M.Sc. in Applied Statistics MT2010

MCMC and Applied Bayesian Statistics

© 2008–10 B. D. Ripley[1]

1 Simulation-based Inference

One very important idea for doing statistical inference on analytically intractable statistical models (that is, most real-world ones) is to make use of *simulation*. As in Geyer (1999)'s comments about MCMC for spatial point processes:

> *If you can write down a model, I can do likelihood inference for it, not only maximum likelihood estimation, but also likelihood ratio tests, likelihood-based confidence intervals, profile likelihoods, whatever. That includes conditional likelihood inference and inference with missing data.*
>
> *This is overstated, of course. ... But analyses that can be done are far beyond what is generally recognized.*

The basic idea is quite simple – simulate data from one or more plausible models (or for a parametric model, at a range of plausible parameter values), apply the same (or similar) procedure to the simulated datasets as was applied to the original data, and then analyse the results. In this section we consider some of the 'classical' applications, but bootstrapping (in the *Computer-Intensive Statstiics* module) is another.

The literature often confuses MCMC methods with Bayesian inference, which is one reasom the opening quote is about likelihoods — and MCMC arose in classical not Bayesian statistics and it is equally applicable to both.

Parts of this module are quite technical—whereas you do not need to know how, for example, the MLE is found for a logistic regression, MCMC is a relatively young subject and you do need to know a fair amount about how things are being simulated to make good use of it.

Simulations are experiments, albeit inside a computer, and need to be designed and their output analysed just like any other experiment. The main difference here is that the output from an MCMC experiment is one or more time series.

[1]Thanks to Martyn Plummer and Ruth Ripley for their comments. Also based on material lectured to APTS.

A motivating example – Strauss processes

Let us start with an example from my PhD thesis based on a 1975 paper by David Strauss.

Suppose you have observed a spatial pattern of trees, say n trees within a sampling area A, and the actual (two-dimensional) locations are $\boldsymbol{x}_i, i = 1, \ldots, n$. The aim was to produce a model for the pattern of the trees. What makes this interesting is

(a) there is no definitive Euclidean coordinate system – the trees do not know where North is nor the Greenwich Meridian nor the Equator.

(b) the trees do not have an order, so we have a *set* not a *sequence*.

(c) the trees are not positioned independently – trees compete for light and nutrients.

(d) trees do not compete just within the study area A but also with trees outside A.

Point (d) makes modelling more difficult – we need to model a process of trees within the whole forest observed only in A – but for simplicity we will ignore that here.

The model Frank Kelly and I came up with implied a joint density for the sample $\boldsymbol{x} = \{\boldsymbol{x}_i, i = 1, \ldots, n\}$ of

$$f(\boldsymbol{x}; c, R) \propto c^{t(\boldsymbol{x},R)} \qquad c \geq 0 \tag{1}$$

where $t(\boldsymbol{x}, R)$ is the number of 'R-close pairs', the number of (unordered) pairs of trees less than R units apart. (Note that this meets modelling aims (a–c).)

At that point we had several questions

1. What do spatial patterns from process (1) typically look like?
2. What values of c are valid? Our theory suggested that $c > 1$ violated point (d), but (1) is a valid pdf – is it modelling anything interesting?

   Note that $c = 0$ *is* interesting – it is a *hard-core* interaction process.

   The original paper by Strauss was entitled *A Model for Clustering*, and clustering would need $c > 1$.
3. How could we do statistical inference on the parameter c, and perhaps R?

At around that time Prof. D. G. Kendall told me

> *you don't really understand a stochastic process until you know how to simulate it*

and perhaps I should add

> *you don't really know how to simulate a stochastic process until you have done so and validated your simulations*

So these points are inter-related – one way to validate the simulation is to be able to do inference on its output.

I did have an idea about how to do the simulation — see Chapter 2.

Inference from (1) superficially looks easy: if we write it (*via* $c = e^{-\theta}$) as

$$f(\boldsymbol{x}; \theta, R) \propto \exp -\theta t(\boldsymbol{x}, R) \qquad -\infty < \theta \leq \infty \tag{2}$$

it is shown as a canonical exponential family for θ, albeit with an unusual parameter range. The problem is that this is really

$$f(\boldsymbol{x}; \theta, R) = C(\theta, R) \exp -\theta t(\boldsymbol{x}, R) \qquad -\infty < \theta \leq \infty$$

and to find $C(\theta, R)$ we need to do a $2n$-dimensional integral that we can only do numerically – and no reasonable approximations were known to us then (nor now).

However, this tells us that the MLE for θ solves

$$E_\theta\, t(\boldsymbol{X}, R) = t(\boldsymbol{x}, R) \tag{3}$$

where $\boldsymbol{X}$ is a sample from (2) with parameter θ. If we can simulate from (2) we can estimate the LHS as a function of θ and hence get the MLE. An example is shown in Figure 1: for more details see Ripley (1988, §4.6). Because we have used simulation we can not only produce a point estimate of the MLE: we can also obtain a non-asymptotic idea of its variability.

So once we can produce simulations we can make a lot of progress on our statistical questions – at least if we can do so within the computing resources available.

Note that by importance sampling we can estimate the RHS of (3) for a range of θ from simulations at one value, an idea sometimes known as *polysampling*. The idea is that

$$E_\theta[t(\boldsymbol{X}, R)] = E_{\theta_0}\left[t(\boldsymbol{X}, R)\frac{f(\boldsymbol{X}; \theta, R)}{f(\boldsymbol{X}; \theta_0, R)}\right]$$

so we can take a series of samples at $\theta = \theta_0$, replace the expectation on the RHS by a weighted average over those samples, and thereby estimate the LHS for any θ. The downside is that the estimator is likely to be a good estimator only for θ near θ_0. What does 'near' mean? Well, this is an experiment and standard statistical methods (e.g. *response surface designs*) can be employed to answer such questions.

Finding marginals

A great deal of statistics is about finding marginal distributions of quantities of interest. This occurs in both frequentist and Bayesian settings—especially the latter, where almost all questions boil down to finding a marginal distribution.

Finding those marginals is often difficult, and textbook examples are chosen so that the integrations needed can be done analytically. A great deal of ingenuity has been used in finding systematic ways to compute marginals: examples include the Lauritzen & Spiegelhalter (1988) message-passing algorithm for graphical models.

It is an almost trivial remark that simulation provides a very simple way to compute marginals. Suppose we have a model that provides a joint distribution for a (finite) collection (X_i) of random variables. Then if we have a way to simulate from the joint distribution, taking a subset of the variables provides a painless way to get a marginal distribution of that subset.

