Chapter 7: Markov Chain Monte Carlo 1

ST4234: Bayesian Statistics

Semester 2, AY 2019/2020

Department of Statistics and Applied Probability

National University of Singapore

LI Cheng

stalic@nus.edu.sg

Introduction

- In Chapter 6, we introduced several approaches such as rejection sampling and importance sampling for simulating from posterior distributions that are not of familiar functional forms. These methods require suitable proposal densities which are good approximations to the posterior but still easy to sample from.

- In high-dimensional problems, it may be quite difficult or even impossible to find appropriate proposal densities.

- In this chapter, we illustrate the use of Markov chain Monte Carlo (MCMC) algorithms for simulating from posterior distributions. MCMC algorithms are very attractive in that they are easy to set up and program and require relatively little prior input from the user.

- We will introduce the Metropolis-Hastings algorithm. In the next Chapter, we introduce the Gibbs sampler.

Outline

7.1 Introduction to Markov Chains

First, let us look at an example of a **discrete** Markov chain.

- Suppose a person takes a random walk on the number line shown below:

- If the person is currently at an interior value (2, 3, 4, or 5), in the next second he is equally likely to remain at that number or move to an adjacent number. If he does move, he is equally likely to move left or right.
- If the person is currently at one of the end values (1 or 6), in the next second he is equally likely to stay still or move to the adjacent location.

7.1 Introduction to Markov Chains

- A Markov chain describes probabilistic movement between a number of states. Here there are six possible states, corresponding to the possible locations of the walker.
- Given that the person is at a current location, he moves to other locations with specified probabilities. The probability that he moves to another location depends only on his current location and not on previous locations visited.

Markov chain

In probability theory, a Markov chain is a sequence of random variables $\theta^{(1)}, \theta^{(2)}, \ldots$, such that, for any t, the distribution of $\theta^{(t)}$ given all previous θs depends only on the most recent value, $\theta^{(t-1)}$. That is

$$p(\theta^{(t)}|\theta^{(1)}, \ldots, \theta^{(t-1)}) = p(\theta^{(t)}|\theta^{(t-1)}).$$

7.1 Introduction to Markov Chains

- We describe movement between states in terms of transition probabilities – they describe the likelihoods of moving between all possible states in a single step in a Markov chain.
- We summarize the transition probabilities using a transition matrix P:

$$P = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0.25 & 0.5 & 0.25 & 0 & 0 & 0 \\ 0 & 0.25 & 0.5 & 0.25 & 0 & 0 \\ 0 & 0 & 0.25 & 0.5 & 0.25 & 0 \\ 0 & 0 & 0 & 0.25 & 0.5 & 0.25 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 \end{bmatrix}$$

- The first row in P gives the probabilities of moving to all states 1 through 6 in a single step from location 1. The second row gives the transition probabilities in a single step from location 2, and so on.

7.1 Introduction to Markov Chains

- We can represent one's current location as a probability row vector $\boldsymbol{p} = (p_1, \ldots, p_6)$, where p_i represents the probability that the person is currently in state i.

- If $\boldsymbol{p}^{(j)}$ represents the location of the traveler at step j, then the location of the traveler at the $j+1$ step is given by the matrix product $\boldsymbol{p}^{(j+1)} = \boldsymbol{p}^{(j)} P$.

- For example, suppose the traveler starts at "1", then $\boldsymbol{p}^{(1)} = (1, 0, 0, 0, 0)$. In the next step, the traveler is located at "1" or "2" with equal probability, so $\boldsymbol{p}^{(2)} = (0.5, 0.5, 0, 0, 0, 0)$ which is equal to $\boldsymbol{p}^{(1)} P$. You can also verify that $\boldsymbol{p}^{(3)} = \boldsymbol{p}^{(2)} P$.

7.1 Introduction to Markov Chains

- Suppose we can find a probability vector π such that $\pi P = \pi$. Then π is said to be the **stationary distribution**.
- If a Markov chain is irreducible and aperiodic, then it has a **unique** stationary distribution, which is the unique solution of π to the equation $\pi P = \pi$.
- For more of Markov chain concepts such as being "irreducible and aperiodic", please refer to the tutorial file "Markov chains tutorial.pdf".
- Moreover, the limiting distribution of this Markov chain, as the number of steps approaches infinity, will be equal to this stationary distribution.
- We can demonstrate the existence of the stationary distribution of our Markov chain by running a simulation experiment.

7.1 Introduction to Markov Chains

- Suppose we start our random walk at location 3, and then simulate many steps (say 50000) of the Markov chain using the transition matrix P.
- The relative frequencies of our traveler in the six locations will eventually approach the stationary distribution π after many steps.
- We start our simulation in R by reading in the transition matrix P and setting up a storage vector `l` for the locations of our traveler in the random walk.
- We indicate that the starting location for our traveler is state 3.

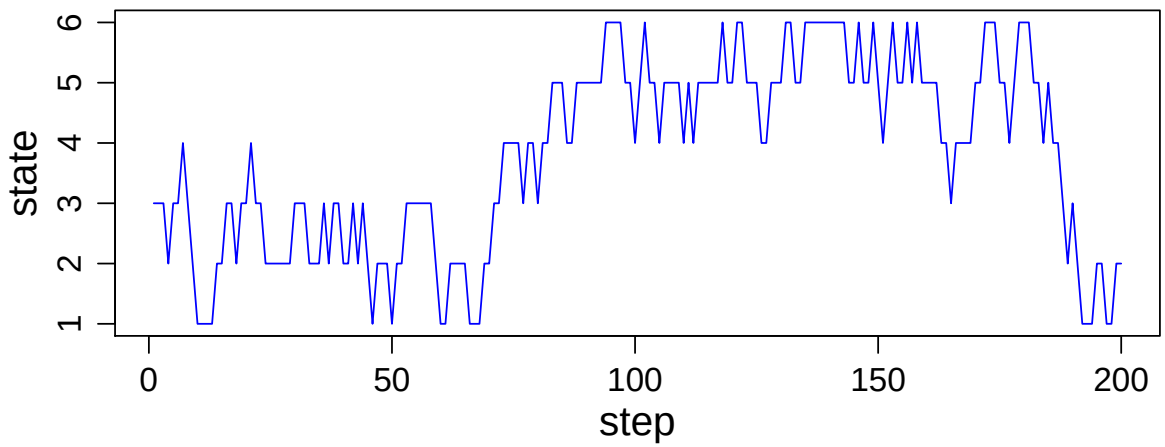
```
P <- rbind(    c(0.5,0.5,0,0,0,0), c(0.25,0.5,0.25,0,0,0),
           c(0,0.25,0.5,0.25,0,0), c(0,0,0.25,0.5,0.25,0),
           c(0,0,0,0.25,0.5,0.25), c(0,0,0,0,0.5,0.5))
N <- 50000          # total no. of steps
l <- rep(0,N)       # path taken
l[1] <- 3           # starting location = 3
```

7.1 Introduction to Markov Chains

- Next, we simulate the path taken by the traveler in 50,000 steps.
- At each step, we use the `sample` function to sample a state from $\{1, 2, 3, 4, 5, 6\}$ with probabilities given by the appropriate row of the transition matrix P. For example, if the traveler is currently at state i, then the probabilities of the next state is given by the ith row of P.

```
for (i in 2:N){l[i] <- sample(1:6, size=1, prob=P[l[i-1],])}
```

- The figure below shows the simulated path for the first 200 steps.



