BIOS 731: Advanced Statistical Computing
Lecture 4 on Bayesian Computation

Jian Kang

Department of Biostatistics and Bioinformatics

Emory University

Running Example

- As a running example, we will use the bivariate normal Metropolis sampler from class.

```
> NormalMHExample <- function(n.sim, n.burnin) {
+ library(mvtnorm)
+ mu.vector <- c(3, 1)
+ variance.matrix <- cbind(c(1, -0.5), c(-0.5, 2))
+ theta.mh <- matrix(NA, nrow = n.sim, ncol = 2)
+ theta.current <- rnorm(n = 2, mean = 0, sd = 4)
+ theta.update <- function(index, theta.current, ...) {
+ theta.star <- rnorm(n = 1, mean = theta.current[index], sd = 2)
+ if (index == 1)
+ theta.temp <- c(theta.star, theta.current[2])
+ else theta.temp <- c(theta.current[1], theta.star)
+ r <- dmvnorm(theta.temp, mu.vector, variance.matrix)/dmvnorm(theta.current,
+ mu.vector, variance.matrix)
+ r <- min(r, 1, na.rm = T)
+ if (runif(1) < r)
+ theta.star
+ else theta.current[index]
+ }
+ for (i in 1:n.sim) {
+ theta.current[1] <- theta.mh[i, 1] <- theta.update(1, theta.current,
+ mu.vector, variance.matrix)
+ theta.current[2] <- theta.mh[i, 2] <- theta.update(2, theta.current,
+ mu.vector, variance.matrix)
+ }
+ theta.mh <- theta.mh[(n.burnin + 1):n.sim, ]
+ }
> mh.draws <- NormalMHExample(n.sim = 5000, n.burnin = 0)
```

Convergence

- From our theory of Markov chains, we expect our chains to eventually converge to the stationary distribution, which is also our target distribution.
- However, there is no guarantee that our chain has converged after M draws.
- How do we know whether our chain has actually converged?
- We can never be sure, but there are several tests we can do, both visual and statistical, to see if the chain appears to be converged

Convergence Diagnostics in R

- All the diagnostics we will use are in the coda package in R.

```
> library(coda)
```

- Before we use the diagnostics, we should turn our chains into mcmc objects.

```
> mh.draws <- mcmc(mh.draws)
```

- We can tell the mcmc() function to burn-in or drop draws with the start and end arguments.
- mcmc() also has a thin argument, which only tells it the thinning interval that was used (it does not actually thin for us).

Posterior Summary

We can do summary() of an mcmc object to get summary statistics for the posterior.

```
> summary(mh.draws)
Iterations = 1:5000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 5000
1. Empirical mean and standard deviation for each variable,
plus standard error of the mean:
Mean SD Naive SE Time-series SE
[1,] 3.0282 1.027 0.01453 0.03859
[2,] 0.9997 1.424 0.02014 0.04109
2. Quantiles for each variable:
2.5% 25% 50% 75% 97.5%
var1 0.864 2.37214 3.0489 3.733 5.016
var2 -1.622 0.03447 0.9984 1.927 3.777
```

The results give the posterior means, posterior standard deviations, and posterior quantiles for each variable

The “naive” standard error is the standard error of the mean, which captures simulation error of the mean rather than posterior uncertainty.

$$\text{naive SE} = \frac{\text{posterior SD}}{\sqrt{n}}$$

The time-series standard error adjusts the “naive” standard error for autocorrelation.

Mixing

One way to see if our chain has converged is to see how well our chain is mixing, or moving around the parameter space.

If our chain is taking a long time to move around the parameter space, then it will take longer to converge.

