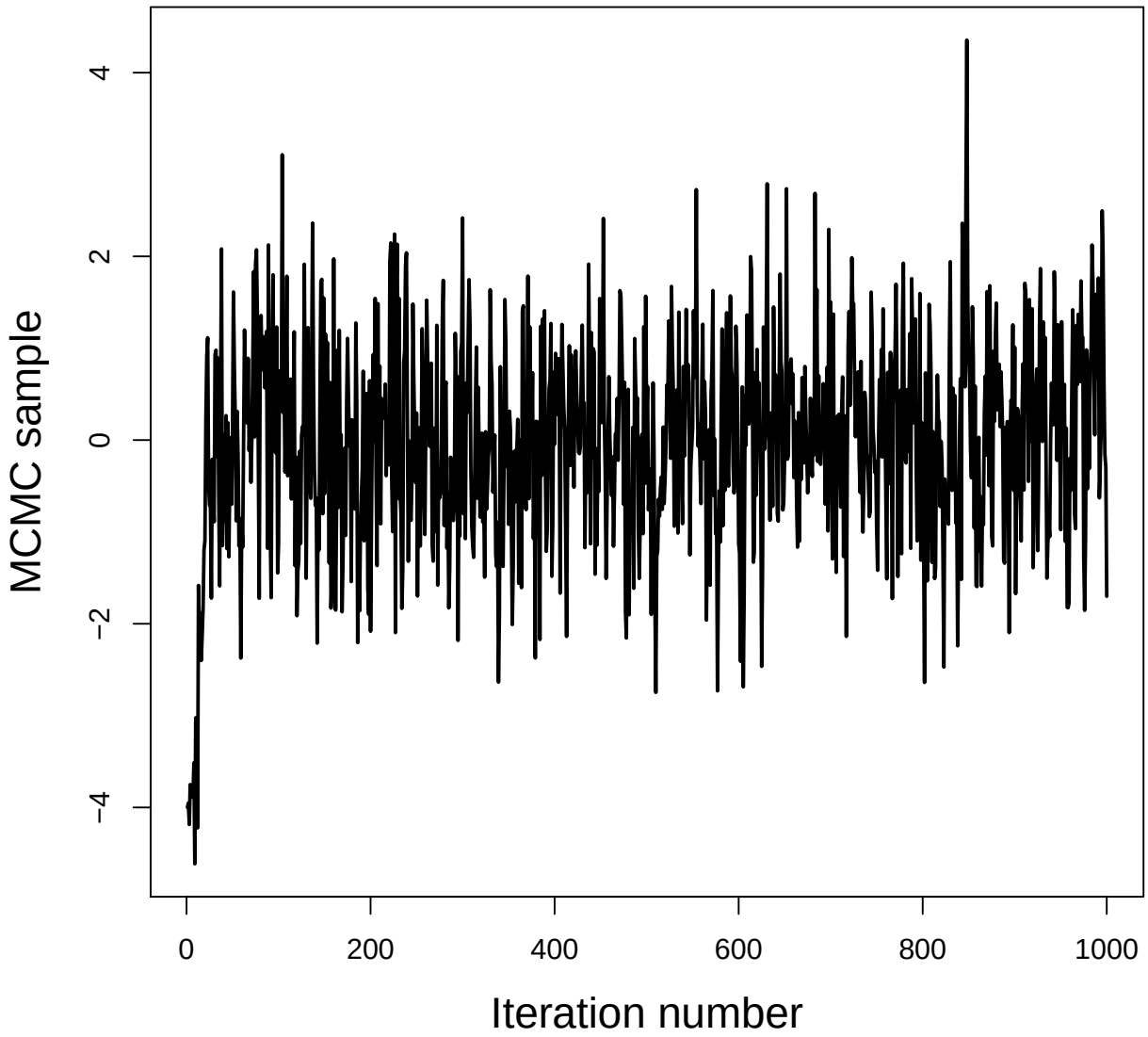
Initial values

- The algorithm will eventually converge no matter what initial values you select
- However taking time to select good initial values will speed up convergence
- It is important to try a few initial values to verify they all give the same result
- Usually 3-5 separate chains is sufficient
- **Option 1**: Select good initial values using method of moments or MLE
- **Option 2**: Purposely pick bad but different initial values for each chain to check convergence