7.1 Introduction to Markov Chains

- We can summarize the frequencies of visits to the six states after 500, 5000 and 50,000 steps of the chain using the table command. The counts are converted to relative frequencies by dividing by the number of steps.

```
> table(l[1:500])/500
    1     2     3     4     5     6
0.094 0.170 0.188 0.198 0.262 0.088

> table(l[1:5000])/5000
     1      2      3      4      5      6
0.0922 0.2004 0.2152 0.2194 0.1904 0.0824

> table(l[1:50000])/50000
      1       2       3       4       5       6
0.10212 0.20108 0.19846 0.20088 0.19884 0.09862
```

7.1 Introduction to Markov Chains

- It appears from the output that the relative frequencies of the states are converging to the stationary distribution

$$\boldsymbol{\pi} = (0.1, 0.2, 0.2, 0.2, 0.2, 0.1).$$

- We can confirm that $\boldsymbol{\pi}$ is indeed the stationary distribution of this chain by multiplying $\boldsymbol{\pi}$ by the transition matrix P (here, `w` represents $\boldsymbol{\pi}$):

```
> w <- matrix(c(0.1,0.2,0.2,0.2,0.2,0.1),nrow=1)
> w%*%P
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  0.1  0.2  0.2  0.2  0.2  0.1
```

7.1 Introduction to Markov Chains

Markov chain simulation

- The discrete Markov chain setup described in the previous section can be generalized to a continuous setting.
- Suppose we are interested in simulating draws from a posterior $p(\theta|\boldsymbol{y})$. Let $\theta^{(0)}$ be some initial guess at the value of θ and $g(\theta^{(t)}|\theta^{(t-1)})$ be a transition probability distribution conditional on $\theta^{(t-1)}$.
- Starting at $\theta^{(0)}$, a Markov chain $\theta^{(1)}, \theta^{(2)}, \ldots$, can be created by drawing $\theta^{(t)}$ from $g(\theta^{(t)}|\theta^{(t-1)})$ for $t = 1, 2, \ldots$.
- If $g(\theta^{(t)}|\theta^{(t-1)})$ is constructed such that the stationary distribution of the Markov chain is $p(\theta|\boldsymbol{y})$ and the simulation has been run long enough such that the chain has converged, then current draws can be regarded as samples from $p(\theta|\boldsymbol{y})$.

7.1 Introduction to Markov Chains

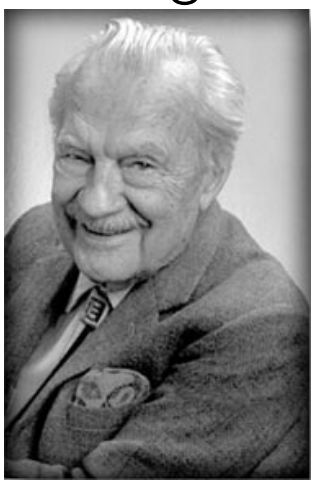
Markov chain simulation

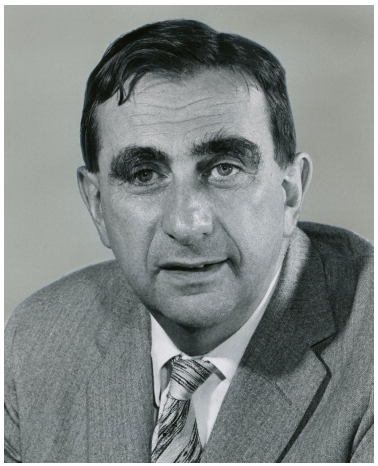
- This process of sequentially sampling parameter values from a Markov chain whose stationary distribution is the posterior of interest is called a Markov chain Monte Carlo (MCMC) method.
- MCMC methods are often used when it is not possible or computationally efficient to sample directly from $p(\theta|\boldsymbol{y})$.
- The key is to create a Markov process whose stationary distribution is $p(\theta|\boldsymbol{y})$ and to run the simulation long enough so that the distribution of the current draws is close enough to this stationary distribution. For any $p(\theta|\boldsymbol{y})$, a variety of Markov chains can be constructed.
- Once the simulation algorithm has been implemented and the simulations drawn, it is necessary to check the convergence of the simulated sequence to its stationary distribution; e.g. by using plots or numerical summaries of the sampled output from the chain.

Outline

7.2 Metropolis-Hastings Algorithm

- The Metropolis-Hastings algorithm (MH) is a general term for a family of Markov chain simulation methods that are useful for drawing samples from Bayesian posterior distributions.






- The basic version was developed in 1953 by Nicholas Constantine Metropolis and 4 coauthors, and was later extended by Wilfred Keith Hastings in 1970.
- In this section, we focus on two particular variants of Metropolis-Hastings algorithms, the independence chain and the random walk chain, that are applicable to a wide variety of Bayesian inference problems.

7.2 Metropolis-Hastings Algorithm

- Suppose we wish to simulate from a posterior density $p(\theta|\boldsymbol{y})$ where

$$p(\theta|\boldsymbol{y}) \propto f(\theta).$$

  For notational convenience, we suppress dependence of f on the data $\boldsymbol{y}$.

- A Metropolis-Hastings algorithm begins with an initial value $\theta^{(0)}$ and specifies a rule for simulating the tth value in the sequence, $\theta^{(t)}$, given the $(t-1)$th value in the sequence, $\theta^{(t-1)}$.

- This rule consists of a proposal density, which simulates a candidate value θ^*, and the computation of an acceptance probability, which indicates the probability that the candidate value will be accepted as the next value in the sequence.