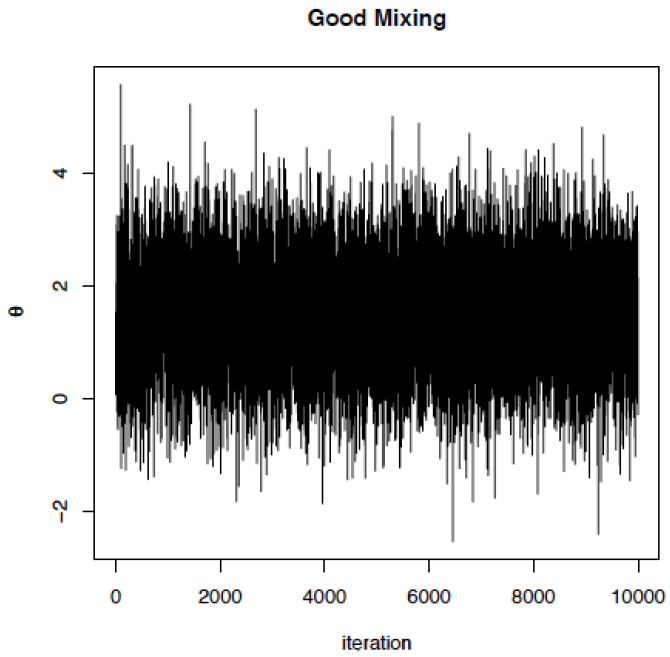
We can see how well our chain is mixing through visual inspection.

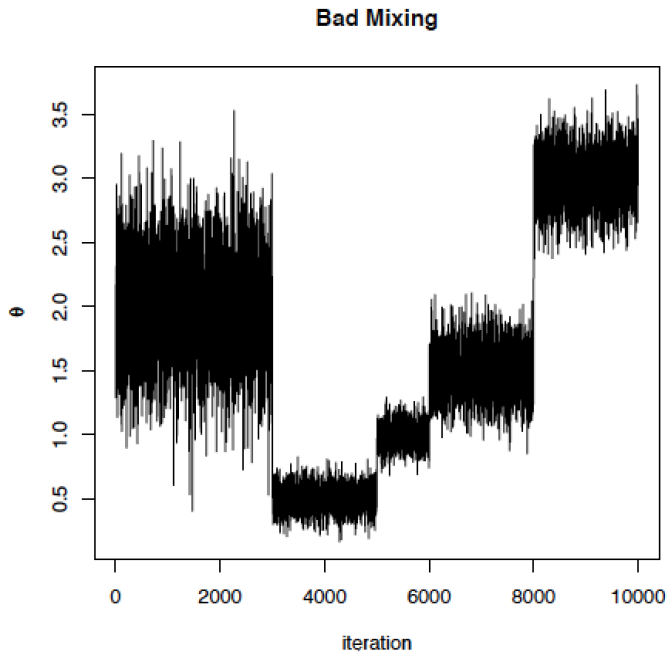
We need to do the inspections for every parameter

Traceplots

A traceplot is a plot of the iteration number against the value of the draw of the parameter at each iteration.

We can see whether our chain gets stuck in certain areas of the parameter space, which indicates bad mixing.

