Chapter 7

MCMC algorithms

7.1 Introduction

In a complicated Bayesian statistical model it may be very difficult to analyze the mathematical form of the posterior and it may be very difficult to draw an i.i.d. sample from it. Fortunately, it is often easy to generate a correlated sample, which approximately comes from the posterior distribution. (In this context, the word *correlated* means *not independent*). However, we would very much prefer to have an i.i.d. sample from the posterior, instead. After one has available a sample, one can estimate posterior expectations and posterior quantiles using the same kind of techniques that are used with i.i.d. samples. This is the idea behind Markov chain Monte Carlo (MCMC) methods.

In this chapter we will introduce the basic MCMC sampling algorithms that are used in practical problems. The emphasis is on trying to understand what one needs to do in order to implement the algorithms. In Chapter 11 we will see why these algorithms work using certain concepts from the theory of Markov chains in a general state space.

There are available computer programs that can implement an MCMC simulation automatically. Perhaps the most famous such program is the BUGS system (Bayesian inference Using Gibbs Sampling), which has several concrete implementations, most notably WinBUGS and OpenBUGS. You can analyze most of the models of interest easily using BUGS. What the user of BUGS needs to do is to write the description of the model in a format that BUGS understands, read the data into the program, and then let the program do the simulation. Once the simulation has finished, one can let the program produce various summaries of the posterior. Using such a tool, it is simple to experiment with different priors and different likelihoods for the same data.

However, in this chapter the emphasis is on understanding how you can write your own MCMC programs. Why would this be of interest?

- If you have not used MCMC before, you get a better understanding of the methods if you try to implement (some of) them yourself.
- For some models, the automated tools fail. Sometimes you can, however, rather easily design and implement a MCMC sampler yourself, once you understand the basic principles. (In some cases, however, designing an efficient MCMC sampler can be an almost impossibly difficult task.)

- Sometimes you want to have more control over the sampling algorithm than is provided by the automated tools. In some cases implementation details can make a big difference to the efficiency of the method.

The most famous MCMC methods are the Metropolis–Hastings sampler and the Gibbs sampler. Where do these names come from?

- Nicholas (Nick) Metropolis (1915–1999) was an American mathematician, physicist and pioneer of computing, who was born in Greece. He published the Metropolis sampler in 1953 jointly with two husband-and-wife teams, namely A.W. and M.N. Rosenbluth and A.H. and E. Teller. At that time the theory of general state space Markov chains was largely unexplored. In spite of this, the authors managed to give a heuristic proof for the validity of the method.

- W. Keith Hastings (1930–) is a Canadian statistician, who published the Metropolis–Hastings sampler in 1970. It is a generalization of the Metropolis sampler. Hastings presented his algorithm using a discrete state space formalism, since the theory of general state space Markov chains was then known only to some specialists in probability theory. Hastings' article did not have a real impact on statisticians until much later.

- The name Gibbs sampler was introduced by the brothers S. and D. Geman in an article published in 1984. Related ideas were published also by other people at roughly the same time. The method is named after the American mathematician and physicist J. Willard Gibbs (1893–1903), who studied thermodynamics and statistical physics, but did not have anything to do with MCMC.

In the late 1980's and early 1990's there was an explosion in the number of studies, where people used MCMC methods in Bayesian inference. Now there was available enough computing power to apply the methods, and besides, the theory of general state space Markov chains had matured so that readable expositions of the theory were available.

Nowadays, many statisticians routinely use the concept of a Markov chain which evolves in a general state space. Unfortunately, their mathematical theory is still explained only in a handful of text books.

7.2 Basic ideas of MCMC

MCMC algorithms are based on the idea of a Markov chain which evolves in discrete time. A Markov chain is a stochastic process

$$\theta^{(0)}, \theta^{(1)}, \theta^{(2)}, \ldots$$

Here $\theta^{(i)}$ (the state of the process at time i) is a RV whose values lie in a state space, which usually is a subset of some Euclidean space $\mathbb{R}^d$. The state space is the same for all times i. We write the time index as a superscript so that we can index the components $\theta^{(i)}$ using a subscript.

Markov chains have the following **Markov property**: the distribution of the next state $\theta^{(i+1)}$ depends on the history $\theta^{(0)}, \theta^{(1)}, \ldots, \theta^{(i)}$ only through the

present state $\theta^{(i)}$. The Markov chains used in MCMC methods are **homogeneous**: the conditional distribution of $\theta^{(i+1)}$ given $\theta^{(i)}$ does not depend on the index i.

The following algorithm shows how one can simulate a Markov chain. Intuitively, a Markov chain is nothing else but the mathematical idealization of this simulation algorithm.

Algorithm 14: Simulating a Markov chain.

1 Generate $\theta^{(0)}$ from a given initial distribution;
2 **for** $i = 0, 1, 2, \ldots$ **do**
3 Compute the next state $\theta^{(i+1)}$ using some rule, where you can use the present state $\theta^{(i)}$ (but no earlier states) and freshly generated random numbers.
4 **end**

If the rule for calculating the next state does not change depending on the value of the loop index i, then the generated Markov chain is homogeneous.

Some (but not all) Markov chains have an **invariant distribution** (or a stationary distribution or equilibrium distribution), which can be defined as follows. If the initial state of the chain $\theta^{(0)}$ follows the invariant distribution, then also all the subsequent states $\theta^{(i)}$ follow it.

If a Markov chain has an invariant distribution, then (under certain regularity conditions) the distribution of the state $\theta^{(i)}$ converges to that invariant distribution (in a certain sense). Under certain regularity conditions, such a chain is **ergodic**, which ensures that an arithmetic average (or an ergodic average) of the form

$$\frac{1}{N} \sum_{i=1}^{N} h(\theta^{(i)})$$

converges, almost surely, to the corresponding expectation calculated under the invariant distribution as $N \to \infty$. That is, the ergodic theorem for Markov chains then states that the strong law of large numbers holds, i.e.,

$$\lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} h(\theta^{(i)}) \to E_f h(\Theta) = \int h(\theta) f(\theta) \, \mathrm{d}\theta, \tag{7.1}$$

where f is the density of the invariant distribution. This will then hold for all functions h for which the expectation $E_f h(\Theta)$ exists, so the convergence is as strong as in the strong law of large numbers for i.i.d. sequences. There are also more advanced forms of ergodicity (geometric ergodicity and uniform ergodicity), which a Markov chain may either have or not have.

Under still more conditions, Markov chains also satisfy a central limit theorem, which characterizes the speed of convergence in the ergodic theorem. The central limit theorem for Markov chains is of the form

$$\sqrt{N} \left(\frac{1}{N} \sum_{i=1}^{N} h(\theta^{(i)}) - E_f h(\Theta) \right) \xrightarrow{\mathrm{d}} N(0, \sigma_h^2).$$

The speed of convergence is of the same order of N as in the central limit theorem for i.i.d. sequences. However, estimating the variance σ_h^2 in the central limit theorem is lot trickier than with i.i.d. sequences.

After this preparation, it is possible to explain the basic idea of MCMC methods. The idea is to set up an ergodic Markov chain which has the posterior distribution as its invariant distribution. Doing this is often surprisingly easy. Then one simulates values

$$\theta^{(0)}, \theta^{(1)}, \theta^{(2)}, \ldots$$

of the chain. When t is sufficiently large, then $\theta^{(t)}$ and all the subsequent states $\theta^{(t+i)}, i \geq 1$ follow approximately the posterior distribution. The time required for the chain to approximately achieve its invariant distribution is called the **burn-in**. After the initial burn-in period has been discarded, the subsequent values

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

can be treated as a dependent sample from the posterior distribution, and we can calculate posterior expectations, quantiles and other summaries of the posterior distribution based on this sample.