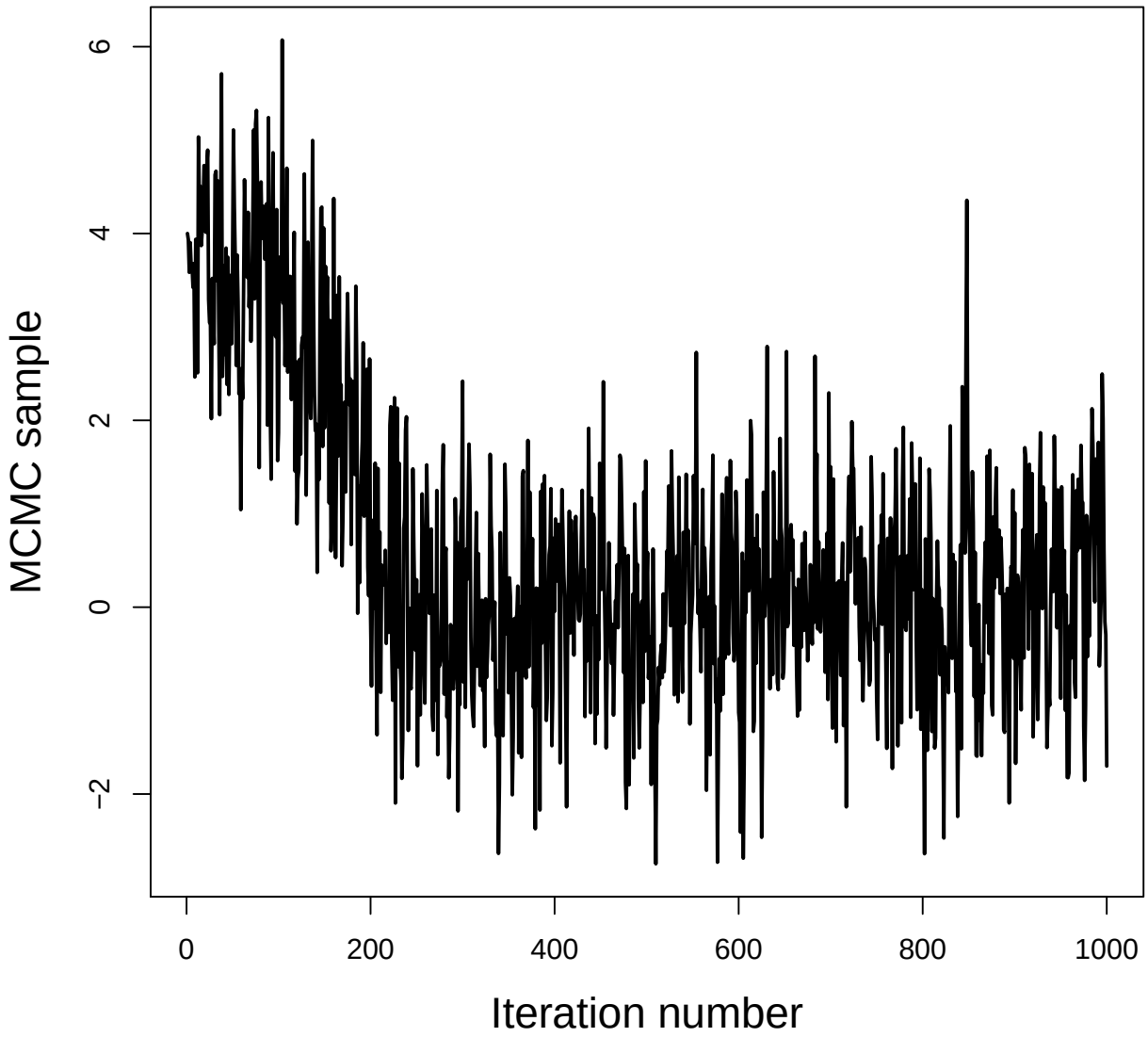
Convergence

- The first few samples are probably not draws from the posterior distribution
- It can take a few dozen, hundreds or even thousands of iterations to move from the initial values to the posterior
- When the sampler reaches the posterior this is called convergence
- Samples before convergence are discard as **burn-in**
- After convergence the samples should not converge to a single point!
- They should be draws from the posterior, and ideally look like a caterpillar or bar code