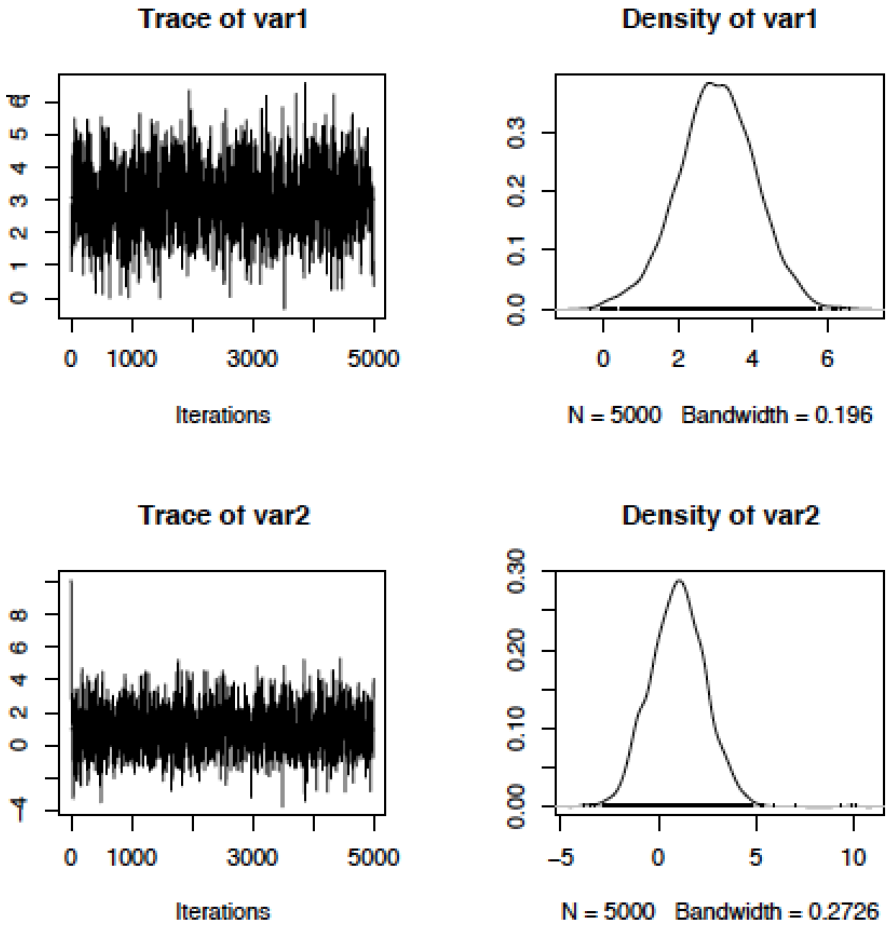




Traceplots and Density Plots

We can do traceplots and density plots by plotting an mcmc object or by calling the traceplot() and densplot() functions.