After the burn-in period we need to store the simulated values of the chain for later use. So, for a scalar parameter we need a vector to store the results, for a vector parameter we need a matrix to store the results and so on. To save space, one often decides to **thin** the sequences by keeping only every kth value of each sequence and by discarding the rest.

Setting up *some* MCMC algorithm for a given posterior is usually easy. However, the challenge is to find an MCMC algorithm which converges rapidly and then explores efficiently the whole support of the posterior distribution. Then one can get a reliable picture of the posterior distribution after stopping the simulation after a reasonable number of iterations.

In practice one may want to try several approaches for approximate posterior inference in order to become convinced that the posterior inferences obtained with MCMC are reliable. One can, e.g., study simplified forms of the statistical model (where analytical developments or maximum likelihood estimation or other asymptotic approximations to Bayesian estimation may be possible), simulate several chains which are initialized from different starting points and are possibly computed with different algorithms, and compute approximations to the posterior.

7.3 The Metropolis–Hastings algorithm

Now we consider a target distribution with density $\pi(\theta)$, which may be available only in an unnormalized form $\tilde{\pi}(\theta)$. Usually the target density is the posterior density of a Bayesian statistical model,

$$\pi(\theta) = p(\theta \mid y).$$

Actually we only need to know an unnormalized form of the posterior, which is given, e.g., in the form of prior times likelihood,

$$\tilde{\pi}(\theta) = p(\theta)\, p(y \mid \theta).$$

The density $\pi(\theta)$ may be a density in the generalized sense, so we may have a discrete distribution for some components of θ and a continuous distribution for others.

For the Metropolis–Hastings algorithm we need a proposal density $q(\theta' \mid \theta)$, from which we are able to simulate. (Some authors call the proposal density the jumping density or candidate generating density.) As a function of θ', the proposal density $q(\theta' \mid \theta)$ is a density on the parameter space for each value of θ. When the current state of the chain is $\theta = \theta^{(i)}$, we propose a value for the next state from the distribution with density

$$\theta' \mapsto q(\theta' \mid \theta)$$

The proposed value θ' is then accepted or rejected in the algorithm. If the proposal is accepted, then the next state $\theta^{(i+1)}$ is taken to be θ', but otherwise the chain stays in the same state, i.e., $\theta^{(i+1)}$ is assigned the current state $\theta^{(i)}$.

The acceptance condition has to be selected carefully so that we get the target distribution as the invariant distribution of the chain. The usual procedure works as follows. We calculate the value of the Metropolis–Hastings ratio (M–H ratio)

$$r = r(\theta', \theta) = \frac{\pi(\theta')\, q(\theta \mid \theta')}{\pi(\theta)\, q(\theta' \mid \theta),} \tag{7.2}$$

where $\theta = \theta^{(i)}$ is the current state and θ' is the proposed state. Then we generate a value u from the standard uniform $\mathrm{Uni}(0, 1)$. If $u < r$, then we accept the proposal and otherwise reject it. For the analysis of the algorithm, it is essential to notice that the probability of accepting the proposed θ', when the current state is θ, is given by

$$\Pr(\text{proposed value is accepted} \mid \theta^{(i)} = \theta, \theta') = \min(1, r(\theta', \theta)). \tag{7.3}$$

We need here the minimum of one and the M–H ratio, since the M–H ratio may very well be greater than one.

Some explanations are in order.

- The denominator of the M–H ratio (7.2) is the joint density of the proposal θ' and the current state θ, when the current state already follows the posterior.
- The numerator is of the same form as the denominator, but θ and θ' have exchanged places.
- If $\pi(\theta^{(0)}) > 0$, then the denominator of the M–H ratio is always strictly positive during the algorithm. When $i = 0$ this follows from the observation that $q(\theta' \mid \theta^{(0)})$ has to be positive, since θ' is generated from that density. Also $\pi(\theta^{(1)})$ has to be positive, thanks to the form of the acceptance test. The rest follows by induction.
- We do not need to know the normalizing constant of the target distribution, since it cancels in the M–H ratio,

$$r = \frac{\pi(\theta')\, q(\theta \mid \theta')}{\pi(\theta)\, q(\theta' \mid \theta)} = \frac{\tilde{\pi}(\theta')\, q(\theta \mid \theta')}{\tilde{\pi}(\theta)\, q(\theta' \mid \theta)} \tag{7.4}$$

- If the target density is a posterior distribution, then the M–H ratio is given by

$$r = \frac{f_{Y|\Theta}(y \mid \theta')\, f_\Theta(\theta')\, q(\theta \mid \theta')}{f_{Y|\Theta}(y \mid \theta)\, f_\Theta(\theta)\, q(\theta' \mid \theta)}. \tag{7.5}$$

- Once you know what the notation is supposed to mean, you can use an abbreviated notation for the M–H ratio, such as

$$r = \frac{p(\theta' \mid y)\, q(\theta \mid \theta')}{p(\theta \mid y)\, q(\theta' \mid \theta)}.$$

  Here, e.g., $p(\theta' \mid y)$ is the value of the posterior density evaluated at the proposal θ'.

An explanation of why the target distribution is the invariant distribution of the resulting Markov chain will be given in Chapter 11. Then it will become clear, that other formulas in place of eq. (7.2) would work, too. However, the formula (7.2) is known to be optimal (in a certain sense), and therefore it is the one that is used in practice.

In the Metropolis–Hastings algorithm the proposal density can be selected otherwise quite freely, but we must be sure that we can reach (with positive probability) any reasonably possible region in the parameter space starting from any initial state $\theta^{(0)}$ with a finite number of steps. This property is called **irreducibility** of the Markov chain.

Algorithm 15: The Metropolis–Hastings algorithm.

Input: An initial value $\theta^{(0)}$ such that $\tilde{\pi}(\theta^{(0)}) > 0$ and the number of iterations N.

Result: Values simulated from a Markov chain which has as its invariant distribution the distribution corresponding to the unnormalized density $\tilde{\pi}(\theta)$.

1 **for** $i = 0, 1, 2, \ldots, N$ **do**

2 $\quad \theta \leftarrow \theta^{(i)}$;

3 $\quad$ Generate θ' from $q(\cdot \mid \theta)$ and u from $\mathrm{Uni}(0, 1)$;

4 $\quad$ Calculate the M–H ratio

$$r = \frac{\tilde{\pi}(\theta')\, q(\theta \mid \theta')}{\tilde{\pi}(\theta)\, q(\theta' \mid \theta)}$$

5 $\quad$ Set

$$\theta^{(i+1)} \leftarrow \begin{cases} \theta', & \text{if } u < r \\ \theta, & \text{otherwise.} \end{cases}$$

6 **end**

Algorithm 15 sums up the Metropolis–Hastings algorithm. When implementing the algorithm, one easily comes across problems, which arise because of underflow or overflow in the calculation of the M–H ratio r. Most of such problems can be cured by calculating with logarithms. E.g., when the target distribution is a posterior distribution, then one should first calculate $s = \log r$ by

$$\begin{aligned} s = {} & \log(f_{Y|\Theta}(y \mid \theta')) - \log(f_{Y|\Theta}(y \mid \theta)) \\ & + \log(f_\Theta(\theta')) - \log(f_\Theta(\theta)) + \log(q(\theta \mid \theta')) - \log(q(\theta' \mid \theta)) \end{aligned}$$

and only then calculate $r = \exp(s)$. Additionally, one might want cancel common factors from r before calculating its logarithm.

Implementing some Metropolis–Hastings algorithm for any given Bayesian statistical model is usually straightforward. However, finding a proposal distribution which allows the chain to converge quickly to the target distribution and allows it to explore the parameter space efficiently may be challenging.