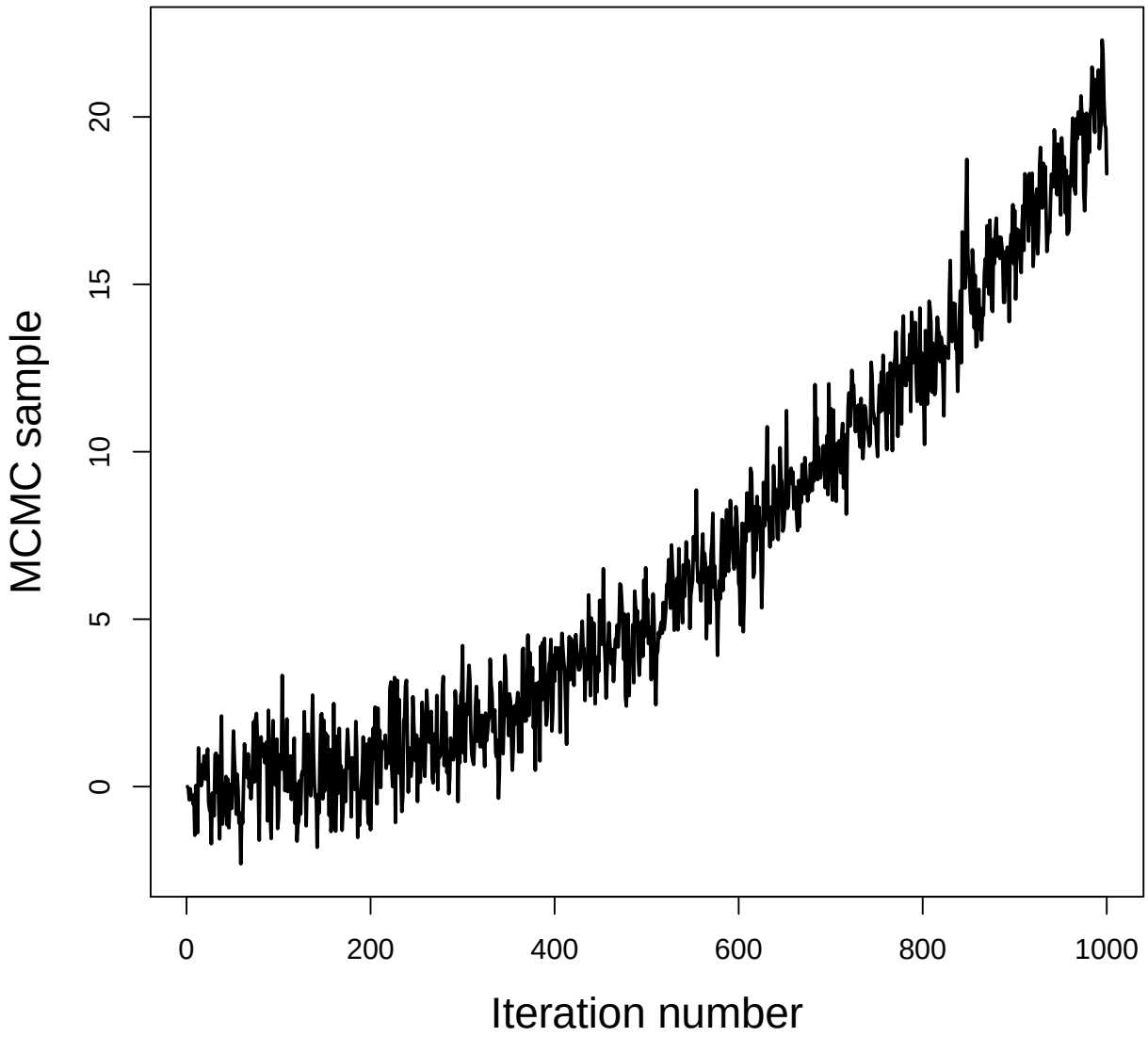
Convergence in a few iterations



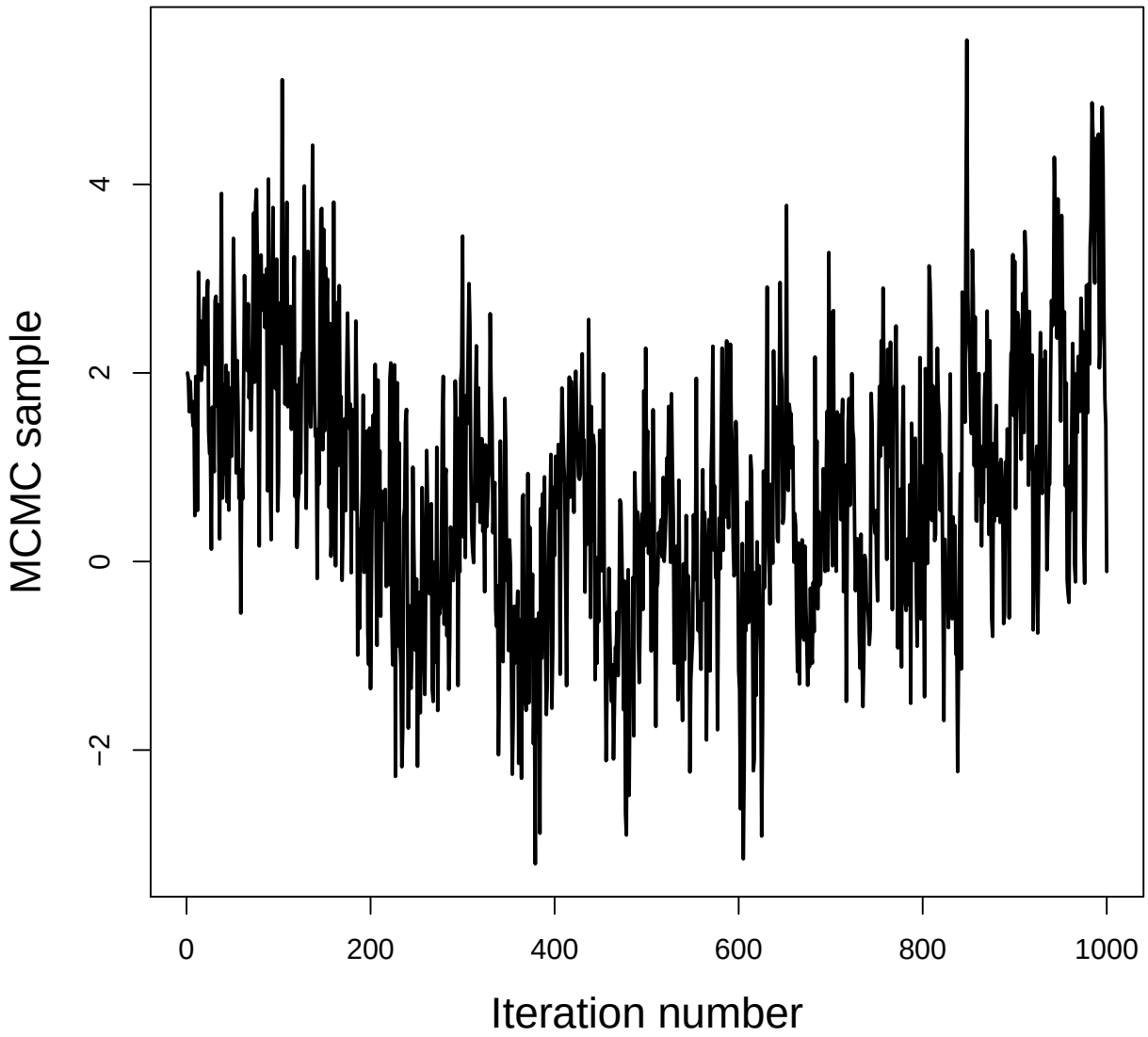
Convergence in a few hundred iterations



This one never converged



Convergence is questionable



Convergence diagnostics

- So far we have visually inspected the chains for convergence
- There are many formal diagnostics
- The `CODA` package in `R` has dozens of diagnostics
- Most give a measure of convergence for each parameter
- Checking convergence using these one-number summaries is more efficient and objective than visual inspection

Convergence diagnostics

- Did my chains converge?
  - Geweke
  - Gelman-Rubin
- Did I run the sampler long enough after convergence?
  - Effective sample size
  - Standard errors for the posterior mean estimate

Geweke diagnostic

- Compares the mean in the beginning of the chain with the mean at the end of the chain
- Can be used for a single chain
- Done separately for each parameter
- The test accounts for autocorrelation
- The test statistic is a z-score, so $|Z| > 2$ indicates poor convergence

Gelman-Rubin statistic

- If we run multiple chains, we hope that all chains give same result