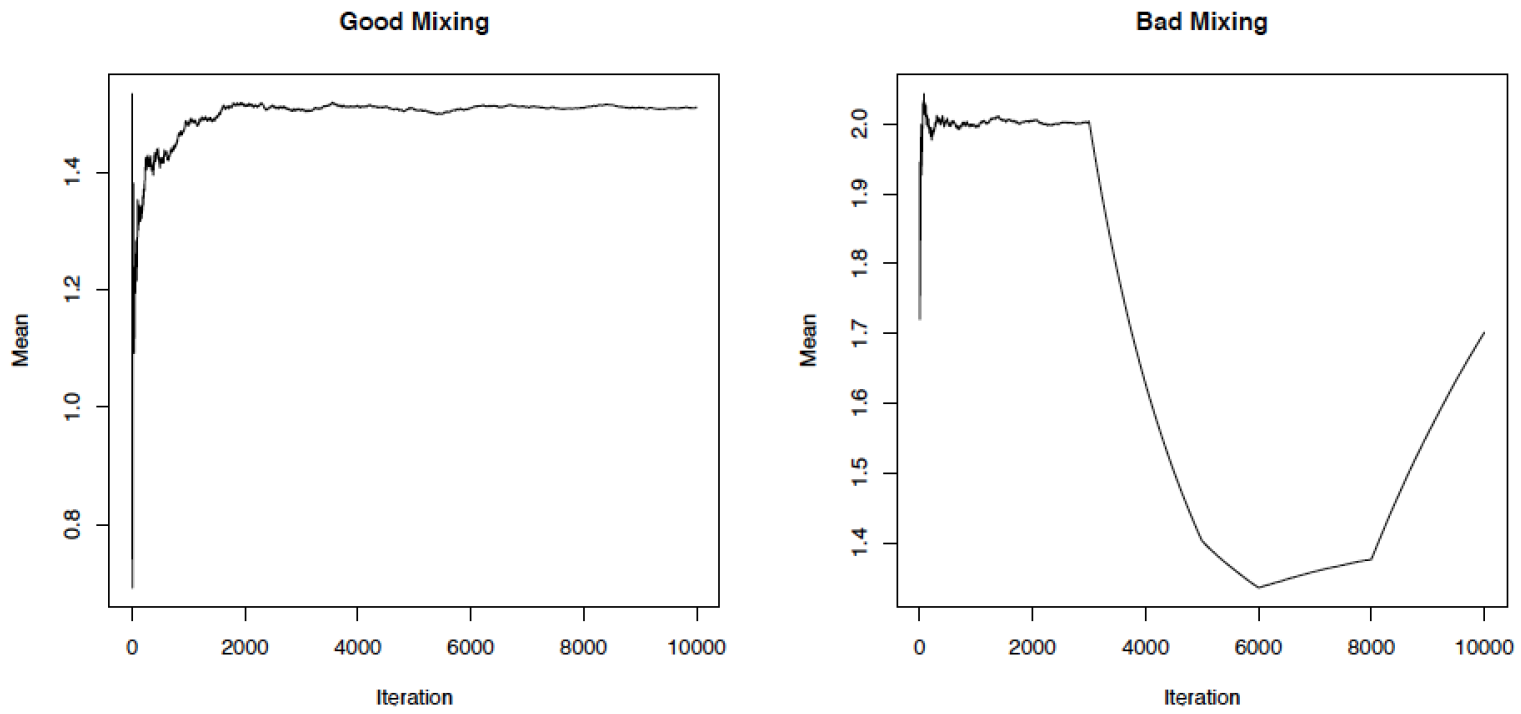
```
> plot(mh.draws)
```



Running Mean Plots

We can also use **running mean plots** to check how well our chains are mixing.

A running mean plot is a plot of the iterations against the mean of the draws up to each iteration.



Autocorrelation

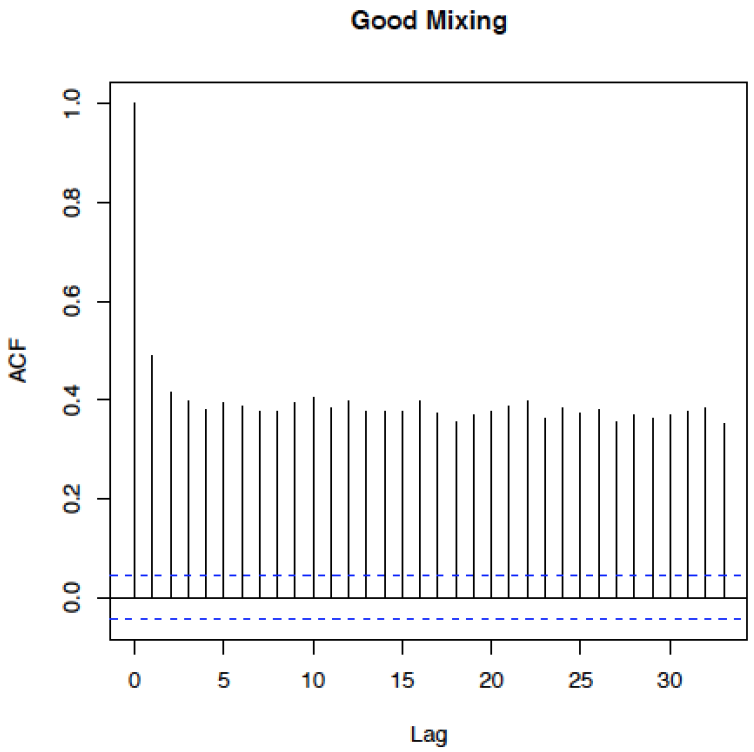
Another way to assess convergence is to assess the autocorrelations between the draws of our Markov chain.

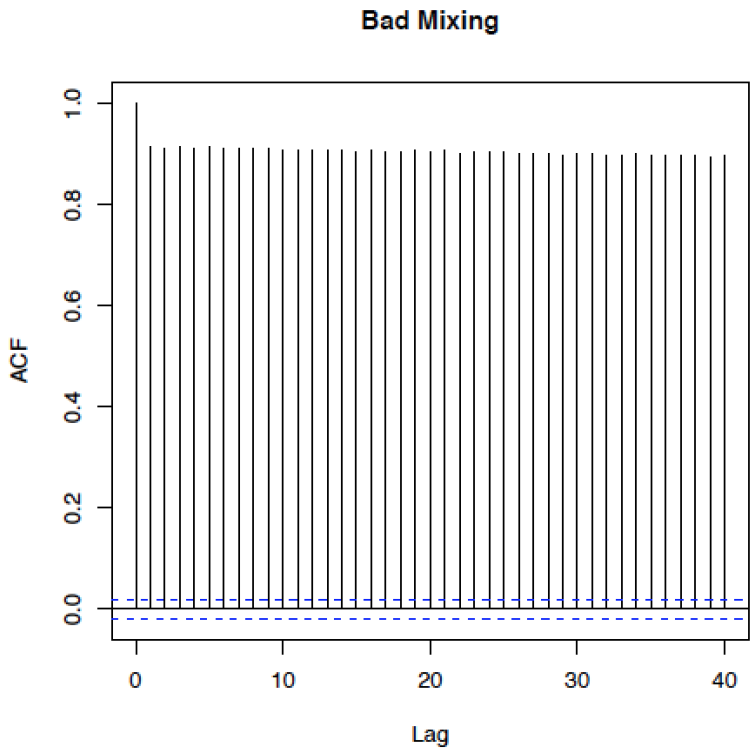
The lag k autocorrelation ρ_k is the correlation between every draw and its kth lag:

$$\rho_k = \frac{\sum_{i=1}^{n-k}(x_i - \overline{x})(x_{i+k} - \overline{x})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

We would expect the kth lag autocorrelation to be smaller as k increases (our 2nd and 50th draws should be less correlated than our 2nd and 4th draws).

If autocorrelation is still relatively high for higher values of k, this indicates high degree of correlation between our draws and slow mixing.

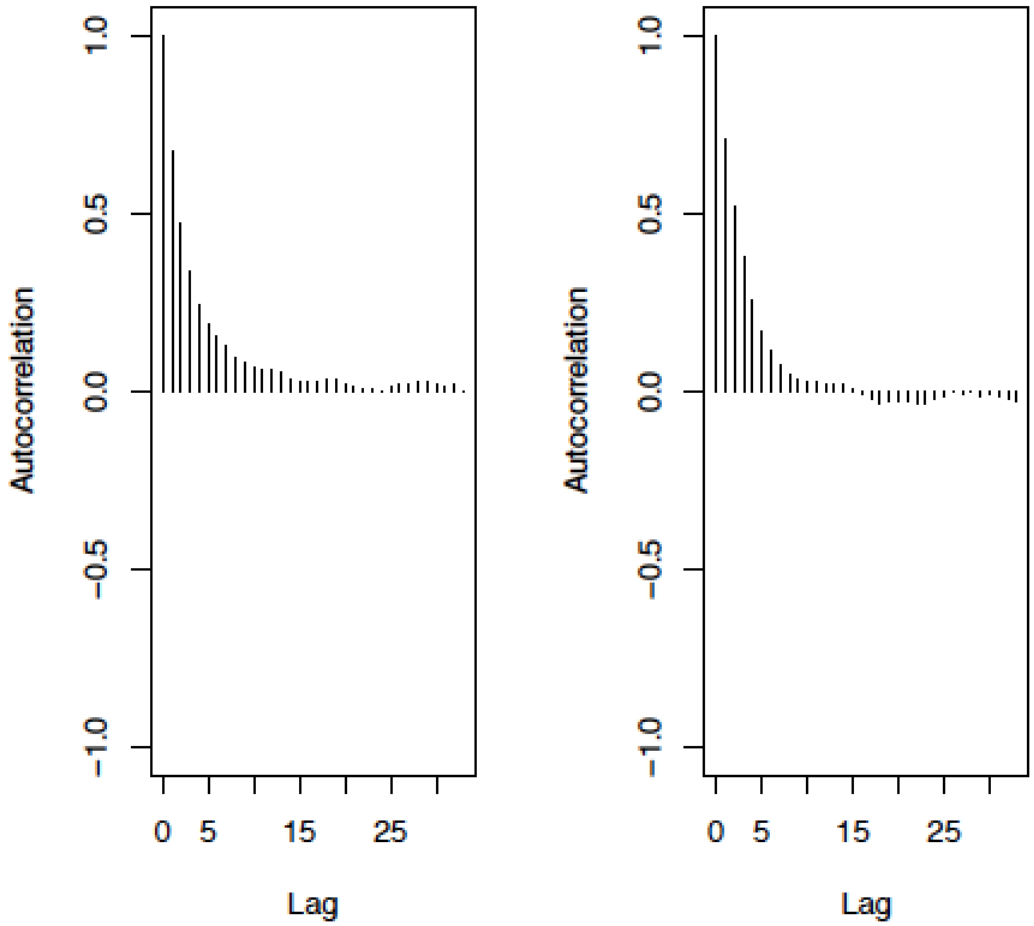

Good Mixing
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25
30
Lag


Bad Mixing
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
Lag

Autocorrelation Plots

We can get autocorrelation plots using the autocorr.plot() function.

```
> autocorr.plot(mh.draws)
```