7.4 Concrete Metropolis–Hastings algorithms

In the Metropolis–Hastings algorithm, the proposal θ' is in practice produced by running a piece of code, which can use the current state $\theta^{(i)}$, freshly generated random numbers from any distribution and arbitrary arithmetic operations. We must be able to calculate the density of the proposal θ', when the current state is equal to θ. This is then $q(\theta' \mid \theta)$, which we must be able to evaluate. Or at least we must be able to calculate the value of the ratio

$$q(\theta \mid \theta')/q(\theta' \mid \theta).$$

Different choices for the proposal density correspond to different choices for the needed piece of code. The resulting Metropolis–Hastings algorithms are named after the properties of the proposal distribution.

7.4.1 The independent Metropolis–Hastings algorithm

In the independent M–H algorithm (other common names: independence chain independence sampler), the proposal density is a fixed density, say $s(\theta')$, which does not depend on the value of the current state. In the corresponding piece of code, we only need to generate the value θ' from the proposal distribution.

If the proposal distribution happens to be the target distribution, then every proposal will be accepted, and as a result we will get an i.i.d. sample from the target distribution.

In order to to sample the target distribution properly with the independent M–H algorithm, the proposal density s must be positive everywhere, where the target density is positive. If there exist a majorizing constant M, such that

$$\pi(\theta) \leq Ms(\theta) \qquad \forall\theta,$$

then the resulting chain can be shown to have good ergodic properties, but if this condition fails, then the convergence properties of the chain can be bad. (In the independent M–H algorithm one does not need to know the value of M.) This implies that the proposal density should be such that the accept–reject method or importance sampling using that proposal distribution would be possible, too. In particular, the tails of the proposal density s should be at least as heavy as the tails of the target density. Finding such proposal densities may be difficult in high-dimensional problems.

7.4.2 Symmetric proposal distribution

If the proposal density is symmetric in that

$$q(\theta' \mid \theta) = q(\theta \mid \theta'), \qquad \forall\theta, \theta',$$

then the proposal density cancels from the M–H ratio,

$$r = \frac{\pi(\theta')\, q(\theta \mid \theta')}{\pi(\theta)\, q(\theta' \mid \theta)} = \frac{\pi(\theta')}{\pi(\theta)}.$$

This the sampling method that was originally proposed by Metropolis. Proposals leading to a higher value for the target density are automatically accepted, and other proposals may be accepted or rejected. Later Hastings generalized the method for non-symmetric proposal densities.

7.4.3 Random walk Metropolis–Hastings

Suppose that g is a density on the parameter space an that we calculate the proposal as follows,

$$\text{generate } w \text{ from density } g \text{ and set } \theta' \leftarrow \theta + w.$$

Then the proposal density is

$$q(\theta' \mid \theta) = g(\theta' - \theta).$$

This kind of a proposal is called a random walk proposal. If the density g is symmetric, i.e.,

$$g(-w) = g(w) \quad \forall w,$$

then the proposal density $q(\theta' \mid \theta)$ is also symmetric, and thus cancels from the M–H ratio. In the case of a symmetric random walk proposal, one often speaks of the random walk Metropolis (RWM) algorithm.

Actually, a random walk is a stochastic process of the form $X_{t+1} = X_t + w_t$, where the random variables w_t are i.i.d. Notice that the stochastic process produced by the random walk M–H algorithm is **not** a random walk, since the proposals can either be accepted or rejected.

The symmetric random walk Metropolis–Hastings algorithm (also known as the random walk Metropolis algorithm) is one of the most commonly used forms of the Metropolis–Hastings method. The most commonly used forms for g are the multivariate normal or multivariate Student's t density centered at the origin. This is, of course, appropriate only for continuous posterior distributions.

Often one selects the covariance matrix of the proposal distribution as

$$a\, C,$$

where C is an approximation to the covariance matrix of the target distribution (in Bayesian inference C is an approximation to the posterior covariance matrix) and the scalar a is a tuning constant which should be calibrated carefully. These kind of proposal distributions work well when the posterior distribution is approximately normal. One sometimes needs to reparametrize the model in order to make this approach work better.

The optimal value of a and the corresponding optimal acceptance rate has been derived theoretically, when the target density is a multivariate normal $N_d(\mu, C)$ and the random walk proposal is $N_d(0, aC)$, see [13]. The scaling constant a should be about $(2.38)^2/d$ when d is large. The corresponding acceptance rate (the number of accepted proposals divided by the total number

of proposals) is from around 0.2 (for high-dimensional problems) to around 0.4 (in dimensions one or two). While these results have been derived using the very restrictive assumption that the target density is a multivariate normal, the results anyhow give rough guidelines for calibrating a in a practical problem.

How and why should one try to control the acceptance rate in the random walk M–H algorithm? If the acceptance rate is too low, then the chain is not able to move, and the proposed updating steps are likely to be too large. In this case one could try a smaller value for a. However, a high acceptance rate may also signal a problem, since then the updating steps may be too small. This may lead to the situation where the chain explores only a small portion of the parameter space. In this case one should try a larger value for a. From the convergence point of view, too high acceptance rate is a bigger problem. A low acceptance rate is a problem only from the computing time point of view.

In order to calibrate the random walk M–H algorithm, one needs an estimate of its acceptance rate. A simple-minded approach is just to keep track of the number of accepted proposals. A better approach is to calculate the average of the acceptance probabilities,

$$\frac{1}{N}\sum_{i=1}^{n}\min(1, r_i),$$

where r_i is the M–H ratio in the ith iteration.

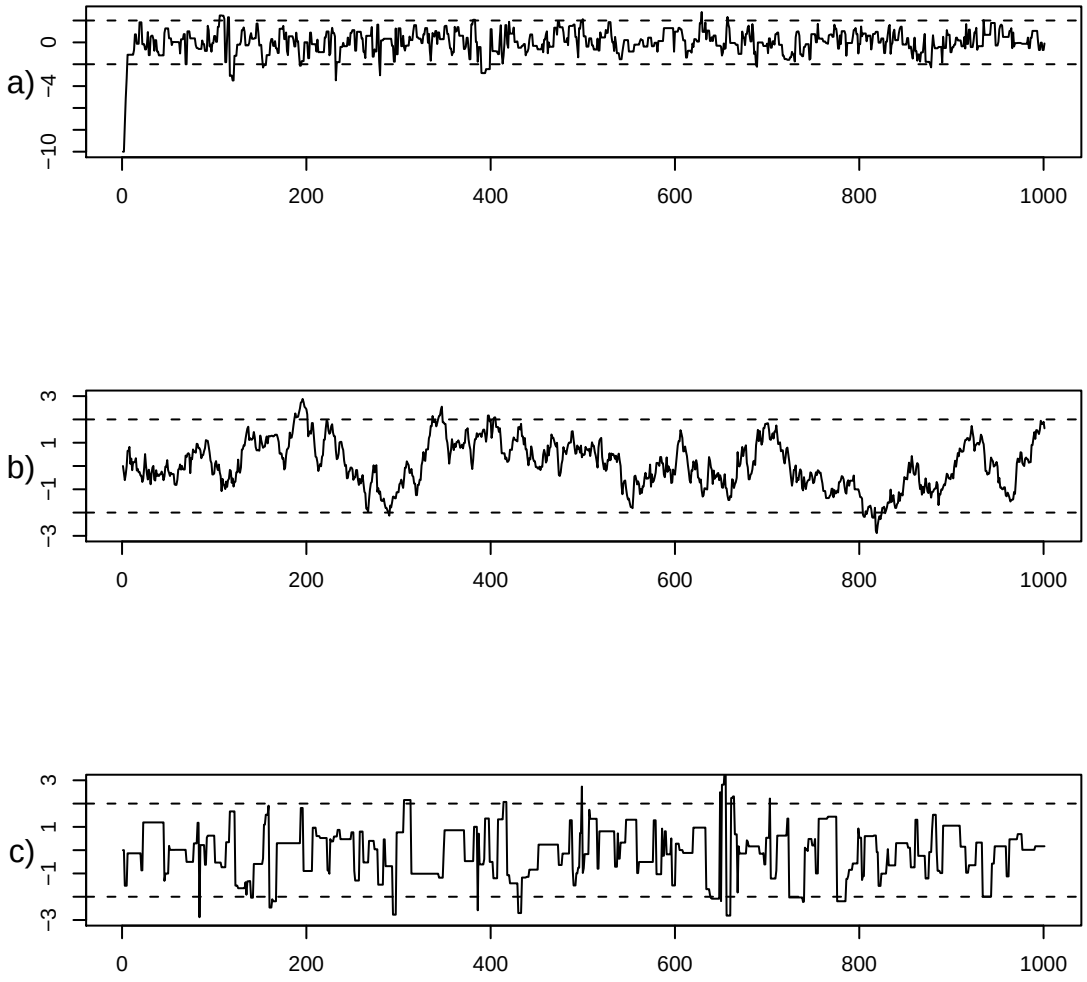
In practice, one can try to tune a iteratively, until the acceptance rate is acceptable. The tuning iterations are discarded, and the MCMC sample on which the inference is based is calculated using the fixed proposal distribution, whose scale a is the selected value. Fixing the proposal distribution is necessary, since the theory of the Metropolis–Hastings algorithm requires a homogeneous Markov chain, i.e., a proposal density $q(\theta' \mid \theta)$ which does not depend on the iteration index.