- The Gelman-Rubin statistics measures agreement between chains
- Is it essentially an ANOVA test of whether the chains have the same mean
- It is scaled so that 1 is perfect and 1.1 is decent but not great convergence
- When the statistic reaches one this indicates convergence

Autocorrelation

- Ideally the samples would be independent across iteration
- The autocorrelation function $\rho(h)$ is the correlation between samples h iterations apart
- Lower values are better, but if the chains are long enough even large values can be OK
- **Thinning**: If autocorrelation is zero after lag h you can thin the samples by h to achieve independence
- This is always less efficient than using all samples, but can save memory

Effective sample size

- Highly correlated samples have less information than independent samples
- Say S is the actual number of MCMC samples
- The **effective samples size** is

$$ESS = \frac{S}{1 + 2\sum_{h=1}^{\infty} \rho(h)}$$

- The correlated MCMC sample of length S has the same information as ESS independent samples
- ESS should be at least a few thousand for all parameters

Standard errors of posterior mean estimates

- The sample mean of the MCMC draws is an estimate of the posterior mean
- The standard error of this estimate as another diagnostic
- Assuming independence the standard error is

$$\text{Naive SE} = \frac{s}{\sqrt{S}}$$

  where s is the sample SD and S is the number of samples

- A more realistic standard error is

$$\text{Times-series SE} = \frac{s}{\sqrt{ESS}}$$

What to do if the chains haven't converged?

- Run it longer
- Pick better initial values
- Pick tighter priors to add stability
- Update highly correlated parameters as a block
- Use a simpler model (e.g., remove collinear predictors)
- Try a different algorithm

Example: Bayesian Linear Regression

Linear regression with two predictors:

$$Y_i = \beta_0 + \beta_1 Z_{i1} + \beta_2 Z_{i2} + \epsilon_i, \qquad \epsilon_i \sim \mathrm{N}(0, \sigma^2)$$

with the following prior specifications.

$$\beta_0, \beta_1, \beta_2 \sim N(0, \tau^2), \quad \tau^{-2} \sim \mathrm{G}(a_\tau, b_\tau), \quad \sigma^{-2} \sim \mathrm{G}(a_\sigma, b_\sigma)$$

where $a_\tau, b_\tau, a_\sigma, b_\sigma$ are pre-specified hyper-parameters.
Let's compare the following MCMC algorithms

- Random walk
- Gibbs sampler
- blocked Gibbs sampler