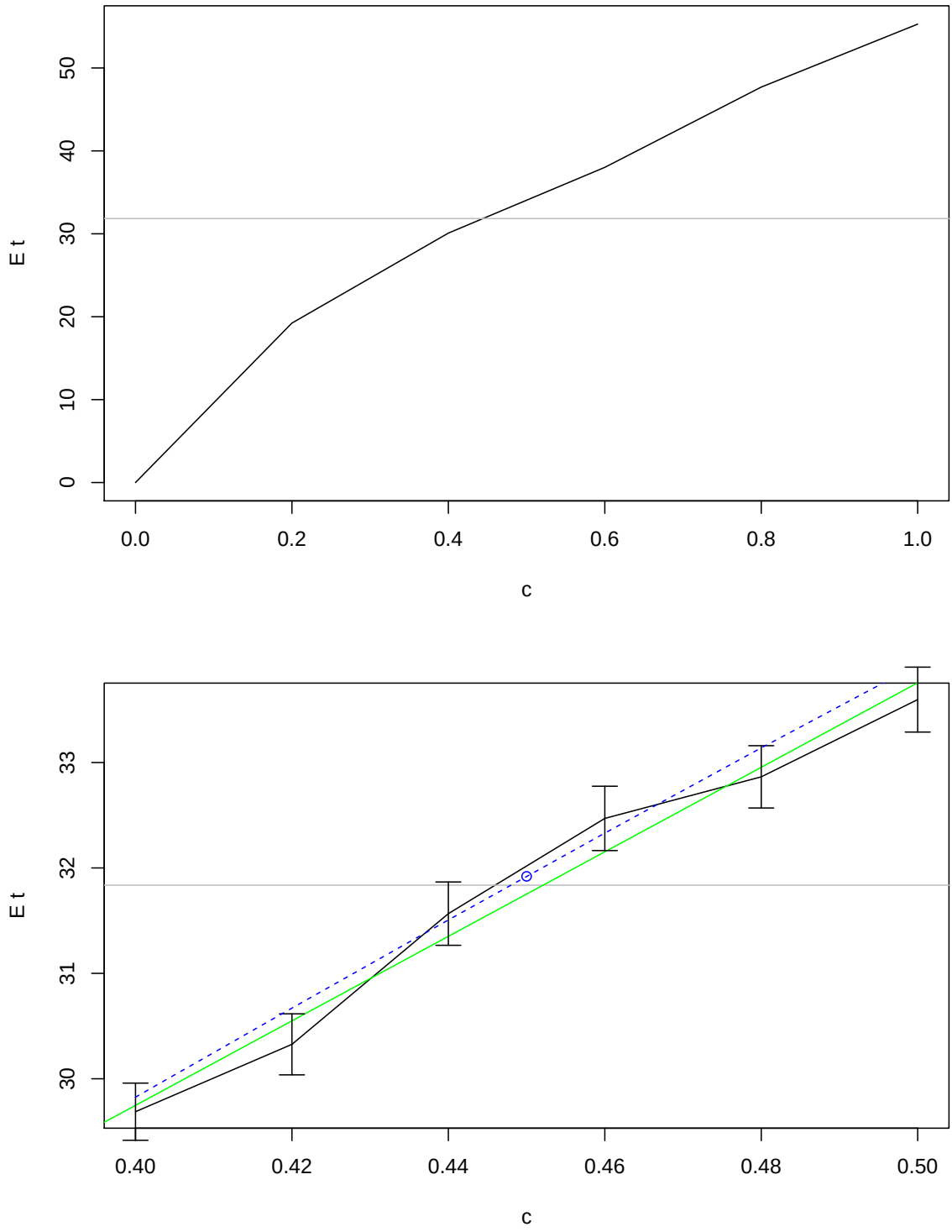


Figure 1: Fitting the Strauss model (1). The top figure shows the average of 100 simulations at five values of c. The lower figure shows the means and 95% confidence intervals based on 1000 simulations at six values of c, plus (green) the fitted regression line and (blue, dashed) values estimated by polysampling from 1000 simulations at $c = 0.45$. The grey line is the observed value – the non-integer value is because some edge corrections are in use.

You should be used to thinking of distributions as represented by samples and so know many ways to make use of that sample as a surrogate for the distribution.

Note that this does not apply directly to marginals in conditional distributions, as we would need to be able to simulate from the conditional distribution. This is not a problem in the standard Bayesian context where we simulate from the posterior distribution, that is the distribution conditional on the observed data. It is an issue when exploring model fits, where we often want to explore how much one (or more) observation is influencing the results, or even to correct data after discovering large influence.

In the examples we will be using anywhere from a handful to 10,000 samples to represent a marginal distribution. It is important to remember that we only have an approximation to the distribution. A few thousand samples seems like a lot when we are looking at univariate marginals (as people almost invariably do), but we are most often looking at univariate marginals because this is easy to do, not because they are the sole or main interest. For figure 1 we are comparing simulations of 69 points in a square with some data – this is a 138-dimensional problem and we have sophisticated multi-dimensional ways to compare such patterns.

SIR

The so-called *sampling-importance-resampling* is a technique for improving on an approximate distribution. Suppose we have M samples x_i simulated from an approximation q to a target distribution p. Then *importance sampling* is the idea of estimating

$$E\,h(X) = \int h(x)\,p(x)\,dx = \int h(x)\frac{p(x)}{q(x)}\,q(x)\,dx$$

by the weighted average of $h(x_i)$ with weights $w_i = p(x_i)/q(x_i)$. So we can represent distribution p by a weighted sample from distribution q. For many purposes it is more convenient to have an unweighted sample, and SIR achieves this approximately by taking a subsample of size $m < M$ by weighted sampling without replacement from the current sample. That is we repeat m times

> Select one of the remaining x_i with probability proportional to w_i and remove it from the (x_i).

(See Gelman *et al.*, 2004, pp. 316f, 450.) (Others, including Rubin's original version[2] in the discussion of Tanner & Wong (1987) describe SIR as the version with replacement: the difference will be small if $m \ll M$.) Despite the name, this is a form of rejection sampling.

We have already seen importance sampling used to explore nearby parameter values, and resampling can be used in the same way. Both can be used to perturb Bayesian analyses, e.g. to vary the prior (perhaps away from one chosen for tractability towards something more realistic), as changing the prior just re-weights the posterior samples. Another perturbation sometimes of interest is to consider their influence by dropping observations one at a time: and for independent observations this rescales the posterior density by the contribution of the

[2] by this name: the idea is older.

observation to the likelihood. If we have n independent observations,

$$p(\theta \mid \boldsymbol{y}) \propto \prod_{i=1}^{n} \ell(y_i; \theta) p(\theta)$$

and hence the posterior discarding observation j is proportional to

$$p(\theta \mid \boldsymbol{y}) / \ell(y_j; \theta)$$

For a pre-MCMC perspective on the potential role of SIR in Bayesian statistics, see Smith & Gelfand (1992).

Stochastic Approximation

An alternative is to solve equation (3) by iterative methods, usually called *Robbins–Monro* methods or *stochastic approximation*. Suppose we seek to solve

$$\Phi(\theta) = E\phi(\theta, \epsilon) = 0$$

for increasing Φ, and that we can draw independent samples from $\phi(\theta, \epsilon)$. A sequence of estimates is defined recursively by

$$\theta_{n+1} = \theta_n - a_n \phi(\theta_n, \epsilon_n)$$

for $a_n \to 0$, e.g. $a_n \propto n^{-\gamma}$ for $0 < \gamma \leq 1$. Kushner & Lin (2003) and Ripley (1987, p. 185) gives further details and more sophisticated variants, which include averaging over recent values of θ_n.