Recently, several researchers have developed adaptive MCMC algorithms, where the proposal distribution is allowed to change all the time during the iterations, see [1] for a review. Be warned that the design of valid adpative MCMC algorithms is subtle and that their analysis requires tools which are more difficult than the general state space Markov chain theory briefly touched upon in Chapter 11.

Example 7.1. Let us try the random walk chain for the target distribution $N(0,1)$ by generating the increment from the normal distribution $N(0,\sigma^2)$ using the following values for the variance: a) $\sigma^2 = 4$ b) $\sigma^2 = 0.1$ c) $\sigma^2 = 40$. In situation a) the chain is initialized far away in the tails of the target distribution, but nevertheless it quickly finds its way to the main portion of the target distribution and then explores it efficiently. Such a chain is said to **mix** well. In situations b) and c) the chains are initialized at the center of the target distribution, but the chains mix less quickly. In situation b) the step length is too small, but almost all proposals get accepted. In situation c) the algorithm proposes too large steps, almost all of which get rejected. Figure 7.1 presents trace plots (or time series plots) of the chain in the three situations.

△

Figure 7.1: Trace plots of the random walk chain using the three different proposal distributions.



7.4.4 Langevin proposals

Unlike a random walk, the Langevin proposals introduce a drift which moves the chain towards the modes of the posterior distribution. When the current state is θ, the proposal θ' is generated with the rule

$$\theta' = \theta + \frac{\sigma^2}{2}\nabla(\log \pi(\theta)) + \sigma\,\epsilon, \qquad \epsilon \sim N_p(0, I).$$

Here $\sigma > 0$ is a tuning parameter and

$$\nabla(\log \pi(\theta)) = \nabla(\log \tilde{\pi}(\theta))$$

is the gradient of the logarithm of the (unnormalized) posterior density. The proposal distribution is motivated by a stochastic differential equation, which has π as its stationary distribution.

This proposal is then accepted or rejected using the ordinary Metropolis–Hastings rule, where the proposal density is

$$q(\theta' \mid \theta) = N_p(\theta' \mid \theta + \frac{\sigma^2}{2}\nabla(\log \pi(\theta)),\ \sigma^2 I).$$

7.4.5 Reparametrization

Suppose that the posterior distribution of interest is a continuous distribution and that we have implemented functions for calculating the log-prior and the log-likelihood in terms of the parameter θ. Now we want to consider a diffeomorphic reparametrization

$$\phi = g(\theta) \quad \Leftrightarrow \quad \theta = h(\phi).$$

Typical reparametrizations one might consider are taking the logarithm of a positive parameter or calculating the logit function of a parameter constrained to the interval $(0, 1)$. What needs to be done in order to implement the Metropolis–Hastings algorithm for the new parameter vector ϕ?

First of all, we need a proposal density $q(\phi' \mid \phi)$ and the corresponding code. We also need to work out how to compute one of the Jacobians

$$J_h(\phi) = \frac{\partial\theta}{\partial\phi} \quad \text{or} \quad J_g(\theta) = \frac{\partial\phi}{\partial\theta}.$$

In ϕ-space the target density is given by the change of variables formula

$$f_{\Phi|Y}(\phi \mid y) = f_{\Theta|Y}(\theta \mid y)\left|\frac{\partial\theta}{\partial\phi}\right| = f_{\Theta|Y}(\theta \mid y)\ |J_h(\phi)|\,,$$

where $\theta = h(\phi)$.

The M–H ratio, when we propose ϕ' and the current value is ϕ, is given by

$$\begin{aligned} r &= \frac{f_{\Phi|Y}(\phi' \mid y)\, q(\phi \mid \phi')}{f_{\Phi|Y}(\phi \mid y)\, q(\phi' \mid \phi)} \\ &= \frac{f_{\Theta|Y}(\theta' \mid y)\, |J_h(\phi')|\, q(\phi \mid \phi')}{f_{\Theta|Y}(\theta \mid y)\, |J_h(\phi)|\, q(\phi' \mid \phi)} \\ &= \frac{f_{Y|\Theta}(y \mid \theta')\, f_\Theta(\theta')\, q(\phi \mid \phi')}{f_{Y|\Theta}(y \mid \theta)\, f_\Theta(\theta)\, q(\phi' \mid \phi)}\ \frac{|J_h(\phi')|}{|J_h(\phi)|} \end{aligned}$$

here $\theta' = h(\phi')$ and $\theta = h(\phi)$. Sometimes it is more convenient to work with the Jacobian J_g, but this is easy, since

$$J_g(\theta) = \frac{1}{J_h(\phi)}.$$

Above we viewed the Jacobians as arising from expressing the target density using the new ϕ parametrization instead of the old θ parametrization. An alternative interpretation is that we should express the proposal density in θ space instead of ϕ space and then use the ordinary formula for M–H ratio. Both viewpoints yield the same formulas.

In order to calculate the logarithm of the M–H ratio, we need to do the following.

- Calculate the θ and θ' values corresponding to the current ϕ and proposed ϕ' values.
- Calculate the log-likelihood and log-prior using the values θ and θ'.
- Calculate the logarithm s of the M–H ratio as

$$\begin{aligned} s = {} & \log(f_{Y|\Theta}(y \mid \theta')) - \log(f_{Y|\Theta}(y \mid \theta)) \\ & + \log(f_\Theta(\theta')) - \log(f_\Theta(\theta)) + \log(q(\phi \mid \phi')) - \log(q(\phi' \mid \phi)) \\ & + \log(|J_h(\phi')|) - \log(|J_h(\phi)|). \end{aligned}$$

  Finally, calculate $r = \exp(s)$.

- The difference of the logarithms of the absolute Jacobians can be calculated either on the ϕ scale or on the θ scale by using the identity

$$\log(|J_h(\phi')|) - \log(|J_h(\phi)|) = \log(|J_g(\theta)|) - \log(|J_g(\theta')|).$$

7.4.6 State-dependent mixing of proposal distributions

Let θ be the current state of the chain. Suppose that the proposal θ' is drawn from a proposal density, which is selected randomly from a list of alternatives

$$q(\theta' \mid \theta, j), \qquad j = 1, \ldots K,$$

What is more, the selection probabilities may depend on the current state, as follows.