Metropolis-Hastings algorithm

1. Choose a starting point $\theta^{(0)}$, for which $p(\theta^{(0)}|\boldsymbol{y}) > 0$.
2. For $t = 1, 2, \ldots, T$:
   - Sample a proposal θ^* from a proposal distribution $g(\theta^*|\theta^{(t-1)})$.
   - Compute the ratio $r = \dfrac{f(\theta^*)g(\theta^{(t-1)}|\theta^*)}{f(\theta^{(t-1)})g(\theta^*|\theta^{(t-1)})}$.
   - Set $\alpha(\theta^{(t-1)}, \theta^*) = \min\{r, 1\}$.
   - Set

$$\theta^{(t)} = \begin{cases} \theta^* & \text{with probability } \alpha(\theta^{(t-1)}, \theta^*), \\ \theta^{(t-1)} & \text{otherwise.} \end{cases}$$

7.2 Metropolis-Hastings Algorithm

- Here, $g(\theta_a|\theta_b)$ is a conditional density function of θ_a given θ_b.
- Under some easily satisfied regularity conditions on the proposal density $g(\theta^*|\theta^{(t-1)})$, the simulated draws $\theta^{(1)}, \theta^{(2)}, \ldots$ will eventually converge to a random variable that is distributed according to the posterior $p(\theta|\boldsymbol{y})$.
- The starting value $\theta^{(0)}$ is an initial guess at the value of θ and may be drawn, say from a normal approximation of the posterior. Alternatively, we may simply choose starting values dispersed around the posterior mode.
- The Metropolis-Hastings algorithm is a generalization of the Metropolis algorithm. The Metropolis algorithm is more restrictive (but practically more commonly used) - it requires that the proposal density be symmetric, i.e. $g(\theta_a|\theta_b) = g(\theta_b|\theta_a)$ for all θ_a, θ_b. Under this condition, the ratio r simplifies to $f(\theta^*)/f(\theta^{(t-1)})$.

7.2 Metropolis-Hastings Algorithm

- Different Metropolis-Hastings algorithms can be constructed depending on the choice of proposal density.
- If the proposal density is independent of the current value in the sequence, i.e.

$$g(\theta^*|\theta^{(t-1)}) = g(\theta^*),$$

then the resulting algorithm is called an independence chain.

- If the proposal density takes the form

$$g(\theta^*|\theta^{(t-1)}) = h(\theta^* - \theta^{(t-1)}),$$

where h is a symmetric density about the origin, then g is symmetric and $g(\theta^*|\theta^{(t-1)}) = g(\theta^{(t-1)}|\theta^*)$. In this type of random walk chain, the ratio $r = f(\theta^*)/f(\theta^{(t-1)})$.

- Common choices for h are the normal densities and the Student's t densities.

7.2 Metropolis-Hastings Algorithm

Intuition for Metropolis Algorithm

- Suppose that g is symmetric and

$$r = \frac{f(\theta^*)}{f(\theta^{(t-1)})} = \frac{p(\theta^*|\boldsymbol{y})}{p(\theta^{(t)}|\boldsymbol{y})}$$

- **If** $r \geq 1$:
  - Intuition: Since $\theta^{(t-1)}$ is already in our set, we should include θ^* as it has a higher probability than $\theta^{(t-1)}$.
  - Procedure: Accept θ^* into our set, i.e. set $\theta^{(t)} = \theta^*$.
- **If** $r < 1$:
  - Intuition: The relative frequency of θ-values in our set equal to θ^* compared to those equal to $\theta^{(t-1)}$ should be $p(\theta^*|\boldsymbol{y})/p(\theta^{(t-1)}|\boldsymbol{y}) = r$. This means that for every instance of $\theta^{(t-1)}$, we should have only a "fraction" of an instance of a θ^* value.
  - Procedure: Set $\theta^{(t)} = \theta^*$ with probability r (accept θ^*), and set $\theta^{(t)} = \theta^{(t-1)}$ with probability $1 - r$ (reject θ^*).

7.2 Metropolis-Hastings Algorithm

Why It Works

- We briefly provide a rigorous justification of the MH algorithm.
- The MH algorithm generates a sequence of numbers

$$\theta^{(0)} \to \theta^{(1)} \to \theta^{(2)} \to \dots$$

  and all θ values can lie in a continuous parameter space Θ.
- The discrete Markov chain we discussed before (in which the state space is discrete such as "1,2,3,4,5,6") is driven by the transition matrix P. Similarly, the Markov chain here is driven by a transition kernel, or a Markov kernel, say $K(\theta_a, \theta_b)$, for any $\theta_a, \theta_b \in \Theta$.
- Given θ_a, $K(\theta_a, \cdot)$ is a density function over its second argument, i.e. $\int_\Theta K(\theta_a, \theta) \mathrm{d}\theta = 1$.

7.2 Metropolis-Hastings Algorithm

Why It Works

- We need to guarantee that the MH chain eventually converges to the correct stationary distribution, which should be the posterior distribution $p(\theta|\boldsymbol{y})$.
- Similar to the equation $\boldsymbol{\pi}P = \boldsymbol{\pi}$ in the discrete case, $p(\theta|\boldsymbol{y})$ is the stationary distribution of the Markov chain with transition kernel $K(\theta_a, \theta_b)$, if and only if it satisfies that for any $\theta \in \Theta$,

$$\int_{\Theta} p(\theta_a|\boldsymbol{y})K(\theta_a, \theta_b)\mathrm{d}\theta_a = p(\theta_b|\boldsymbol{y}). \tag{1}$$

- We can easily verify that one sufficient condition for (1) to hold true is the so-called detailed balance condition: For any $\theta_a, \theta_b \in \Theta$,

$$p(\theta_a|\boldsymbol{y})K(\theta_a, \theta_b) = p(\theta_b|\boldsymbol{y})K(\theta_b, \theta_a). \tag{2}$$

7.2 Metropolis-Hastings Algorithm

Why It Works

- Question: What is the transition kernel in the MH algorithm?
- We can show that the transition kernel for the MH algorithm here is

$$K(\theta_a, \theta_b) = g(\theta_b|\theta_a)\alpha(\theta_a, \theta_b)\mathbb{1}(\theta_a \neq \theta_b) + \left[1 - \int_{\Theta} g(\theta_b|\theta_a)\alpha(\theta_a, \theta_b)\mathrm{d}\theta_b\right] \mathbb{1}(\theta_a = \theta_b), \quad (3)$$

  where g is the proposal density, $\mathbb{1}(A)$ is the indicator function that the event A happens, and $\alpha(\theta_a, \theta_b)$ is the acceptance probability

$$\alpha(\theta_a, \theta_b) = \min\left\{\frac{p(\theta_b|\boldsymbol{y})g(\theta_a|\theta_b)}{p(\theta_a|\boldsymbol{y})g(\theta_b|\theta_a)}, \ 1\right\}.$$

- The first term in (3) describes an acceptance move from θ_a to a different parameter value θ_b. The second term in (3) describes a rejection move to stay at θ_a.