The `SIENA` program[3] for fitting models of social networks is almost entirely based on these ideas. These are networks with a finite set of nodes (actors) but with links that evolve through time (e.g. who is 'best friends' with whom in a school). Snijders (2006) writes

> *These models can be simulated on computers in rather straightforward ways (cf. Snijders, 2005). Parameter estimation, however, is more complicated, because the likelihood function or explicit probabilities can be computed only for uninteresting models. This section presents the Methods of Moments estimates proposed in Snijders (2001). [. . .]*

This is just a Big Name for the idea we have illustrated for the Strauss model, equating empirical and simulated moments, mainly by using stochastic approximation.

Simulated annealing

Simulated annealing is an idea for optimizing functions of many variables, most often discrete variables so a *combinatorial optimization* problem. The name comes from Kirkpatrick *et al.* (1983) and from *annealing*, a process in a metallurgy in which molten metal is cooled extremely slowly to produce a (nearly) stress-free solid. Since annealing is a process to produce

[3] 'Simulation Investigation for Empirical Network Analysis': `http://www.stats.ox.ac.uk/siena`.

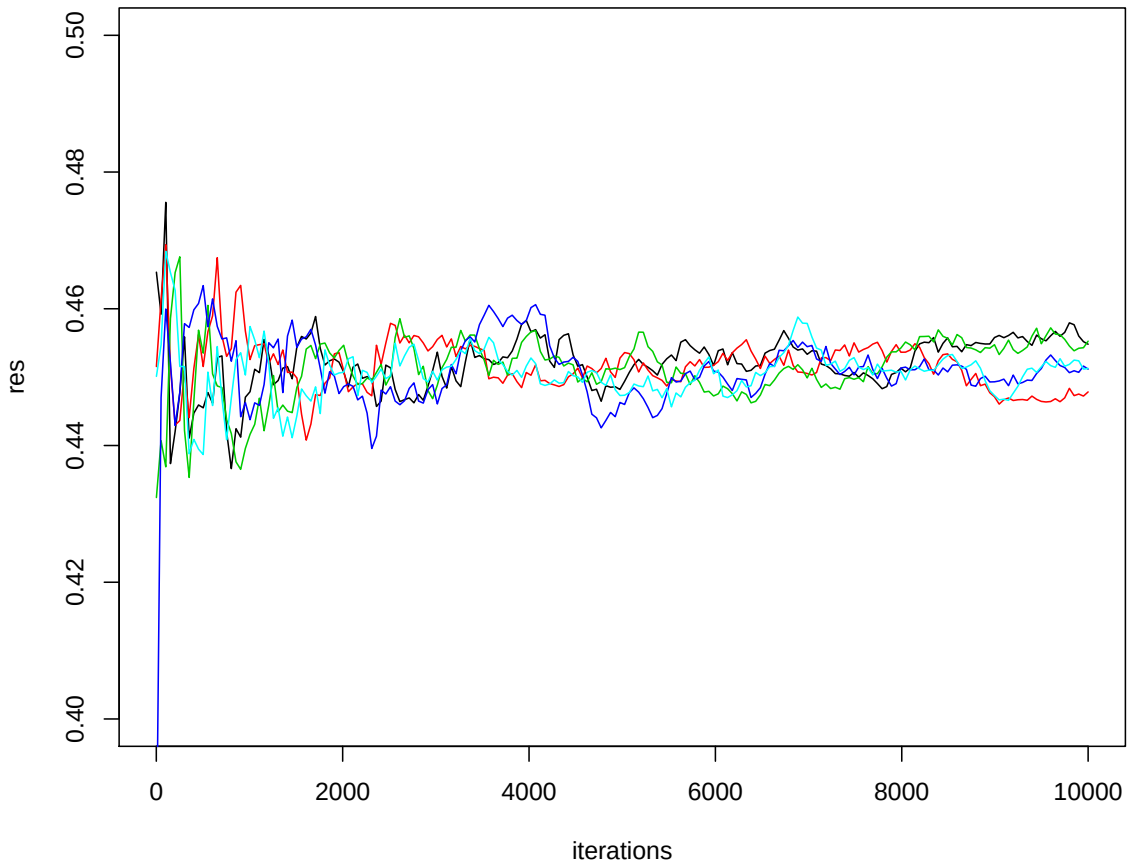


Figure 2: Five runs of fitting the Strauss model (1) by stochastic approximation from $U(0.4, 0.5)$ starting values with $\gamma = 0.5$.

a low-energy configuration of the atoms, it is natural[4] to consider its application to optimization of complex problems.

The ground was set by Pincus (1970), based on the idea that if f is continuous over a compact set D and has a unique global maximum at x^* then

$$x^* = \lim_{\lambda \to \infty} \frac{\int_D x \exp\ \lambda f(x)\, dx}{\int_D \exp\ \lambda f(x)\, dx}$$

So if we take a series of samples from density proportional to

$$\exp\ \lambda f(x)$$

for increasing λ, then the distribution of the samples will become increasingly concentrated about x^*. And this procedure is particularly suited to the iterative simulation methods of MCMC since we can use the sample(s) at the previous value of λ to start the iterative process. However, the rate at which λ needs to be increased is very slow, with some studies suggesting that $\lambda \propto \log(1 + t)$ with t the number of iterative steps completed.

Despite the unpromising theoretical behaviour, simulated annealing has proved useful in finding improved solutions to both continuous and combinatorial optimization problems – see e.g. Aarts & Korst (1989)

[4]at least to those with some knowledge of statistical physics.

For some examples, look at the `optim` function in R, e.g.

```
?optim
example(optim)
```

and for animations, script `SimulatedAnnealing.R` on the website.

2 Markov Chain Monte Carlo sampling schemes

The idea of *Markov Chain Monte Carlo* is to simulate from a probability distribution as the stationary distribution of a Markov process. This is normally employed for quite highly structured problems, typically involving large numbers of dependent random variables. Such problems first arose in statistical physics, and the ideas were re-discovered in spatial statistics in the 1970s and 1980s. Then those wanting to implement Bayesian models jumped on the bandwagon around 1990, rarely giving credit to those whose work in spatial statistics they had taken the ideas from.

The key questions about MCMC from a practical viewpoint are

1. How do we find a suitable Markov process with our target distribution π as its stationary distribution?
2. Assuming we cannot start from the stationary distribution (since if we could we would know another way to simulate from the process), how rapidly does the process reach equilibrium? And how can we know that it is already close to equilibrium?
3. How correlated are successive samples from the process, or (to put it another way), how far apart do we need to take samples for them to contain substantially different information?

These points are all interrelated—a good MCMC sampling scheme will be one for which each step is computationally quick, and which *mixes* well, that is traverses the sample space quickly.

Some of the statements made here about convergence need technical conditions which are omitted. It is generally accepted that the cases that are being excluded are pathological, and since MCMC allows a lot of freedom to design a suitable scheme the conditions are easily satisfied in practice. The clearest and most accessible account of the relevant theory I have seen is Roberts & Rosenthal (1998).

Strauss processes

My idea for simulation was to make use of a *spatial birth-and-death* process which we had recently heard about in some graduate lectures from Chris Preston. Here is a stripped-down version.

Suppose we have a population of N objects in study area A. Each of them has an exponential lifetime until death, and objects are being born at a given rate. Both the rate of birth and the rate of death can depend on the current crowding. This is clearly a Markov process, and fairly clearly stationary under modest conditions (as the more points there are, the faster the overall rate of death, so the population cannot explode). Also, we can choose (in several ways) the birth and death rates so that the stationary distribution is the variable-n version of (1).