- Draw j from the pmf $\beta(\cdot \mid \theta), j = 1, \ldots, K$.
- Draw θ' from the density $q(\theta' \mid \theta, j)$ which corresponds to the selected j.
- Accept the proposed value θ' as the new state, if $U < r$, where $U \sim \text{Uni}(0, 1)$, and

$$r = \frac{\pi(\theta')\,\beta(j \mid \theta')\,q(\theta \mid \theta', j)}{\pi(\theta)\,\beta(j \mid \theta)\,q(\theta' \mid \theta, j)}. \tag{7.6}$$

  Otherwise the chain stays at θ.

This formula (7.6) for the M–H ratio r is contained in Green's article [6], which introduced the reversible jump MCMC method. The algorithm could be called the Metropolis–Hastings–Green algorithm.

The lecturer does know any trick for deriving formula (7.6) from the M–H ratio of the ordinary M–H algorithm. The beauty of formula (7.6) lies in the fact that one only needs to evaluate $q(\theta' \mid \theta, j)$ and $q(\theta \mid \theta', j)$ for the proposal density which was selected. A straightforward application of the M–H algorithm would require one to evaluate these densities for all of the K possibilities.

If the selection probabilities $\beta(j \mid \theta)$ do not actually depend on θ, then they cancel from the M–H ratio. In this case (7.6) is easily derived from the ordinary M–H algorithm.

7.5 Gibbs sampler

One of the best known ways of setting up an MCMC algorithm is Gibbs sampling, which is now discussed supposing that the target distribution is a posterior distribution. However, the method can be applied to any target distribution, when the full conditional distributions of the target distribution are available.

Suppose that the parameter vector has been divided into components

$$\theta = (\theta_1, \theta_2, \ldots, \theta_d),$$

where θ_j need not be a scalar. Suppose also that the posterior full conditional distributions of each of the components are available in the sense that we know how to simulate them. This is the case when the statistical model exhibits conditional conjugacy with respect to all of the components θ_j. Then the basic idea behind Gibbs sampling is that we simulate successively each component θ_j from its (posterior) full conditional distribution. It is convenient to use the abbreviation θ_{-j} for the vector, which contains all the other components of θ but θ_j, i.e.

$$\theta_{-j} = (\theta_1, \ldots, \theta_{j-1}, \theta_{j+1}, \ldots, \theta_d). \tag{7.7}$$

Then the posterior full conditional of θ_j is

$$p(\theta_j \mid \theta_{-j}, y) = f_{\Theta_j \mid \Theta_{-j}, Y}(\theta_j \mid \theta_{-j}, y). \tag{7.8}$$

A convenient shorthand notation for the posterior full conditional is

$$p(\theta_j \mid \cdot),$$

where the dot denotes all the other random variables except θ_j.

The most common form of the Gibbs sampler is the systematic scan Gibbs sampler, where the components are updated in a fixed cyclic order. It is also possible to select at random which component to update next. In that case one has the random scan Gibbs sampler.

Algorithm 16 presents the systematic scan Gibbs sampler, when we update the components using the order $1, 2, \ldots, d$. In the algorithm i is the time index of the Markov chain. One needs d updates to get from $\theta^{(i)}$ to $\theta^{(i+1)}$. To generate the j'th component, $\theta_j^{(i+1)}$, one uses the most recent values for the other components, some of which have already been updated. I.e., when the value for

$\theta_j^{(i+1)}$ is generated, it is generated from the corresponding full conditional using the following values for the other components,

$$\theta_{-j}^{\text{cur}} = (\theta_1^{(i+1)}, \ldots, \theta_{j-1}^{(i+1)}, \theta_{j+1}^{(i)}, \ldots, \theta_d^{(i)}).$$

Algorithm 16: Systematic scan Gibbs sampler.

Input: An initial value $\theta^{(0)}$ such that $f_{\Theta|Y}(\theta^{(0)} \mid y) > 0$ and the number of iterations N.

Result: Values simulated from a Markov chain which has the posterior distribution as its invariant distribution.

```
θ^cur ← θ^(0)
for i = 0, 1, ..., N do
    for j = 1, ..., d do
        draw a new value for the jth component θ_j^cur of θ^cur from the
        posterior full conditional f_{Θ_j|Θ_{-j},Y}(θ_j | θ_{-j}^cur, y)
    end
    θ^(i+1) ← θ^cur
end
```

Usually the updating steps for the components of θ are so heterogeneous, that the inner loop is written out in full. E.g., in the case of three components, $\theta = (\phi, \psi, \tau)$, the actual implementation would probably look like the following algorithm 17. This algorithm also demonstrates, how one can write the algorithm using the abbreviated notation for conditional densities.

Algorithm 17: Systematic scan Gibbs sampler for three components $\theta = (\phi, \psi, \tau)$ given initial values for all the components except the one that gets updated the first.

```
ψ^cur ← ψ_0;   τ^cur ← τ_0;
for i = 0, 1, ..., N do
    draw φ^cur from p(φ | ψ = ψ^cur, τ = τ^cur, y);
    draw ψ^cur from p(ψ | φ = φ^cur, τ = τ^cur, y);
    draw τ^cur from p(τ | φ = φ^cur, ψ = ψ^cur, y);
    φ_{i+1} ← φ^cur;   ψ_{i+1} ← ψ^cur;   τ_{i+1} ← τ^cur;
end
```

Algorithm 18 presents the random scan Gibbs sampler. Now one time step of the Markov chain requires only one update of a randomly selected component. In the random scan version, one can have different probabilities for updating the different components of θ, and this freedom can be useful for some statistical models.

If the statistical model exhibits conditional conjugacy with respect to all the components of θ, then the Gibbs sampler is easy to implement and is the method of choice for many statisticians. One only needs random number generators for all the posterior full conditionals, and these are easily available for the standard distributions. An appealing feature of the method is the fact that one does not need to choose the proposal distribution as in the Metropolis–Hastings sampler;

Algorithm 18: Random scan Gibbs sampler.

Input: An initial value $\theta^{(0)}$ such that $f_{\Theta|Y}(\theta^{(0)} \mid y) > 0$, the number of iterations N and a probability vector $\beta_1, \ldots, \beta_d$: each $\beta_j > 0$ and $\beta_1 + \cdots + \beta_d = 1$.

Result: Values simulated from a Markov chain which has the posterior distribution as its invariant distribution.

$\theta^{\text{cur}} \leftarrow \theta^{(0)}$;
for $i = 0, 1, \ldots, N$ **do**
    select j from $\{1, \ldots, d\}$ with probabilities $(\beta_1, \ldots, \beta_d)$;
    draw a new value for the component θ_j^{cur} from the posterior full conditional $f_{\Theta_j|\Theta_{-j},Y}(\theta_j \mid \theta_{-j}^{\text{cur}}, y)$;
    $\theta^{(i+1)} \leftarrow \theta^{\text{cur}}$;
end

the proposals of the Gibbs sampler are somehow automatically tuned to the target posterior. However, if some of the components of θ are strongly correlated in the posterior, then the convergence of the Gibbs sampler suffers. So one might want to reparametrize the model so that the transformed parameters are independent in their posterior. Unfortunately, most reparametrizations destroy the conditional conjugacy properties on which the attractiveness of the Gibbs sampler depends.

The name Gibbs sampling is actually not quite appropriate. Gibbs studied distributions arising in statistical physics (often called Gibbs distributions or Boltzmann distributions), which have densities of the form

$$f(x_1, \ldots, x_d) \propto \exp\left(-\frac{1}{kT} E(x_1, \ldots, x_d)\right),$$

where $(x_1, \ldots, x_d)$ is the state of physical system, k is a constant, T is the temperature of the system, and $E(x_1, \ldots, x_d) > 0$ is the energy of the system. The Geman brothers used a computational method (simulated annealing), where a computational parameter corresponding to the the temperature of a Gibbs distribution was gradually lowered towards zero. At each temperature the distribution of the system was simulated using the Gibbs sampler. This way they could obtain the configurations of minimal energy in the limit. The name Gibbs sampling was selected in order to emphasize the relationship with the Gibbs distributions. However, when the Gibbs sampler is applied to posterior inference, the temperature parameter is not needed, and therefore the reason for the name Gibbs has disappeared. Many authors have pointed this out this deficiency and proposed alternative names for the sampling method, but none of them have stuck.