7.2 Metropolis-Hastings Algorithm

Why It Works

Proof of detailed balance condition in (2):

- If $\theta_a = \theta_b$, the first term in (3) is zero. And the second term in (3) becomes $1 - \int_\Theta g(\theta_a|\theta_a)\alpha(\theta_a, \theta_a)\mathrm{d}\theta_a = 1 - \int_\Theta g(\theta_a|\theta_a)\mathrm{d}\theta_a$. So by symmetry,

$$p(\theta_a|\boldsymbol{y})K(\theta_a, \theta_b) = p(\theta_a|\boldsymbol{y})\left[1 - \int_\Theta g(\theta_a|\theta_a)\mathrm{d}\theta_a\right]$$
$$= p(\theta_b|\boldsymbol{y})\left[1 - \int_\Theta g(\theta_b|\theta_b)\mathrm{d}\theta_b\right] = p(\theta_b|\boldsymbol{y})K(\theta_b, \theta_a).$$

- If $\theta_a \neq \theta_b$, then the second term in (3) is zero, and

$$p(\theta_a|\boldsymbol{y})K(\theta_a, \theta_b) = p(\theta_a|\boldsymbol{y})g(\theta_b|\theta_a)\alpha(\theta_a, \theta_b)$$
$$= p(\theta_a|\boldsymbol{y})g(\theta_b|\theta_a) \times \min\left\{\frac{p(\theta_b|\boldsymbol{y})g(\theta_a|\theta_b)}{p(\theta_a|\boldsymbol{y})g(\theta_b|\theta_a)},\ 1\right\}$$
$$= \min\left\{p(\theta_b|\boldsymbol{y})g(\theta_a|\theta_b),\ p(\theta_a|\boldsymbol{y})g(\theta_b|\theta_a)\right\}$$
$$= p(\theta_b|\boldsymbol{y})g(\theta_a|\theta_b) \times \min\left\{1,\ \frac{p(\theta_a|\boldsymbol{y})g(\theta_b|\theta_a)}{p(\theta_b|\boldsymbol{y})g(\theta_a|\theta_b)}\right\}$$
$$= p(\theta_b|\boldsymbol{y})g(\theta_a|\theta_b)\alpha(\theta_b, \theta_a) = p(\theta_b|\boldsymbol{y})K(\theta_b, \theta_a).$$

This has proved that the posterior $p(\theta|\boldsymbol{y})$ is the stationary distribution.

Outline

7.3 R Implementation

- The R functions `rwmetrop` and `indepmetrop` in the `LearnBayes` package implement, respectively, the random walk and independence Metropolis-Hastings algorithms for special choices of proposal densities.
- The usage of these functions are

```
rwmetrop(logpost,proposal,start,m,...)
indepmetrop(logpost,proposal,start,m,...)
```

where

`logpost` : a function that computes the log posterior density up to an additive constant (not depending on θ)

`proposal` : a list containing the parameters of the proposal density

`start` : a starting value for θ

`m` : the number of samples desired

`...` : any data used in the function `logpost`

7.3 R Implementation

- For `rwmetrop`, the proposal density has the form

$$\theta^* = \theta^{(t-1)} + \texttt{scale}Z$$

  where $Z \sim \mathrm{N}(0, V)$ and `scale` is a positive scale parameter. Thus the `proposal` should be a list containing `var`, the covariance matrix V, and `scale`.

- For `indepmetrop`, the proposal density for θ is multivariate normal with mean μ and covariance matrix V. In this case the `proposal` should be a list containing `mu`, the mean μ and `var`, the covariance matrix V of the normal proposal density.

- The output of these functions has two components:

  `par` : matrix of simulated draws; each row is a draw of θ,
  `accept` : acceptance rate of the algorithm.

7.3 R Implementation

General Tricks for MH Algorithm

- Acceptance rate is the average number of accepted moves in the MH algorithm. It can be easily obtained empirically from the MCMC simulation.

- Desirable features of the proposal density depend on the MCMC algorithm employed. For an independence chain, we desire that the proposal density $g(\theta)$ approximate the posterior density $p(\theta|\boldsymbol{y}) \propto f(\theta)$, suggesting a high acceptance rate. But, as in rejection sampling, it is important that the ratio $f(\theta)/g(\theta)$ be bounded, especially in the tail portion of the posterior density. This means that one may choose a proposal $g(\theta)$ that is more diffuse than the posterior, resulting in a lower acceptance rate.

- For random walk chains with normal proposal densities, it has been suggested that acceptance rates between 20% and 50% are good. The "best" choice of acceptance rate ranges from 45% for one and two parameters to 23% for problems with more parameters.

7.3 R Implementation

General Tricks for MH Algorithm

- Initial value $\theta^{(0)}$: We can first find the posterior mode $\widehat{\theta}$ by maximizing $\log p(\theta|\boldsymbol{y})$, if it is possible. Using $\widehat{\theta}$ as $\theta^{(0)}$ is usually a good idea. This can be helpful when θ has a large dimension, because in that case, a randomly picked initial value $\theta^{(0)}$ can be very far away from the region where most of the posterior mass is located. A side product from the maximization of $\log p(\theta|\boldsymbol{y})$ is the Hessian matrix (denoted by H) at $\widehat{\theta}$. H is usually a negative definite matrix.

- Normal proposal kernel: If a normal kernel $g(\theta^*|\theta^{(t-1)})$ is used, and if the dimension of θ is at least 3, then the optimal acceptance rate of a random walk Metropolis algorithm is 0.234 (Roberts, Gelman and Gilks 1997). This can be achieved by setting $g(\theta^*|\theta^{(t-1)})$ as the density of $N(\theta^{(t-1)}, c^2(-H)^{-1})$, where H is the Hessian matrix), $c = 2.4/\sqrt{d}$, where d is the dimension of θ (and $d \geq 3$).

7.3 R Implementation

General Tricks for MH Algorithm

- Parameter space: If we use the random walk Metropolis algorithm with a normal proposal density g (as in most applications), it is better to transform all parameters in θ such that they lie in $\mathbb{R}$ unbounded.

- For example, in the normal model in Chapter 3, the variance parameter σ^2 is defined only on $(0, +\infty)$. We can use the log transformation $\eta = \log \sigma^2$ such that $\eta \in (-\infty, +\infty)$, and we perform the MH algorithm on η instead of σ^2. This can help us avoid getting negative values for σ^2.

Outline

7.4 MCMC Output Analysis

- In practice, one monitors the performance of an MCMC algorithm by inspecting the acceptance rate, constructing graphs, and computing diagnostic statistics on the stream of simulated draws. We call this investigation an MCMC output analysis.

- By means of this exploratory analysis, one decides if the chain has sufficiently explored the entire posterior distribution (there is good mixing) and the sequence of draws has approximately converged.

- A sufficient number of draws is required so that one can accurately estimate any particular summary of the posterior of interest.