The idea of using a Markov process to simulate from its stationary distribution was original to us (but not new to the world) and is now called *Markov Chain Monte Carlo*. The term is reasonably appropriate, for it seems *Monte Carlo sampling* arose as a code phrase at Los Alamos just after WWII—and MCMC is usually attributed to a five-author paper (Metropolis *et al.*, 1953) from a group at Los Alamos. What they proposed is now known as *Metropolis*

sampling. Note though that the Markov processes used are most often on a continuous state space and so not Markov *chains* in the most common definition (see Appendix A).

Suppose we restrict our birth-and-death process to just n or $n-1$ objects. With n objects we can only have a death, and we take a uniform death rate (so pick an object at random, and kill it). Now we have $n-1$ objects, so we must have a birth, and add point ξ with density

$$f(\xi \mid \boldsymbol{x}) \propto \frac{c^{t(\boldsymbol{x} \cup \{\xi\}, R)}}{c^{t(\boldsymbol{x}, R)}} \quad . \tag{4}$$

Now $t(\boldsymbol{x} \cup \{\xi\}, R)$ is the number of R-close pairs when we add an object at ξ, and this is the number of R-close pairs in $\boldsymbol{x}$ (constant) plus the number of objects in $\boldsymbol{x}$ which are R-close to ξ. More verbosely, we consider adding a new object, count the number of existing objects within distance R of the new object, and arrange to keep the new object with probability proportional to c raised to the power of the count.

By combining a death and a birth as a single step we have a Markov chain on sets of n objects, and we showed that its stationary distribution is (1).

This is in fact a quite general way to sample from a multivariate distribution, and I published it as such in Ripley (1979). It was rediscovered by and published in Geman & Geman (1984) as the *Gibbs sampler*. The general description for the joint distribution of n variables is to select one variable at random, and re-sample it from its conditional distribution given all the others.

It may not be obvious that we have made progress, as (4) also has an omitted normalizing constant. However, this is just a 2-dimensional integral, and there are methods of simulation which just need an integrable upper bound such as *rejection sampling*. For $c \leq 1$ it is clear that the RHS of (4) is bounded by 1 (and we may be able to find a better bound). So you might like to check for yourself[5] that one way to sample from (4) is to draw a uniformly distributed object from A, and accept it with probability $c^{\{t(\boldsymbol{x} \cup \{\xi\}) - t(\boldsymbol{x})\}}$ or try again. (In case anyone wonders why we do not apply rejection sampling for the whole process – for the sort of datasets we are interested in the probability of acceptance would be astronomically small, and in this small example it is about 3×10^{-11}.)

There are all sorts of practical considerations, but this provided us with a way to simulate our process for the sort of parameter values we were interested in that was just about fast enough for our limited computing resources[6], and hence to begin to answer our questions.

Be aware that there are often a lot of choices in designing MCMC methods – the one I have sketched is good for $c < 1$, but not for $c > 1$, for example. People have since found ways using MCMC to simulate it exactly (Berthelsen & Møller, 2003) but very much more slowly.

Data augmentation

Suppose we have a parametric model $p(Y \mid \theta)$ for some observable random variables Y. It is rather common for this to be the manifestation of a richer model $p(Y, Z \mid \theta)$ for both the *manifest* variables Y and some *latent* (unobserved, 'missing') variables Z. This can arise in many ways, including

[5] this is an example of rejection sampling, so you need write it in the standard form and check the distribution it samples from is indeed the correct one.

[6] a few CPU seconds a day on a mainframe far slower than today's PCs: see Ripley (2005).

- Missing data, so Z represents e.g. responses from a survey that were unobserved.
- Partial observation, e.g. in social networks we only observe the links at some times: in family studies we have genetic data on only some members.
- Censored data, e.g. lifetimes in which all we know for some subjects is that they were still alive on a particular date. So for each subject we have two pieces of information, whether they were alive at the end of the study, and the actual date of death. For all subjects the first is part of Y whereas for some the second is part of Y and for some part of Z.
- Latent variable/class problems in which Z is some unobserved 'true' characteristic such as intelligence or the component of a mixture distribution. In genetics Y might be the phenotype and Z the genotype.

For simplicity of exposition we will take a Bayesian viewpoint with a prior probability distribution on θ, and the main object of interest is then the posterior distribution $g(\theta) = p(\theta \mid Y)$. Note that

$$g(\theta) = p(\theta \mid Y) = \int p(\theta \mid Y, Z) p(Z \mid Y)\, dZ$$

and

$$P(Z \mid Y) = \int p(Z \mid \theta, Y) p(\theta \mid Y)\, d\theta$$

and hence g satisfies

$$g(\theta) = \int K(\theta, \phi)\, g(\phi)\, d\phi, \quad \text{where} \quad K(\theta, \phi) = \int p(\theta \mid Y, Z)\, p(Z \mid \phi, Y)\, dZ \tag{5}$$

Under mild conditions we can solve (5) by successive substitution,[7] but we do have to integrate out the unobserved variables Z. Tanner & Wong (1987) (see also Tanner, 1996) call a Monte Carlo version *data augmentation*. This alternates the steps

a. Generate a sample (z_i) of size m from the current approximation to $p(Z \mid Y)$. This will probably be done by first sampling θ_i^* from the current approximation $g(\theta)$ and then sampling z_i from $p(z \mid \theta_i^*, Y)$.

b. Use this sample to update the approximation to $g(\theta) = p(\theta \mid Y)$ as the average of $p(\theta \mid z_i, Y)$.

So what this is doing is approximating $p(\theta \mid Y)$ by a finite mixture from $(p(\theta \mid z, Y))$. As iteration progresses we might want to take larger and larger samples to get better approximations.

This is closely related to the notion of *multiple imputation* in the analysis of sample surveys, where missing data are replaced by a sample of their uncertain values. So data augmentation alternates between multiple imputation of the unobserved variables in the model and inference based on the augmented data. From a theoretical viewpoint, the multiple imputations are being used to approximate the integral in the definition of K at (5) by an average over samples.

However, we can take another point of view, as K is the transition kernel of a Markov chain, and successive substitution will converge to the stationary distribution of that Markov chain. Suppose that we just simulate from the Markov chain? This alternates

[7] Start with some candidate g for $p(\theta \mid Y)$, and repeatedly use (5) to obtain a new and better candidate. Under mild conditions this does work – there is a unique solution, the new candidate is closer in L_1 norm to that solution and convergence is geometric.

a. Generate a single sample z from $p(Z \mid \theta, Y)$ with the current θ.

b. Use z to sample θ from $p(\theta \mid z, Y)$.

In this version we give up both multiple imputation and any attempt to keep probability distributions in partially analytical form—rather we represent distributions by a single sample, and run the Markov chain as a stochastic process on parameter values θ (rather than iterating an integral operator). This variant is called *chained data augmentation* by Tanner (1996). Clearly we would eventually want more than one sample, but we can get that by simulating the whole Markov chain multiple times, rather than simulating each step multiple times.

In a *particle filter*[8] evolving distributions are represented by a finite set of values, not just one, that is by a finite mixture, usually but not always unweighted.