7.6 Componentwise updates in the Metropolis–Hastings algorithm

Already Metropolis *et al.* and Hastings pointed out that one can use componentwise updates in the Metropolis–Hastings algorithm. This is sometimes called single-site updating.

When the parameter vector is divided into d components

$$\theta = (\theta_1, \theta_2, \ldots, \theta_d),$$

one needs d proposal densities

$$\theta'_j \mapsto q_j(\theta'_j \mid \theta^{\text{cur}}), \qquad j = 1, \ldots, d,$$

which may all be different.

When it is time to update the jth component, we do a single Metropolis–Hastings step. When the current value of the parameter vector is θ^{cur}, we propose the vector θ', where the jth component is drawn from the proposal density $q_j(\theta_j \mid \theta^{\text{cur}})$, and the rest of the components of θ' are equal to those of the current value θ^{cur}. Then the proposal is accepted or rejected using the M–H ratio

$$r = \frac{p(\theta' \mid y)\, q_j(\theta_j^{\text{cur}} \mid \theta')}{p(\theta^{\text{cur}} \mid y)\, q_j(\theta'_j \mid \theta^{\text{cur}})} \tag{7.9}$$

The vectors θ' and θ^{cur} differ only in the jth place, and therefore one can write the M–H ratio (for updating the jth component) also in the form

$$r = \frac{p(\theta'_j \mid \theta_{-j}^{\text{cur}}, y)\, q_j(\theta_j^{\text{cur}} \mid \theta')}{p(\theta_j^{\text{cur}} \mid \theta_{-j}^{\text{cur}}, y)\, q_j(\theta'_j \mid \theta^{\text{cur}})}, \tag{7.10}$$

where we used the multiplication rule to express the joint posterior as

$$p(\theta \mid y) = p(\theta_{-j} \mid y)\, p(\theta_j \mid \theta_{-j}, y)$$

both in the numerator and in the denominator, and then cancelled the common factor $p(\theta_{-j}^{\text{cur}} \mid y)$. Although eqs. (7.9) and (7.10) are equivalent, notice that in eq. (7.9) we have the M–H ratio when we regard the joint posterior as the target distribution, but in eq. (7.10) we have ostensibly the M–H ratio, when the target is the posterior full conditional of component j. If one then selects as q_j the posterior full conditional of the component θ_j for each j, then each proposal is accepted and the Gibbs sampler ensues.

One can use this procedure either a systematic or a random scan sampler, as is the case with the Gibbs sampler. The resulting algorithm is often called the Metropolis–within–Gibbs sampler. (The name is illogical: the Gibbs sampler is a special case of the Metropolis–Hastings algorithm with componentwise updates.) This is also a very popular MCMC algorithm, since then one does not have to design a single complicated multivariate proposal density but p simpler proposal densities, many of which may be full conditional densities of the posterior.

Small modifications in the implementation can sometimes make a big difference to the efficiency of the sampler. One important decision is how to divide the parameter vector into components. This is called **blocking** or **grouping**. As a general rule, the less dependent the components are in the posterior, the better the sampler. Therefore it may be a good idea to combine highly correlated components into a single block, with is then updated as a single entity.

It is sometimes useful to update the whole vector jointly using a single Metropolis–Hastings acceptance test, even if the proposed value is build up component by component taking advantage of conditional conjugacy properties. These and other ways of improving the performance of MCMC algorithms in the context of specific statistical models are topics of current research.

7.7 Analyzing MCMC output

After the MCMC algorithm has been programmed and tested, the user should investigate the properties of the algorithm for the particular problem he or she is trying to solve. There are available several tools, e.g., for

- diagnosing convergence
- estimating Monte Carlo standard errors.

We discuss some of the simpler tools.

A **trace plot** of a parameter ϕ is a plot of the iterates $\phi^{(t)}$ against the iteration number t. These are often examined for each of the components of the parameter vector, and sometimes also for selected scalar functions of the parameter vector. A trace plot is also called a *sample path*, a *history plot* or a *times series plot*. If the chain mixes well, then the trace plots move quickly away from their starting values and they wiggle vigorously in the region supported by the posterior. In that case one may select the length of the burn-in by examining trace plots. (This is not foolproof, since the chain may only have converged momentarily to some neighborhood of a local maximum of the posterior.) If the chain mixes poorly, then the traces will remain nearly constant for many iterations and the state may seem to wander systematically towards some direction. Then one may need a huge number of iterations before the traces show convergence.

An **autocorrelation plot** is a plot of the autocorrelation of the sequence $\phi^{(t)}$ at different iteration lags. These can be produced for all the interesting components of θ, but one should reject the burn-in before estimating the autocorrelation so that one analyzes only that part of the history where the chain is approximately stationary. The autocorrelation function (acf) of a stationary sequence of RVs (X_i) at lag k is defined by

$$R(k) = \frac{E[(X_i - \mu)(X_{i+k} - \mu)]}{\sigma^2}, k = 0, 1, 2, \ldots,$$

where $\mu = EX_i$, $\sigma^2 = \operatorname{var} X_i$, and the assumption of stationarity entails that μ, σ^2 and $R(k)$ do not depend on index i. For an i.i.d. sequence the autocorrelation function is one at lag zero and zero otherwise. A chain that mixes slowly exhibits slow decay of the autocorrelation as the lag increases. When there are more than one parameter, one may also examine cross-correlations between the parameters.

There exist tools for **convergence diagnostics**, which try to help in deciding whether the chain has already approximately reached its stationary distribution and in selecting the length of the burn-in period. E.g., in the approach of Gelman and Rubin, the chain is run many times starting from separate starting values dispersed over the support of the posterior. After the burn-in has been discarded, one calculates statistics which try to check whether all the chains have converged to the same distribution. In some other approches one needs to simulate only a single chain and one compares the behaviour of the chain in the beginning and in the end of the simulation run. Such convergence diagnostic are available in the `coda` R package and in the `boa` R package. However, convergence diagnostic tools can not prove that the chain has converged. They only help you to detect obvious cases of non-convergence.

If the chain seems to have converged, then it is of interest to estimate standard errors for the scalar parameters. The naive estimate (which is correct for i.i.d. sampling) would be to calculate the sample standard deviation of the last L iterations divided by $\sqrt{L}$ (after the burn-in has been discarded). However, MCMC iterates are typically positively correlated, and therefore this would underestimate severely the standard error.

A simple method for estimating the standard errors for posterior expectations

$$E[h(\Theta) \mid Y = y]$$

is the method of **batch means** [8], where the L last iterates are divided into a non-overlapping batches of length b. Then one computes the mean $\bar{h}_j$ of the values $h(\theta^{(t)})$ inside each of the batches $j = 1, \ldots, a$ and estimates the standard error of the grand mean $\bar{h}$ as the square roof of

$$\frac{1}{a} \frac{1}{a-1} \sum_{j=1}^{a} (\bar{h}_j - \bar{h})^2,$$

where $\bar{h}$ is the grand mean calculated from all the the L last iterates $h(\theta^{(t)})$. The idea here is to treat the batch means as i.i.d. random variables whose expected value is the posterior expectation. One should perhaps select the batch length as a function of the simulation length, e.g., with the rule $b = \lfloor \sqrt{L} \rfloor$.