- We discuss briefly some of the important issues in interpreting MCMC output and describe a few graphical and numerical diagnostics for assessing convergence.

7.4 MCMC Output Analysis

Burn-in

- One issue in understanding MCMC output is detecting the size of the burn-in period.
- The simulated values of θ obtained at the beginning of an MCMC run are not distributed from the posterior distribution. However, after some number of iterations have been performed (the burn-in period), the effect of the initial values wears off and the distribution of the new iterates approaches the true posterior distribution.

7.4 MCMC Output Analysis

Burn-in

- One way of estimating the length of the burn-in period is to examine trace plots of simulated values of a component or particular function of θ against the iteration number.
- In some cases, the first half of each sequence is discarded to diminish the effect of the starting distribution and inference is based only on the second half.
- Trace plots are especially important when MCMC algorithms are initialized with parameter values that are far from the center of the posterior distribution.

7.4 MCMC Output Analysis

Autocorrelation

- A second concern in analyzing output from MCMC algorithms is the degree of autocorrelation in the sampled values.
- In an MCMC algorithm, the simulated value of θ at the $(t+1)$th iteration is dependent on the simulated value at the tth iteration.
- If there is strong correlation between successive values in the chain, then two consecutive values provide only marginally more information about the posterior distribution than a single simulated draw.
- A strong correlation between successive iterates may also prevent the algorithm from exploring the entire region of the parameter space.

7.4 MCMC Output Analysis

Autocorrelation

- To assess how much correlation there is in the chain, we often compute the sample autocorrelation function. For a sequence of draws $\{\theta_i^{(1)}, \ldots, \theta_i^{(T)}\}$, the lag-$L$ autocorrelation function estimates the correlation between elements of the sequence that are L steps apart:

$$\text{acf}_L(\{\theta_i^{(1)}, \ldots, \theta_i^{(T)}\}) = \frac{\sum_{t=1}^{T-L}(\theta_i^{(t)} - \bar{\theta}_i)(\theta_i^{(t+L)} - \bar{\theta}_i)}{\sum_{t=1}^{T}(\theta_i^{(t)} - \bar{\theta}_i)^2},$$

where $\bar{\theta}_i = \frac{1}{T}\sum_{t=1}^{T}\theta_i^{(t)}$.

- A standard approach is to plot the values of the autocorrelation against the lag L. If the chain is mixing adequately, the values of the autocorrelation will decrease to zero as the lag value is increased. Autocorrelation can be computed in R using `acf`.

7.4 MCMC Output Analysis

- Another issue that arises in output analysis is the choice of the simulated sample size and the resulting accuracy of calculated posterior summaries.

- Suppose we estimate the posterior mean of θ_i with the sample mean

$$\bar{\theta}_i = \frac{1}{T}\sum_{t=1}^{T} \theta_i^{(t)}.$$

- If $\{\theta_i^{(1)}, \ldots, \theta_i^{(T)}\}$ were independent Monte Carlo samples, then the standard error of $\bar{\theta}_i$ is $\sqrt{\text{Var}(\theta_i)/T}$ and an estimate is given by

$$\sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(\theta_i^{(t)} - \bar{\theta}_i)^2/T}.$$

- However, since the iterates in an MCMC algorithm are not independent, one cannot use standard "independent sample" methods to compute estimated standard errors.

7.4 MCMC Output Analysis

- One simple method of computing standard errors for correlated output is the method of batch means.
- In the batch means method, the stream of simulated draws $\{\theta_i^{(t)}\}$ is subdivided into b batches, each batch of size v, where $T = bv$.
- In each batch, we compute a sample mean; call the set of sample means $\bar{\theta}_i^1, \ldots, \bar{\theta}_i^b$. If the lag one autocorrelation in the sequence in the batch means is small, then we can approximate the standard error of the estimate $\bar{\theta}_i$ by the standard deviation of the batch means divided by the square root of the number of batches.
- The batch standard error can be computed via the function `batchSE` in the `coda` package.

Outline

7.5 Example: Learning from grouped data

- Let us assume that the heights of men from a local college are normally distributed with mean μ and standard deviation σ.
- A random sample of 211 men is taken and data about their heights is summarized in the Table below.

Height (inches)	Frequency
less than 66	14
between 66 and 68	30
between 68 and 70	49
between 70 and 72	70
between 72 and 74	33
over 74	15

7.5 Example: Learning from grouped data

- We are observing multinomial data with unknown bin probabilities $p_1, \ldots, p_6$, where the probabilities are functions of (μ, σ). For example,

$$\begin{aligned}&\text{P(a male student's height is between 66 and 68 inches)}\\ &= p_2 = \Phi\left(\tfrac{68-\mu}{\sigma}\right) - \Phi\left(\tfrac{66-\mu}{\sigma}\right),\end{aligned}$$

  where $\Phi(\cdot)$ is the cdf a standard normal.

- The likelihood of the normal parameters given this grouped data is

$$\begin{aligned}L(\mu,\sigma) = p(\text{data}|\mu,\sigma) \propto{}& \Phi\left(\tfrac{66-\mu}{\sigma}\right)^{14}\left[\Phi\left(\tfrac{68-\mu}{\sigma}\right) - \Phi\left(\tfrac{66-\mu}{\sigma}\right)\right]^{30}\\ &\times\left[\Phi\left(\tfrac{70-\mu}{\sigma}\right) - \Phi\left(\tfrac{68-\mu}{\sigma}\right)\right]^{49}\left[\Phi\left(\tfrac{72-\mu}{\sigma}\right) - \Phi\left(\tfrac{70-\mu}{\sigma}\right)\right]^{70}\\ &\times\left[\Phi\left(\tfrac{74-\mu}{\sigma}\right) - \Phi\left(\tfrac{72-\mu}{\sigma}\right)\right]^{33}\left[1 - \Phi\left(\tfrac{74-\mu}{\sigma}\right)\right]^{15}.\end{aligned}$$

7.5 Example: Learning from grouped data

- Assume the noninformative prior $p(\mu, \sigma) \propto \frac{1}{\sigma}$. Then the posterior is

$$p(\mu, \sigma|\text{data}) \propto p(\text{data}|\mu, \sigma)\frac{1}{\sigma} = L(\mu, \sigma)\frac{1}{\sigma}.$$

- Let us transform the positive σ to the real line by $\lambda = \log(\sigma)$. Since $\sigma = \exp(\lambda)$ and $\frac{d\sigma}{d\lambda} = \exp(\lambda)$, the posterior of θ is

$$p(\theta|\text{data}) \propto L(\mu, \exp(\lambda)).$$

- Let us first create the matrix `data` containing the observations. Each row corresponds to a class; the first two columns contains the left and right limits respectively and the third column contains the frequency.

```
data <- cbind( c(-Inf,66,68,70,72,74),
               c(66,68,70,72,74,Inf),
               c(14,30,49,70,33,15) )
```