The observable data Y have played a passive rôle throughout this subsection: what we have been considering is a way to simulate from the joint distribution of (θ, Z) conditional on Y. So we do not need an explicit Y, and 'chained data augmentation' gives us a way to simulate from any joint distribution of two (groups of) random variables by alternately simulating from each of the two conditional distributions of one conditioned on the other.

Logistic and probit regression models

We can illustrate data augmentation by the examples of logit and probit regression models. These can be described as taking a *linear predictor* $\eta = X\beta$ which gives the location of a logistic or normal random variable, the *propensity* Z. The propensity is unobserved, hence *latent*: only its sign is observed, and conventionally 'success' corresponds to a negative sign.

This is a natural problem for an MCMC algorithm based on data augmentation, and for the probit case it was given by Albert & Chib (1993). The conditional distribution of Z given the data and β is a truncated normal distribution (truncated because we have observed the sign), and so easy to sample from, *provided* we choose a suitable (conjugate, multivariate normal) prior for β.

For a logit model the conditional distribution of Z is no longer of known form. Holmes & Held (2006) overcome this by extending the model, and regarding the logistic distribution as a scale-mixture of normals: thus for each observation we get a pair of latent variables (Z, ϕ) where Z is again normal with random multiplier ϕ. Again for a multivariate normal prior on β they offer a (rather complicated) data augmentation algorithm to sample from the posterior.

Gibbs sampler

so named by Geman & Geman (1984).

This applies to a multivariate distribution, so we can think of Y as m-dimensional. The simplest Gibbs sampler consists of selecting a random component i of Y, and replacing Y_i by a sample from $p(Y_i \mid Y_{-i})$, where Y_{-i} denotes all the variables *except* Y_i.

[8] `http://en.wikipedia.org/wiki/Particle_filter`, Robert & Casella (2004, Chapter 14).

Chained data augmentation is a simple example of the Gibbs sampler. It alternately samples from the conditional distributions of Z and θ given the remaining variables.

In practice the Gibbs sampler is often used with a systematic selection of i rather than a random one (as in chained data augmentation). The theory is then not so simple as the process is no longer necessarily reversible—this is discussed in Geman & Geman (1984) and some[9] of the references. One simple modification that makes the process reversible is to use a systematic order of the m components, and then run through them in reverse order (chained DA is an example).

When we have an m-dimensional distribution, it is not necessary to think of each component in the Gibbs sampler as a single random variable. Sometimes the variables naturally form blocks, and it is the blocks to which the Gibbs sampler should be applied. Once again, chained DA provides the simplest example.

Note that the Gibbs sampler does not necessarily converge to the stationary distribution: there are conditions which need to be checked and are related to when a joint distribution is determined by all of its univariate conditionals. Consider the simple example of a two-dimensional joint distribution of (X, Y) in which X has a standard normal distribution and $Y = X$.

Metropolis-Hasting schemes

A general way to construct a Markov chain with a given stationary distribution π was given by Metropolis *et al.* (1953) which was given added flexibility by Hastings (1970).

These MCMC schemes start with a transition kernel $q(x, y)$ of a Markov process on the state space. Given a current state Y_t this is used to generate a candidate next state Y^*. Then *either* the transition is accepted and $Y_{t+1} = Y^*$ *or* it is not when $Y_{t+1} = Y_t$. The probability that the move is accepted is $\alpha(Y_t, Y^*)$ where

$$\alpha(x, y) = \min\left\{1, \frac{\pi(y)q(y, x)}{\pi(x)q(x, y)}\right\}$$

It is simple to show that the stationary distribution id π. For the stationary distribution to be also the limiting distribution we need the chain to be aperiodic: note that it *will* be aperiodic if there is a positive probability of rejecting a move.

The original Metropolis *et al.* scheme had a symmetric transition kernel, so the move is accepted with probability $\min\{1, \pi(x)/\pi(y)\}$. That is, all moves to a more or equally plausible state are accepted, but those to a less plausible state are accepted only with a probability less than one, the ratio of the probabilities.

That only the ratio of the probabilities enters is often exploited. If x is a high-dimensional state vector, choosing transitions such that y differs from x only in one or a few components can simplify greatly the computation of $\pi(Y^*)/\pi(Y_t)$, and also avoid rejecting most proposed moves (which will happen if $\pi(Y^*)$ is almost always very much smaller than $\pi(Y_t)$). Indeed, the Gibbs sampler is a special case of the Metropolis-Hastings sampler in which only single-component moves are considered, and $q(x, y) = p(x_i \mid x_{-i})$ where i is the chosen component (and hence $\alpha(x, y) \equiv 1$).

[9] e.g. Gamerman & Lopes (2006, §5.3.2).

A couple of other special cases are worth mentioning. One suggested by Hastings (1970) and others is a *random-walk sampler* in which q specifies a random walk (and so makes most sense when the state space is a vector space, but could apply to a lattice). Another is an *independence sampler* in which $q(x, y) = q(y)$, so the proposed move is independent of the current state.

For a gentle introduction to the many choices in implementing a Metropolis-Hastings MCMC scheme see Chib & Greenberg (1995).

So-called *Metropolis within Gibbs* schemes use a Metropolis MCMC sampler (usually a random walk sampler) for an update step of a Gibbs sampler. We do not need to run the Metropolis sampler until convergence and early proposals suggested a few steps, say 5. Nowadays most often only one Metropolis update is used at each Gibbs step, which is also a valid MCMC scheme. (See Gamerman & Lopes (2006, §6.4.2) and Robert & Casella (2004, §10.3.3).)

Slice sampling

Slice sampling is an MCMC sampling scheme that is sometimes used for a single update step in the Gibbs sampler: for fuller details see Neal (2003) and Robert & Casella (2004, Chapter 8).

We only consider sampling a univariate variable with pdf proportional to f. Consider the region $U = \{(x, y) \,|\, 0 \le y \le f(x)\}$, which has finite area (area one if f really is the pdf) so we can sample (X, Y) uniformly from U, and X has the desired distribution. Sampling uniformly from U is not easy in general, and generic Gibbs sampler software needs to sample from pretty arbitrary univariate distributions. So we consider using a Gibbs sampler to sample uniformly from U: this alternates steps

- Sample $Y \sim U(0, f(x))$ for the current value x of X.
- Sample X uniformly from $S(Y) = \{x \,|\, f(x) = Y\}$, a horizontal *slice* through the graph of f, hence the name of the sampler.

The difficulty is to find $S(y)$, which (at least for continuous f) is a union of intervals. If the support[10] of the pdf is within a finite interval $I = (L, R)$, we can use rejection sampling from a uniform variable on I, but this may be inefficient and does not work with unbounded support, so increasingly complex schemes have been suggested.

A Java applet illustrating slice sampling where the computation of $S(y)$ is trivial can be found at `http://www.probability.ca/jeff/java/slice.html`. This is for $f(x) \propto \exp x^{-1/d}$: try small values of $a = 1/d$, which is the example of slow convergence by Roberts & Rosenthal mentioned on Robert & Casella (2004, p. 332).

A R demonstration script is available on the module's website.

Other schemes

The only limit on the plethora of possible MCMC schemes is the ingenuity of developers. A similar idea to spatial birth-and-death processes, the *reversible jump MCMC* of Green (1995),

[10] The support is $\{x \,|\, f(x) > 0\}$.

has been applied to model choice in a Bayesian setting.