7.8 Example

Consider the two dimensional normal distribution $N(0, \Sigma)$ as the target distribution, where

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}, \qquad -1 < \rho < 1, \quad \sigma_1, \sigma_2 > 0,$$

and ρ is nearly one. Of course, it is possible to sample this two-variate normal distribution directly. However, we next apply MCMC algorithms to this highly correlated toy problem in order to demonstrate properties of the Gibbs sampler and a certain Metropolis–Hastings sampler.

The full conditionals of the target distribution are given by

$$\begin{aligned} [\Theta_1 \mid \Theta_2 = \theta_2] &\sim N\left(\tfrac{\rho\sigma_1}{\sigma_2}\theta_2, (1-\rho^2)\sigma_1^2\right) \\ [\Theta_2 \mid \Theta_1 = \theta_1] &\sim N\left(\tfrac{\rho\sigma_2}{\sigma_1}\theta_1, (1-\rho^2)\sigma_2^2\right), \end{aligned}$$

and these are easy to simulate. We now suppose that

$$\rho = 0.99, \quad \sigma_1 = \sigma_2 = 1.$$

Figure 7.2 shows the ten first steps of the Gibbs sampler, when all the component updates ("half-steps" of the sampler) are shown. Since ρ is almost one, the Gibbs sampler is forced to take small steps, and it takes a long time for it to explore the main support of the target distribution.

Figure 7.2: The first ten iterations of the Gibbs sampler. The three contour lines enclose 50 %, 90 % and 99 % of the probability mass of the target distribution.

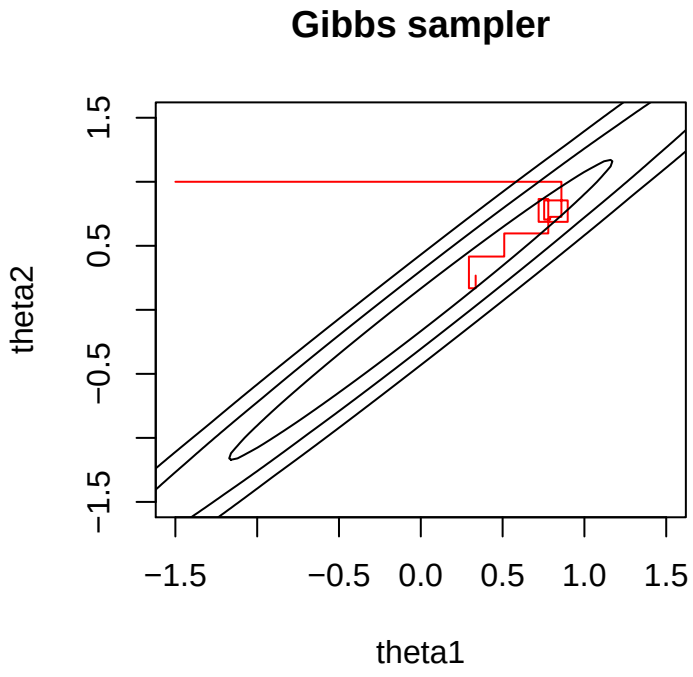
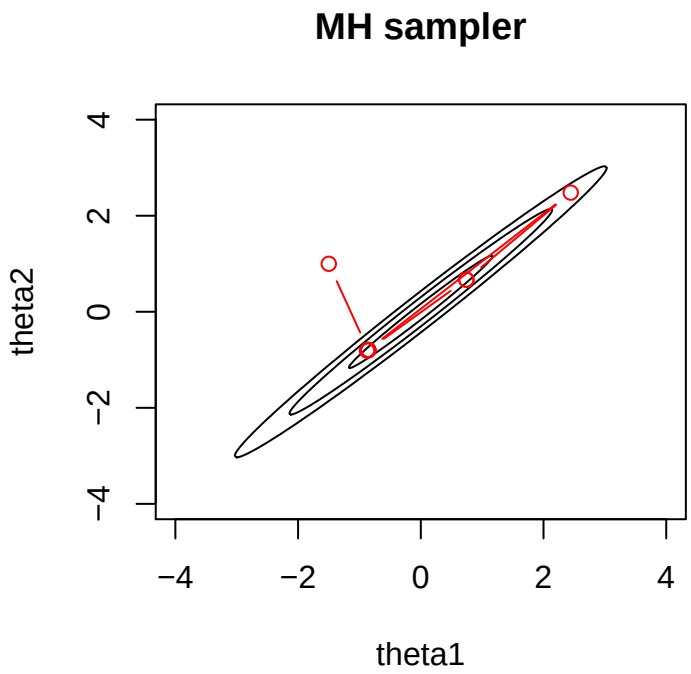


Figure 7.3: The first ten iterations of the Metropolis–Hastings sampler. Notice the sampler produced less than ten distinct θ values. The three contour lines enclose 50 %, 90 % and 99 % of the probability mass of the target distribution.



Another strategy would be to generate the proposal in two stages as follows. We first draw θ_1' from some convenient proposal distribution, e.g., by the random walk proposal

$$\theta_1' = \theta_1^{\text{cur}} + w,$$

where w is generated from (say) $N(0, 4)$. Then we draw θ_2' from the full conditional distribution of θ_2 conditioning on the proposed value θ_1'. Then the overall proposal density is given by

$$q((\theta_1', \theta_2') \mid (\theta_1^{\text{cur}}, \theta_2^{\text{cur}})) = N(\theta_1' - \theta_1^{\text{cur}} \mid 0, 4)\, N(\theta_2' \mid \frac{\rho\sigma_2}{\sigma_1}\theta_1', (1-\rho^2)\sigma_2^2)$$

We then either accept or reject the transition from θ^{cur} to θ' using the ordinary acceptance rule of the Metropolis–Hastings sampler. This algorithm explores the target distribution much more efficiently, as can be guessed from Figure 7.3, which shows the first ten iterations of the sampler. The random walk proposal gives the component θ_1 freedom to explore the parameter space, and then the proposal from the full conditional for θ_2 draws the proposed pair into the main support of the target density.

Figure 7.4 shows the traces of the components using the two algorithms. The Metropolis–Hastings sampler seems to mix better than the Gibbs sampler, since there seems to be less dependence between the consecutive simulated values. Figure 7.5 shows the autocorrelation plots for the two components using the two different samplers. The autocorrelation functions produced by the Gibbs sampler decay more slowly than those produced by the Metropolis–Hastings sampler, and this demonstrates that we obtain better mixing with the Metropolis–Hastings sampler.

7.9 Literature

The original references on the Metropolis sampler, the Metropolis–Hastings sampler and the Gibbs sampler are [9, 7, 4]. The article by Gelfand and Smith [3] finally convinced the statistical community about the usefulness of these methods in Bayesian inference. The book [5] contains lots of information on MCMC methods and their applications.

The books by Nummelin [11] or Meyn and Tweedie [10] can be consulted for the theory of Markov chains in a general state space. The main features of the general state space theory are explained in several sources, including [2, Ch. 14] or [12, Ch. 6].

Bibliography

[1] Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and Computing*, 18:343–373, 2008.

[2] Krishna B. Athreya and Soumendra N. Lahiri. *Measure Theory and Probability Theory*. Springer Texts in Statistics. Springer, 2006.

[3] A. E. Gelfand and A. F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85:398–409, 1990.

Figure 7.4: Sampler traces for the two components θ_1 and θ_2 using the Gibbs sampler and the Metropolis–Hastings sampler.