Rejection Rate for Metropolis-Hastings Algorithm

We can also get a rejection rate for the Metropolis-Hastings algorithm using the rejectionRate() function the rejectionRate(mh.draws)

```
var1  var2
0.5277055  0.4094819
```

To get the acceptance rate, we just want 1 - rejection rate.

```
> acceptance.rate <- 1 - rejectionRate(mh.draws)
> acceptance.rate
var1  var2
0.4722945  0.5905181
```

Gelman and Rubin Multiple Sequence Diagnostic

Steps (for each parameter):

1. Run $m \geq 2$ chains of length $2n$ from overdispersed starting values.
2. Discard the first n draws in each chain.
3. Calculate the within-chain and between-chain variance.
4. Calculate the estimated variance of the parameter as a weighted sum of the within-chain and between-chain variance.
5. Calculate the potential scale reduction factor

Within Chain Variance

$$W = \frac{1}{m} \sum_{j=1}^{m} s_j^2$$

where

$$s_j^2 = \frac{1}{n-1} \sum_{i=1}^{n} (\theta_{ij} - \bar{\theta})^2$$

s_j^2 is just the formula for the variance of the jth chain.

W likely underestimates the true variance of the stationary distribution since our chains have probably not reached all the points of the stationary distribution

Between Chain Variance

$$B = \frac{n}{m-1} \sum_{j=1}^{m} (\bar{\theta}_j - \bar{\theta})^2$$

where

$$\bar{\theta} = \frac{1}{m} \sum_{j=1}^{m} \bar{\theta}_j$$

This is the variance of the chain means multiplied by n because each chain is based on n draws

Estimated Variance

We can then estimate the variance of the stationary distribution as a weighted average of W and B.

$$\widehat{\text{Var}}(\theta) = (1 - \frac{1}{n})W + \frac{1}{n}B$$

Because of overdispersion of the starting values, this overestimates the true variance, but is unbiased if the starting distribution equals the stationary distribution (if starting values were not overdispersed).

Potential Scale Reduction Factor

The potential scale reduction factor is

$$\widehat{R} = \sqrt{\frac{\widehat{\text{Var}}(\theta)}{W}}$$

When $\widehat{R}$ is high (perhaps greater than 1.1 or 1.2), then we should run our chains out longer to improve convergence to the stationary distribution.

If we have more than one parameter, then we need to calculate the potential scale reduction factor for each parameter.

We should run our chains out long enough so that all the potential scale reduction factors are small enough.

We can then combine the mn total draws from our chains to produce one chain from the stationary distribution.

Gelman and Rubin diagnostic in R.

First, we run M chains at different (should be overdispersed) starting values ($M = 5$ here) and convert them to mcmc objects.

```
> mh.draws1 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
> mh.draws2 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
> mh.draws3 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
> mh.draws4 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
> mh.draws5 <- mcmc(NormalMHExample(n.sim = 5000, n.burnin = 0))
```

We then put the M chains together into an mcmc.list.

```
> mh.list <- mcmc.list(list(mh.draws1, mh.draws2, mh.draws3, mh.draws4,
+ mh.draws5))
```

We then run the diagnostic with the gelman.diag() function.

```
> gelman.diag(mh.list)
Potential scale reduction factors:
Point est. 97.5% quantile
[1,] 1.00 1.00
[2,] 1.00 1.00
Multivariate psrf
1.00
```

We can see how the potential scale reduction factor changes through the iterations using the gelman.plot() function.

```
> gelman.plot(mh.list)
```