Silce sampling is an example of a class of *auxiliary variable* schemes in which we add artificial random variables either to get simpler steps or to walk around the sample space ('mix') better.

We do not even need to confine attention to Markov processes which jump: Grenander & Miller (1994) and others have used Langevin methods, that is diffusions. See Robert & Casella (2004, §7.8.5) for a brief account.

Further reading

MCMC can be approached from wide range of viewpoints – from theoretical to practical, as a general technique or purely Bayesian, and at different levels (especially in probability background). Texts which have interesting perspectives include Chen *et al.* (2000), Gamerman & Lopes (2006), Gelman *et al.* (2004), Gilks *et al.* (1996), Liu (2001) and Robert & Casella (2004). Roberts & Tweedie (2005) cover the Markov chain theory. As a topic in simulation, it is covered in Ripley (1987) and Dagpunar (2007),[11] and as a method of integration in Evans & Swartz (2000).

Albert (2009), Carlin & Louis (2009), Jackman (2009) and (especially) Gelman *et al.* (2004) provide accessible introductions to the computational aspects of applied Bayesian work with non-trivial worked examples.

[11] and at a higher mathematical level, Asmussen & Glynn (2007).

3 Using a MCMC sampler

So far we have described using a Markov chain to obtain a single sample from a stochastic process by running it for an infinite number of steps. In practice we run it for long enough to get close to equilibrium (called a 'burn-in' period) and then start sampling every $m \geq 1$ steps (calling *thinning*). We can estimate any distributional quantity *via* the law of large numbers

$$H_N = \frac{1}{N} \sum_{i=1}^{N} h(X_{mi}) \to E\, h(\boldsymbol{X})$$

for any m, so if $h()$ is cheap to compute we may as well average over all steps. In practice we often take m large enough so that samples are fairly dissimilar—thinning is also used to reduce storage requirements.

There are many practical issues – where do we start? How do we know when we are 'close to equilibrium'? And so on. Note that the issue of whether we are yet close to equilibrium is critical if we are simulating to get an idea of how the stochastic process behaves – Geman & Geman (1984) based all their intuition on processes which were far from equilibrium, but incorrect intuition led to interesting statistical methods.

A run of an MCMC algorithm provides a time series of correlated observations. There is a lot of earlier work on analysing such time series from other simulation experiments, for example of queueing problems: see Ripley (1987, Chapter 6). Most of these need a Central Limit Theorem, which holds if the Markov chain is geometric ergodic, for example. (Roberts & Rosenthal (1998, p.10) give an example of an MCMC scheme where the CLT fails to hold.) The most-used results are that (Ripley, 1987, p. 144)

$$N\text{var}H_N \to \sigma^2 \left[1 + 2\sum_{1}^{\infty} \rho_s\right] = 2\pi f(0)$$

where σ^2, (ρ_s) and f are the variance, the autocorrelation sequence and spectral density[12] of the stationary time series $h(X_{mi})$. When you see 'Time-series SE' in the CODA summary, it is estimated using the RHS[13] of this formula. This is (for large N) the same variance we would have with M independent samples, where

$$M = N/\left[1 + 2\sum_{1}^{\infty} \rho_s\right]$$

is known as the *effective sample size*.

Convergence diagnostics

Or 'How do we know when we are close to equilibrium?' Sometimes called *monitoring* (a term from the BUGS literature).

[12] or at least one definition of it!

[13] with $f(0)$ estimated by fitting a lower-order polynomial (default linear) near the origin to to the periodogram as estimated by `spectrum`.

This led to much heated discussion in the early 1990s, and several survey papers. The scale of the problem is often dramatically underestimated – twenty years ago we found an example (Ripley & Kirkland, 1990) in which the Gibbs sampler appeared to have converged after a few minutes, but jumped to a very different state after about a week. In statistical physics such behaviour is sometimes call *metastability*.

The proponents have split into two camps, those advocating running a single realization of the chain, and after a 'burn-in' period sampling it every m steps, and those advocating running several parallel realizations, and taking fewer samples from each. Note that the computing environment can make a difference, as the simplest and computationally most efficient way to make use of multiple CPUs is to use parallel runs.

If we knew something about the rate of convergence of the Markov chain to equilibrium we could use such knowledge to assess how long the 'burn-in' period needed to be. But this is very rarely helpful, for

(i) we rarely have such knowledge,

(ii) when we do it is in the form of upper bounds on convergence rates and those upper bounds are normally too crude, and

(iii) the theory is about convergence from any initial distribution of all aspects of the distribution. Many of the MCMC schemes converge fast for some aspects of the target distribution and slowly for others—hopefully the scheme was chosen so that the former are the aspects we are interested in.

Nevertheless, there are some exceptions: e.g. simple ones in Roberts & Rosenthal (1998, §5) (some of which apply to slice sampling of log-concave densities: Robert & Casella (2004, §8.3)) and an application to randomized graph-colouring algorithms in Jerrum (1995) (see also Asmussen & Glynn, 2007, §XIV.3).

After all those notes of caution, here are some of the main ideas. Let (X_t) be the output from a single MCMC run, possibly sub-sampled every m steps and of one (usually) or more aspects of interest.

- *Tests of stationarity*. If the output is stationary, we can divide into two or more parts which will have the same distribution, and apply a test for equality of distribution such as the Kolomogorov–Smirnov test. Such tests are usually most sensitive to changes in location (which is normally of most interest here), and for IID samples (and so need adjustment, as done by Heidelberger & Welch (1983) and Geweke (1992)). Tests of drift such as CUSUM charts (Yu & Mykland, 1998) come into this category.

- *Regeneration*. Some of the most powerful ideas in the analysis of discrete-event simulations (Ripley, 1987, Chapter 6) are based on the idea that the process will from time-to-time come back to an identifiable state and excursions from that state are independent (by the strong Markov property). (Think for example of a queueing system emptying completely.) Regeneration may be too rare to be useful, but this is one of the few fully satisfactory approaches.

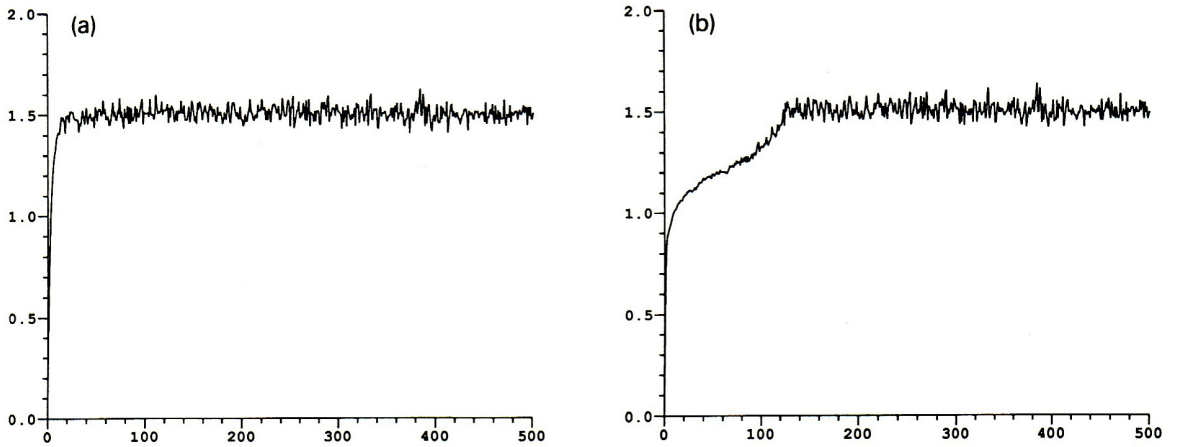


Fig. 2. Pseudolikelihood (a) and (asymptotic) maximum likelihood (b) estimates for 500 sweeps of Metropolis' method.