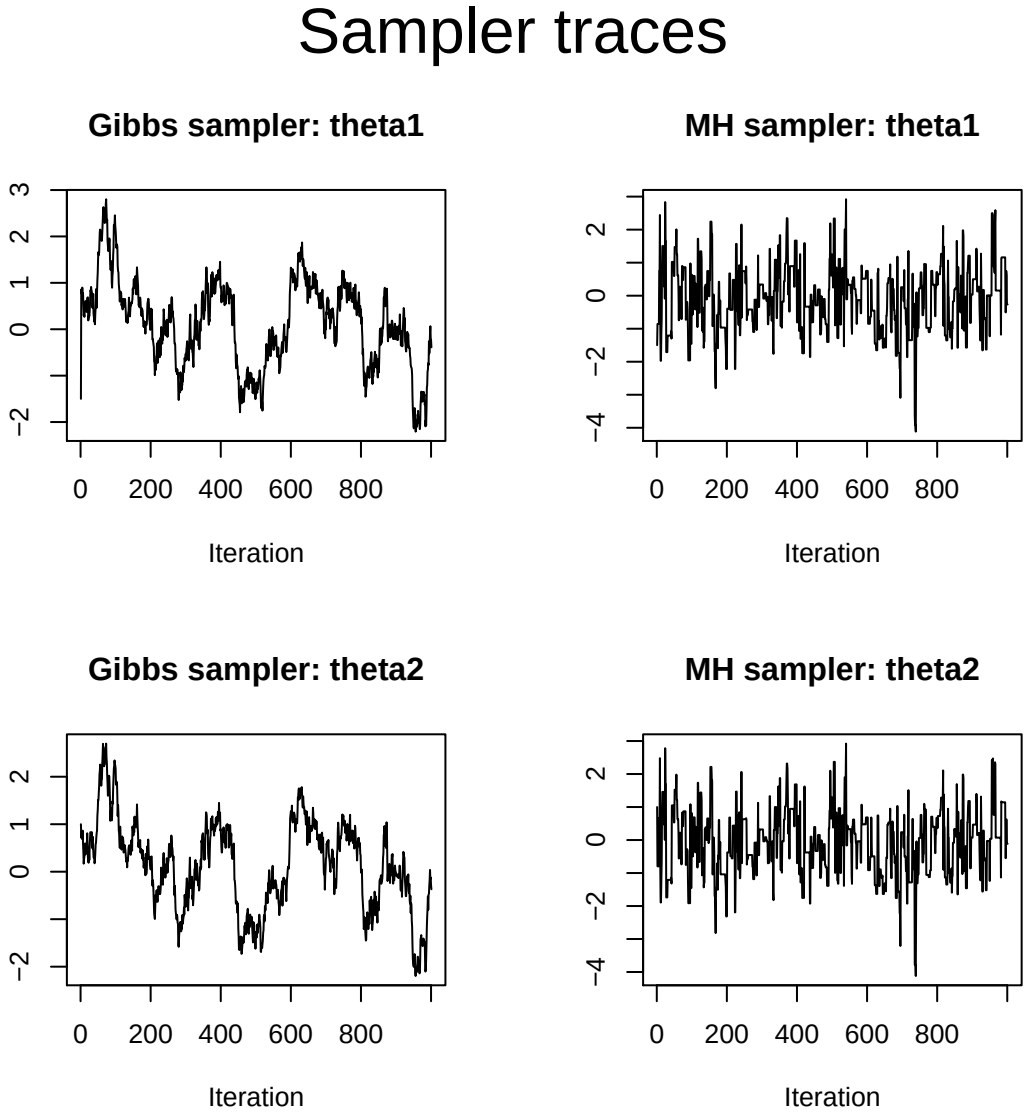
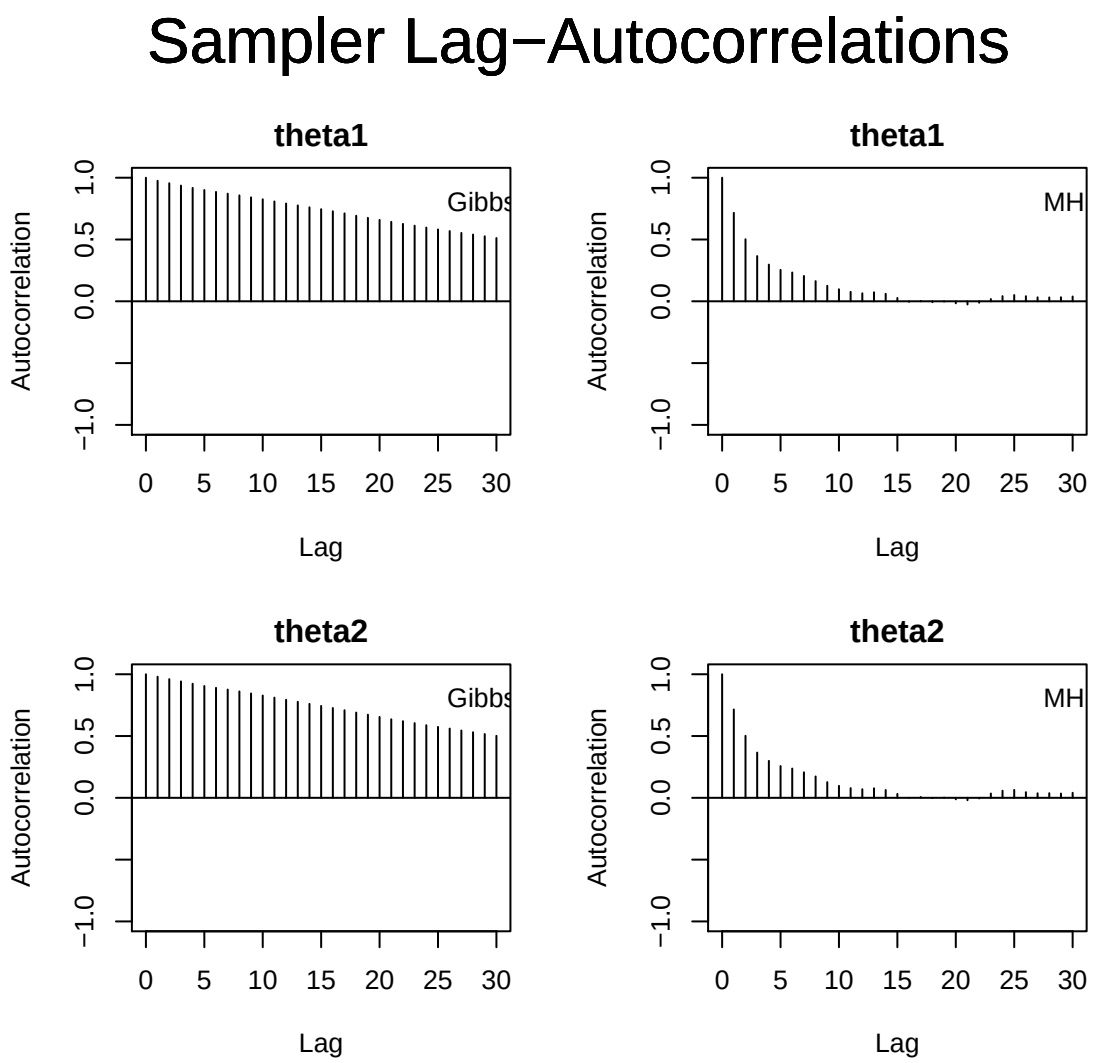


Figure 7.5: Sampler autocorrelation functions for the two components θ_1 and θ_2 using the Gibbs sampler and the Metropolis–Hastings sampler.



[4] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[5] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman & Hall, 1996.

[6] Peter J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82:711–732, 1995.

[7] W. Hastings. Monte Carlo sampling methods using Markov chains and their application. *Biometrika*, 57:97–109, 1970.

[8] Averll M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, Inc., 2nd edition, 1991.

[9] N. Metropolis, A. Rosenbluth, , M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[10] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Springer, 1993.

[11] Esa Nummelin. *General Irreducible Markov Chains and Nonnegative Operators*. Cambridge University Press, 1984.

[12] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, second edition, 2004.

[13] Gareth O. Roberts and Jeffrey S. Rosenthal. Optimal scaling for various Metropolis–Hastings algorithms. *Statistical Science*, 16(4):351–367, 2001.