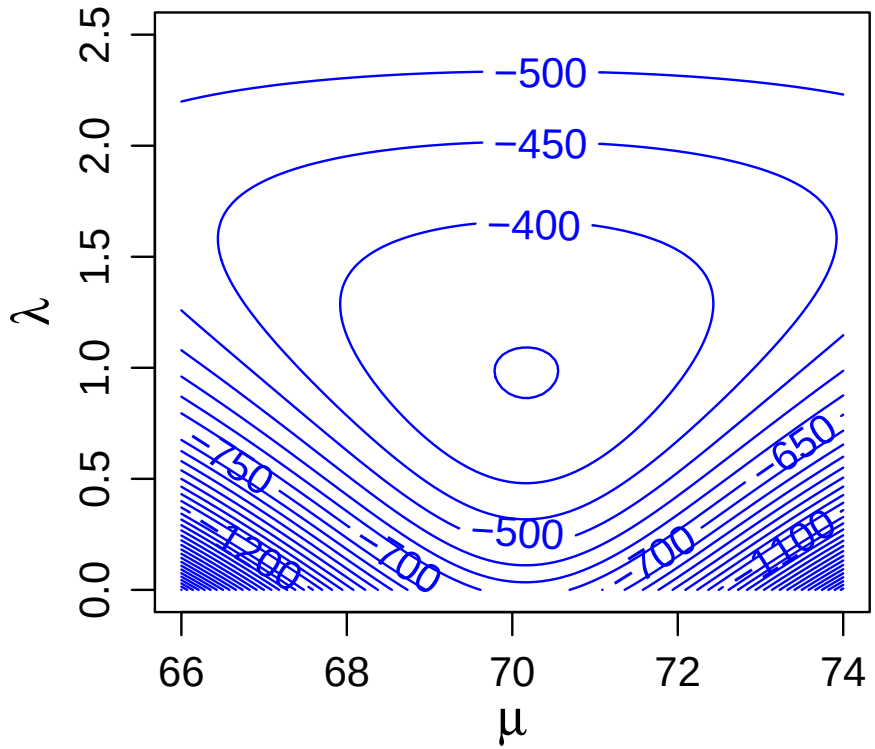
7.5 Example: Learning from grouped data

- Next, we write the function `logpost`, which computes $\log L(\mu, \exp(\lambda))$.

```
dpnorm <- function(a,b,mu,sigma){
    pnorm(b,mean=mu,sd=sigma) - pnorm(a,mean=mu,sd=sigma)
}
logpost <- function(theta, data){
    L <- 0
    G <- nrow(data)
    mu <- theta[1]
    sigma <- exp(theta[2])
    for (g in 1:G){
        L <- L + data[g,3]*
        log(dpnorm(data[g,1],data[g,2],mu,sigma))
    }
    return(L)
}
```

7.5 Example: Learning from grouped data

- We first find a normal approximation to the posterior and then use it to construct a proposal density for the Metropolis-Hastings algorithm.
- A contour plot of `logpost` reveals that the posterior mode is close to (70, 1). We will use this point as the initial value in the `optim` function.



7.5 Example: Learning from grouped data

- Using the `optim` function, we find the posterior mode and compute the covariance matrix of the normal approximation.

```
> start <- c(70,1)
> out <- optim(par=start,fn=logpost,hessian=TRUE,
+          control=list(fnscale=-1),data=data)
> (post.mode <- out$par)
[1] 70.169880  0.973644
> (post.cov <- -solve(out$hessian))
              [,1]          [,2]
[1,] 3.534713e-02 3.520776e-05
[2,] 3.520776e-05 3.146470e-03
```

- We use the normal approximation to design a Metropolis random walk algorithm. For the proposal density, we use the covariance matrix of the normal approximation and set the scale parameter as 2.

7.5 Example: Learning from grouped data

- We run 10,000 iterations of the random walk algorithm starting at `start`.
- The output `fit1` is a list with two components: `par` is a matrix of simulated values where each row corresponds to a single draw of θ, and `accept` gives the acceptance rate of the random walk chain.
- Here, the acceptance rate is 0.2936, which is close to the desired acceptance rate for this Metropolis random walk algorithm.

```
> require(LearnBayes)
> proposal <- list(var=post.cov,scale=2)
> set.seed(4234)
> fit1 <- rwmetrop(logpost,proposal,start,10000,data)
> fit1$accept
[1] 0.2936
```

7.5 Example: Learning from grouped data

- We can summarize the parameters μ and λ by computing the posterior means and standard deviations using the last 5000 simulated draws.

```
> (post.means <- apply(fit1$par[5001:10000,],2,mean))
[1] 70.1696631  0.9786188
> (post.sds <- apply(fit1$par[5001:10000,],2,sd))
[1] 0.19253831 0.05757205
```

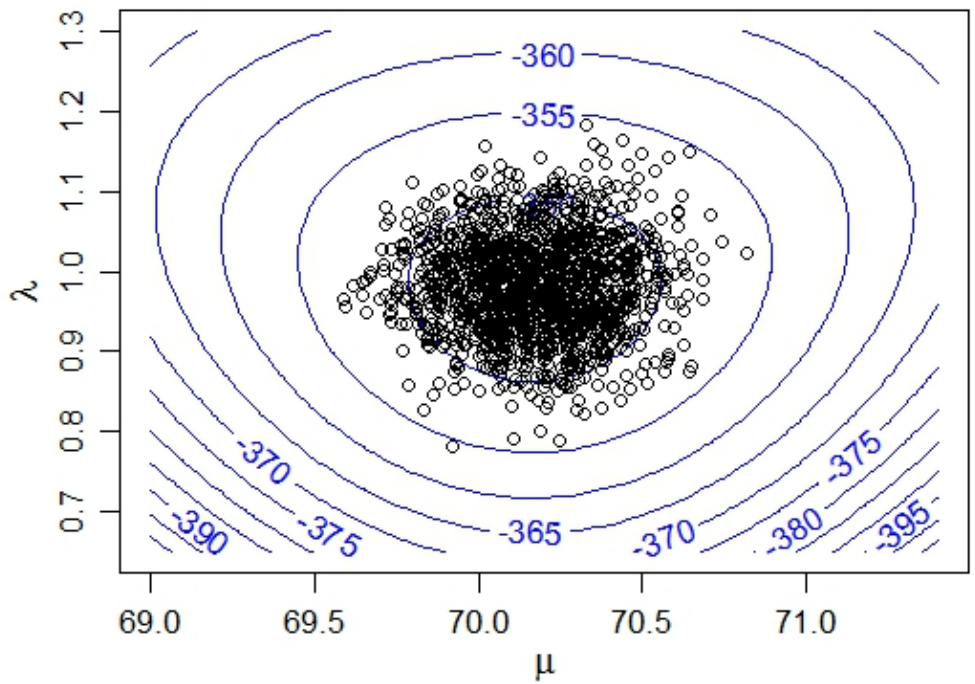
  The corresponding values in the normal approximation are

```
> post.mode
[1] 70.169880  0.973644
> sqrt(diag(post.cov))
[1] 0.18800834 0.05609341
```

- There is close agreement between the two sets of posterior moments which indicates that the normal approximation is reasonably accurate.