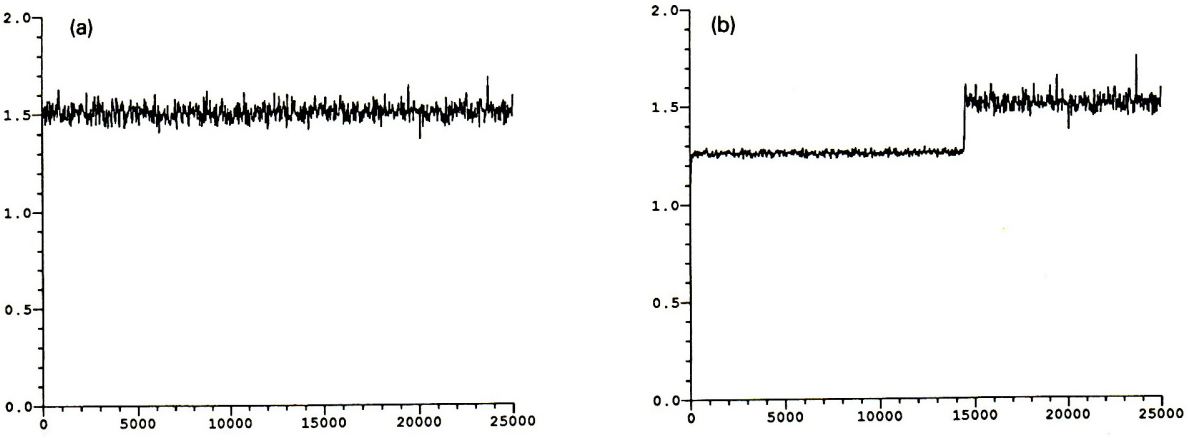


Fig. 3. As Fig. 2, but another sample with 25,000 sweeps!

Figure 3: Diagnostic plots from two realizations of an MCMC simulation. Note the different scales. These are for two estimators of a quantity known to be $\beta = 1.5$. From Ripley & Kirkland (1990).

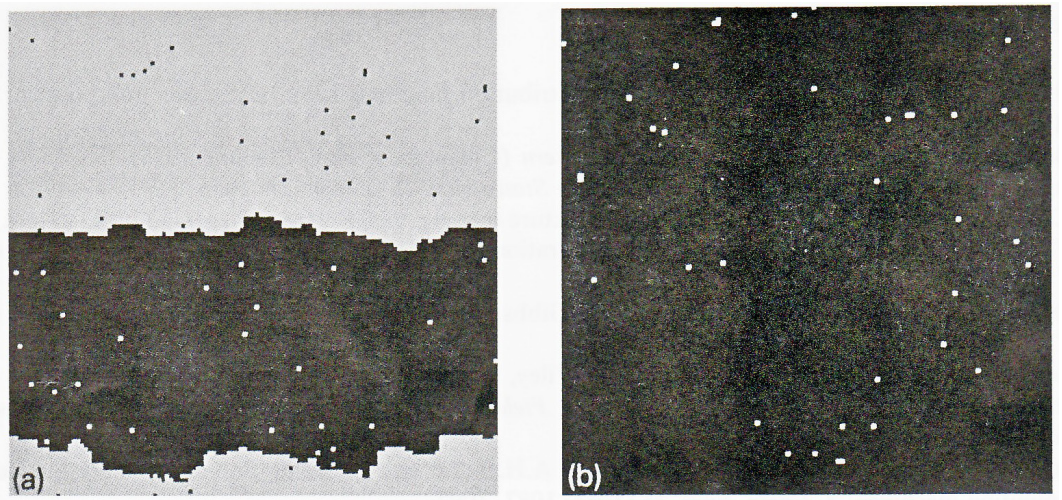


Fig. 4. The realization after 10,000 (a) and 25,000 (b) sweeps of Metropolis' method.

Figure 4: Two snapshots of the second MCMC simulation. From Ripley & Kirkland (1990).

- *Coverage*. The idea is to assess how much of the total mass of π has been explored. For a one-dimensional summary and sorted values $X_{(t)}$ the Riemann sum

$$\sum_{t=1}^{T} |X_{(t+1)} - X_{(t)}| \, \pi(X_{(t)})$$

  provides an approximation to $\int \pi(x)dx = 1$, and so its convergence to one is a measure of coverage of the MCMC to date. This is only applicable if there is a one-dimensional summary of which we know the marginal distribution explicitly (so we can evaluate $\pi(X_{(t)})$), and it only tells us about coverage of that marginal.

- *Multiple chains*. If we have a small number of runs from suitable starting points we can compare the variability within and between runs, and when the between-run variability has reduced to that predicted from the within-run variability all the runs should be close to equilibrium. The series (X_t) is autocorrelated, and we need to take that into account in assessing the within-run variability: but that is a standard problem in the simulation literature. This approach is principally associated with Gelman & Rubin (1992). The problem is to choose suitable starting points so the runs considered do representatively sample π.

- *Discretization*. Some methods look at a discretization of (X_t) to a process with a small number of states. The original proposal by Raftery & Lewis (1992) was reduce to a two-state process. The discretized process will not normally be Markov, but a sub-sampled process (every m steps) might be approximately so and *if so* we know enough about two-state Markov chains to study their convergence, estimating the two parameters of the transition matrix from the observed data. The issues are the Markov approximation and whether convergence of the discretized version tells us enough useful about convergence of the original (although non-convergence definitely does).

Another cautionary note: these diagnostic tests must not be used as stopping rules, as that would introduce bias.

Quite a lot of software has been written for convergence diagnostics. Two of the main suites, `coda` and `boa`, are available as R packages. Our examples use `coda`: see Appendix C.

A less brief introduction to convergence with an emphasis on the methods in `coda` is given in Robert & Casella (2010, §8.2–4).

There are some methods for using MCMC to produce a sample exactly from π. Propp & Wilson (1996) called these *exact sampling*, but Wilfrid Kendall's term *perfect simulation* has stuck (see, e.g. Kendall, 2005). They cover only a limited set of circumstances and are most definitely computer-intensive.[14] So these are not techniques for mainstream use (and probably never will be), but they could be used for example

- as a reference against which to compare cheaper simulation schemes, and
- to provide a small number (e.g. one) of samples from which to start an MCMC sampler.

See also Asmussen & Glynn (2007), Casella *et al.* (2001) and Robert & Casella (2004, Chapter 13). One possibly more practical idea that arises from Propp & Wilson's work is the idea of

[14] I understood (from Persi Diaconis) that Propp & Wilson ran a simulation for *six weeks* without any knowledge of how long it would actually take to reach an exact sample.

monotonicity of MCMC samplers. Suppose there are some extreme states for the distribution of interest, e.g. an image coloured entirely white or black. Then if we start an MCMC scheme at those states, and the realizations become 'similar', there is some hope that realizations starting from any initial state would have become similar by that time. 'Monotonicity' provides a theoretical guarantee of this and it (or similar ideas) underlies most perfect sampling schemes.