7.5 Example: Learning from grouped data

- The figure below shows a contour plot of the joint posterior of μ and λ with the last 5000 simulated draws from the random walk Metropolis algorithm drawn on top.
- The contour lines have an elliptical shape that confirms the accuracy of the normal approximation in this example.



7.5 Example: Learning from grouped data

- We illustrate MCMC output analysis using the R package coda.
- Suppose we rerun the Metropolis random walk algorithm with poor choices of starting value and proposal density.
- As a starting value, we choose $(\mu, \lambda) = (65, 1)$ (choice of μ is too small) and the small scale factor of 0.2 instead of 2. Hereafter, we refer to this modified algorithm as "Algorithm 2" and the original algorithm as "Algorithm 1".

```
> start <- c(65,1)
> proposal <- list(var=post.cov,scale=0.2)
> set.seed(4234)
> fit2 <- rwmetrop(logpost,proposal,start,10000,data)
> fit2$accept
[1] 0.887
```

7.5 Example: Learning from grouped data

- The acceptance rate of Algorithm 2 is 0.89, which is much larger than the 0.29 rate that we found using the scale factor 2.
- To analyze the draws using `coda`, we first convert the simulated values into MCMC objects. The function `mcmc` in `coda` takes as input a vector or a matrix where each column corresponds to a different variable.

```
require(coda)
colnames(fit1$par) <- c("mu","lambda")
colnames(fit2$par) <- c("mu","lambda")
mcmcobj1 <- mcmc(fit1$par)
mcmcobj2 <- mcmc(fit2$par)
```

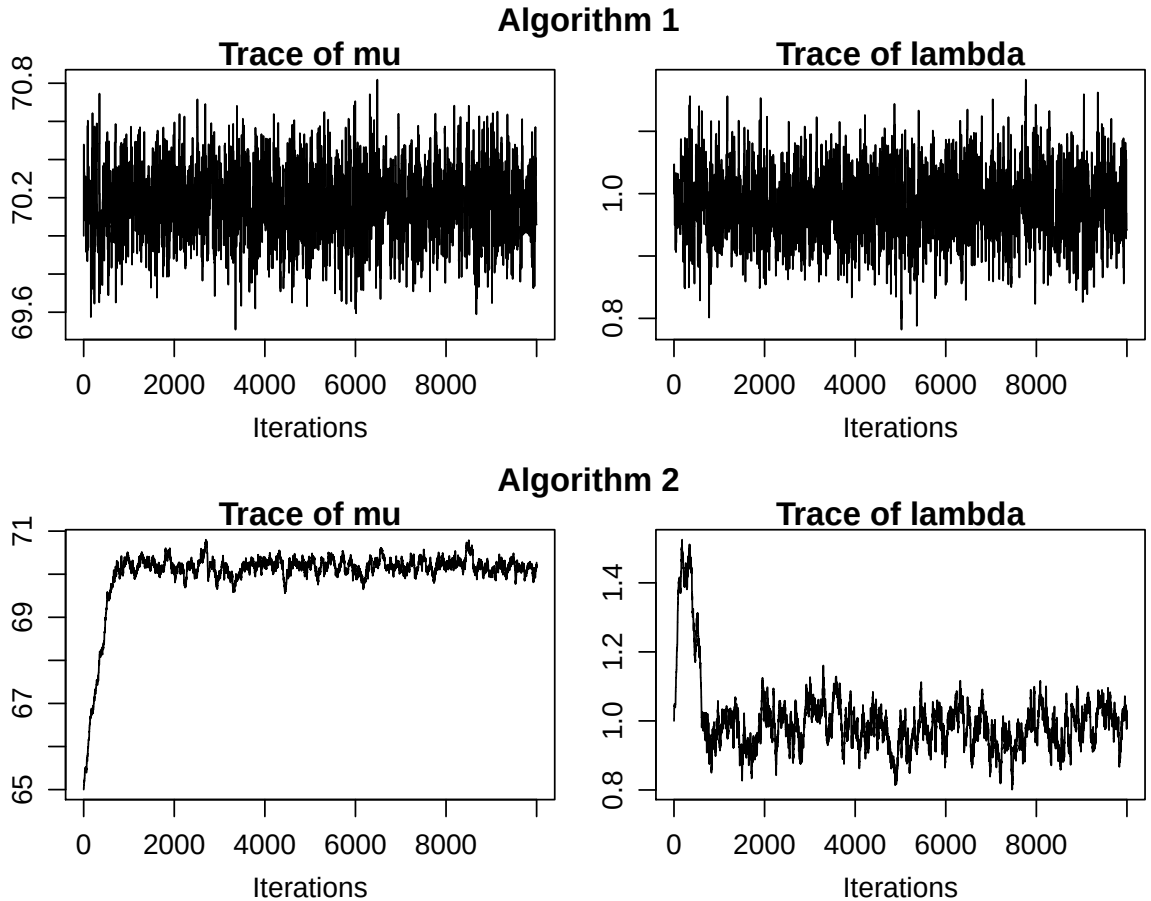
The MCMC objects `mcmcobj1` and `mcmcobj2` correspond to Algorithms 1 and 2 respectively.

7.5 Example: Learning from grouped data

- Trace plots of the simulated draws of μ and λ can be obtained using the `traceplot` function in `coda`.

```
par(mfrow=c(1,2),
  oma=c(0,0,1,0))
traceplot(mcmcobj1)
title("Algorithm 1",
  outer=TRUE)

par(mfrow=c(1,2),
  oma=c(0,0,1,0))
traceplot(mcmcobj2)
title("Algorithm 2",
  outer=TRUE)
```

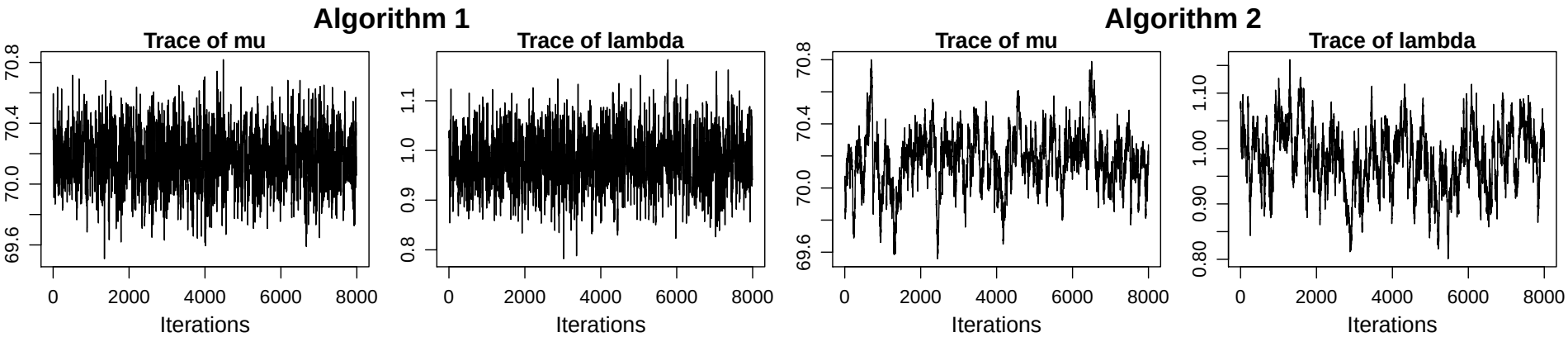


7.5 Example: Learning from grouped data

- The original algorithm converges quickly while convergence is slow for the modified algorithm due to the poor starting values.
- Let us discard the first 2000 iterations as burn-in for both algorithms and look at the trace plots again.

```
mcmcobj1 <- mcmc(fit1$par[2001:10000,])
mcmcobj2 <- mcmc(fit2$par[2001:10000,])
```

7.5 Example: Learning from grouped data

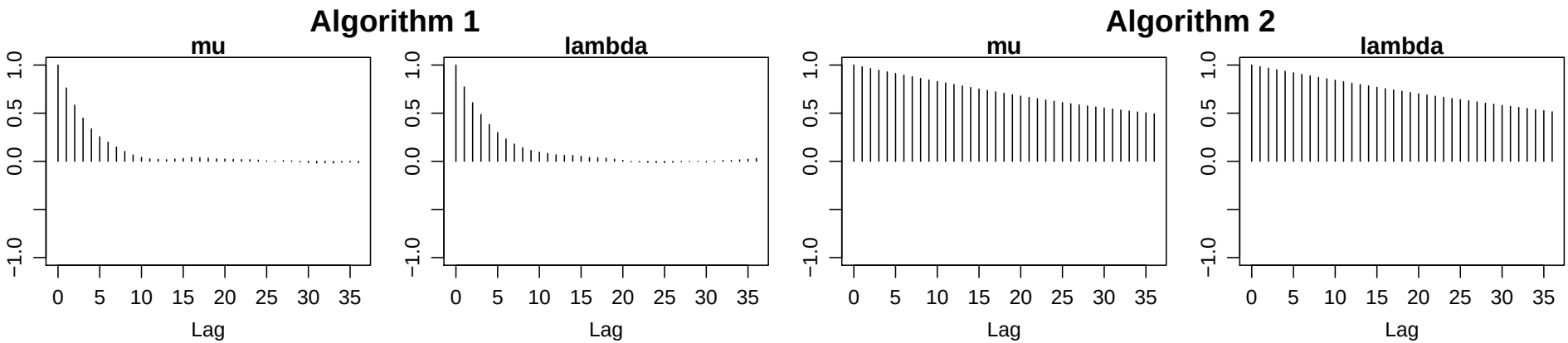


- The trace plot of the simulated streams of μ and λ look more like random noise for Algorithm 1.
- For Algorithm 2, the simulated draws appear to have reached the stationary distribution. However the simulated sequence appears irregular; e.g. the iterates will explore the region $\mu > 70.5$ for a while before returning to the center of the distribution.

7.5 Example: Learning from grouped data

- Autocorrelation plots can be produced by the `autocorr.plot` function in `coda`.

```
autocorr.plot(mcmcobj1)
autocorr.plot(mcmcobj2)
```



- The simulated sequences in Algorithm 2 has strong correlation structure; the autocorrelations are close to one for lag one and reduce very slowly as a function of the lag. For Algorithm 1, the lag one autocorrelations are high, but the autocorrelation values dissipate rapidly as a function of the lag.

7.5 Example: Learning from grouped data

- Summary statistics of the simulated draws can be obtained by taking the `summary` of the MCMC objects.

```
> summary(mcmcobj1)
Iterations = 1:8000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 8000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean     SD  Naive SE Time-series SE
mu     70.1683 0.1899 0.0021228       0.005829
lambda  0.9803 0.0569 0.0006361       0.001824

2. Quantiles for each variable:

          2.5%     25%     50%    75%  97.5%
mu     69.8056 70.0409 70.1677 70.290 70.566
lambda  0.8707  0.9401  0.9788  1.018  1.091
```

7.5 Example: Learning from grouped data

```
> summary(mcmcobj2)

Iterations = 1:8000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 8000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

          Mean      SD  Naive SE Time-series SE
mu     70.1703 0.19690 0.0022014       0.025003
lambda  0.9897 0.06571 0.0007347       0.009397

2. Quantiles for each variable:

          2.5%     25%     50%    75%  97.5%
mu     69.7920 70.0269 70.1674 70.311 70.545
lambda  0.8613  0.9452  0.9877  1.035  1.118
```

7.5 Example: Learning from grouped data

- The `Naive SE` in the summary statistics assumes that the sampled values are independent and is computed by taking the standard deviation of the iterates (computed using `sd`) divided by the square root of the number of iterates.

- The `Time-series SE` take into account the autocorrelation among the iterates and is computed by taking the standard deviation of the iterates divided by the "effective sample size" (computed using `effectiveSize` in `coda`). The "effective sample size" can be interpreted as the number of independent Monte Carlo samples necessary to give the same precision as the MCMC samples.

- The graphs and the summary statistics confirm the better performance of the MCMC chain with a starting value $(\mu, \lambda) = (70, 1)$ and scale factor of 2 (i.e. "Algorithm 1").

7.5 Example: Learning from grouped data

Direct Coding of MH Algorithm

- Although it is convenient, we do not have to fully rely on the R function `rwmetrop` to code the Metropolis-Hastings algorithm.
- The MH algorithm is simple to code using a generic `for` loop. The only things we need to do is to define a log posterior function, and generate new proposals from a proposal density g.
- At the end of the file "Chapter7_Rcodes.r", I have included a direct R coding of random walk Metropolis algorithm for the grouped data example, realizing the same functionality as `rwmetrop`. It takes slightly longer time to run due to the `for` loop. You can easily adapt this example to other examples.