Software

Because MCMC is a meta-algorithm, there are very many specific applications and corresponding software: I counted 34 such R packages on CRAN just by looking at their `DESCRIPTION` files. (See also `http://cran.r-project.org/web/views/Bayesian.html` and the March 2006 issue of *R News* at `http://www.r-project.org/doc/Rnews/Rnews_2006-1.pdf`.)

The exercises and practicals will use R package `MCMCpack`, which is a collection of MCMC algorithms for Bayesian inference on specific models, coded in C++ with an R interface and producing output suitable for analysis by `coda`. The choice of models is slanted towards social science applications,[15] but includes linear regression, logit, probit, log-linear and factor analysis models.

Creating general software for MCMC is close to impossible, and all attempts known to me restrict themselves in one or both of two ways. Some confine attention to a family of sampling schemes—e.g. the grandly-named R package `mcmc` works with *"normal random-walk" Metropolis* and perhaps the best-known software, BUGS, works with the Gibbs sampler. Others confine attention to a particular class of statistical models and to a particular way to approach inference on those models. One common restriction is to the Bayesian analysis of hierarchical or graphical models.

BUGS[16] was a program developed from 1989 at MRC's Biostatistics Unit in Cambridge, coded in an arcane language that has restricted the platforms it could run on. It used an R-like language (see Appendix B) to specify graphical models for which it then creates a Gibbs sampling scheme, plus the ability to simulate from the created sampler. That version is now known as 'classic BUGS' and spawned WinBUGS[17] (Lunn *et al.*, 2000) with a GUI interface: this was then re-implemented as OpenBUGS.[18] A book about BUGS by its authors, Spiegelhalter *et al.* (2010), is said to be forthcoming (but that was also said in 2008). Ntzoufras (2009) describes how to use WinBUGS as a standalone Windows program. See Lunn *et al.* (2009) for some of the history of BUGS.

The blurb for Jackman (2009) claims

> *It features examples of how to implement the methods using WinBUGS – the most widely used Bayesian analysis software in the world – and R – an open-source statistical software.*

[15] such as item response models

[16] **B**ayesian inference **U**sing **G**ibbs **S**ampling.

[17] which as its name suggests is for (32-bit) Windows only, `http://www.mrc-bsu.cam.ac.uk/bugs`. People have managed to run it on `ix86` Linux *via* WINE and on Mac OS X *via* WINE or CrossOver—but others have failed.

[18] `http://www.openbugs.info/`; this is Open Source but is *de facto* also restricted to Windows—there is a `i386` Linux version that few people have managed to get to work. OpenBUGS is compiled by Oberon Microsystems' 'BlackBox' development system, which is offered only for 32-bit Windows.

although I have no idea how one could know that a progam is 'most-widely used', or even how widely a freely-available program is *used*. Lunn *et al.* (2000) quoted 'over 2,000 registered users', and the WinBUGS website mentions 'over 30,000 downloads'. (There are estimates ranging from tens of thousands to 2 million R users.)

JAGS[19] is an Open Source program by Martyn Plummer written in C++ that re-implements the BUGS language. It is much more recent and at present does not have so well-tuned internal algorithms so can be slower: conversely it has a richer language and is much easier to extend. It does have the great advantage of running on Linux, Mac OS X, Solaris ... as well as (32-bit and 64-bit) Windows.

From the BUGS website

> Health warning
>
> *'The programs are reasonably easy to use and come with a wide range of examples. There is, however, a need for caution. A knowledge of Bayesian statistics is assumed, including recognition of the potential importance of prior distributions, and MCMC is inherently less robust than analytic statistical methods. There is no in-built protection against misuse.'*
>
> About JAGS:
>
> *'JAGS uses essentially the same model description language but it has been completely re-written. Independent corroboration of MCMC results is always valuable!'*

Note that all the BUGS-like programs require a proper Bayesian model, so exclude improper priors. There are frequent references in Gelman & Hill (2007) to crashes[20] in 'Bugs' (presumably WinBUGS) with difficult-to-fit models, and it seems that at least WinBUGS and OpenBUGS are less robust to numerical issues than programs such as R. One of those numerical issues seems to be attempts to specify very diffuse (but still proper) priors as surrogates for improper priors.

There are R packages[21] `BRugs`, `R2WinBUGS` and `rjags` to interface with OpenBUGS, WinBUGS/OpenBUGS and JAGS respectively. Gelman *et al.* (2004, Appendix C) discusses driving WinBUGS from R *via* `R2WinBUGS`: it is available on-line at `http://www.stat.columbia.edu/~gelman/bugsR/software.pdf`.

We will also look briefly at R package `LearnBayes`, which is a companion to Albert (2009): the latter includes examples of MCMC both *via* `LearnBayes` and *via* `WinBUGS`.

[19] `http://www-fis.iarc.fr/~martyn/software/jags/`.

[20] *'Nobody could get the developers of BUGS under the trade descriptions act.'* quoth `http://www.senns.demon.co.uk/Confuseus.htm`.

[21] `BRugs` and `rjags` communicate with the OpenBUGS and JAGS DLLs directly and hence run these programs in the same process—`BRugs` includes a copy of the OpenBUGS DLL. `R2WinBUGS` sends scripts to a WinBUGS executable or calls `BRugs`.

These have been joined recently by `rbugs` (to send scripts to a OpenBUGS executable), `R2jags` (a wrapper for `rjags`) and `runjags` (to send scripts to a JAGS executable).

4 MCMC examples in Bayesian Statistics

Binomial logistic regression

The *Statistical Methods* module had an example on data from accidents on Ford Explorer SUVs. We can recap the basic classical analysis (without grouping the data) as

```
library(Sleuth2)
SUV <- ex2018
# we will need a 0/1 response later
SUV$Cause <- with(SUV, Cause == "Tire")
SUV$Make <- relevel(SUV$Make, ref = "Other")
fit <- glm(Cause ~ Make, data = SUV, family = "binomial")
summary(fit)
```

For a Bayesian analysis we can use

```
library(MCMCpack)
Bfit <- MCMClogit(Cause ~ Make, data = SUV, burnin = 1e3, mcmc = 1e4)
summary(Bfit)
```

which returns an object for exploration by the package `coda`. The summary output is something like (it is random):

```
Iterations = 1001:11000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 10000

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

              Mean     SD Naive SE Time-series SE
(Intercept) -5.985 0.4676 0.004676        0.01476
MakeFord     2.844 0.5101 0.005101        0.01521
...
```

This contains 10,000 steps of the Markov Chain after 1000 steps of burn-in (starting at the MLE). We can look at the output by

```
plot(Bfit) # or Bfit[, 2]
densityplot(Bfit)
```

and see how independent the realizations are by

```
acfplot(Bfit)
effectiveSize(Bfit)
```

So the effective size is about 1000 and the realizations are far from independent. For simplicity we do another run with thinning (which takes ca 15 seconds; if the code supported it, it would be much quicker to work with aggregated data):

```
Bfit <- MCMClogit(Cause ~ Make, data = SUV,
                  burnin = 1e3, mcmc = 2.5e4, thin = 25)
summary(Bfit)
```

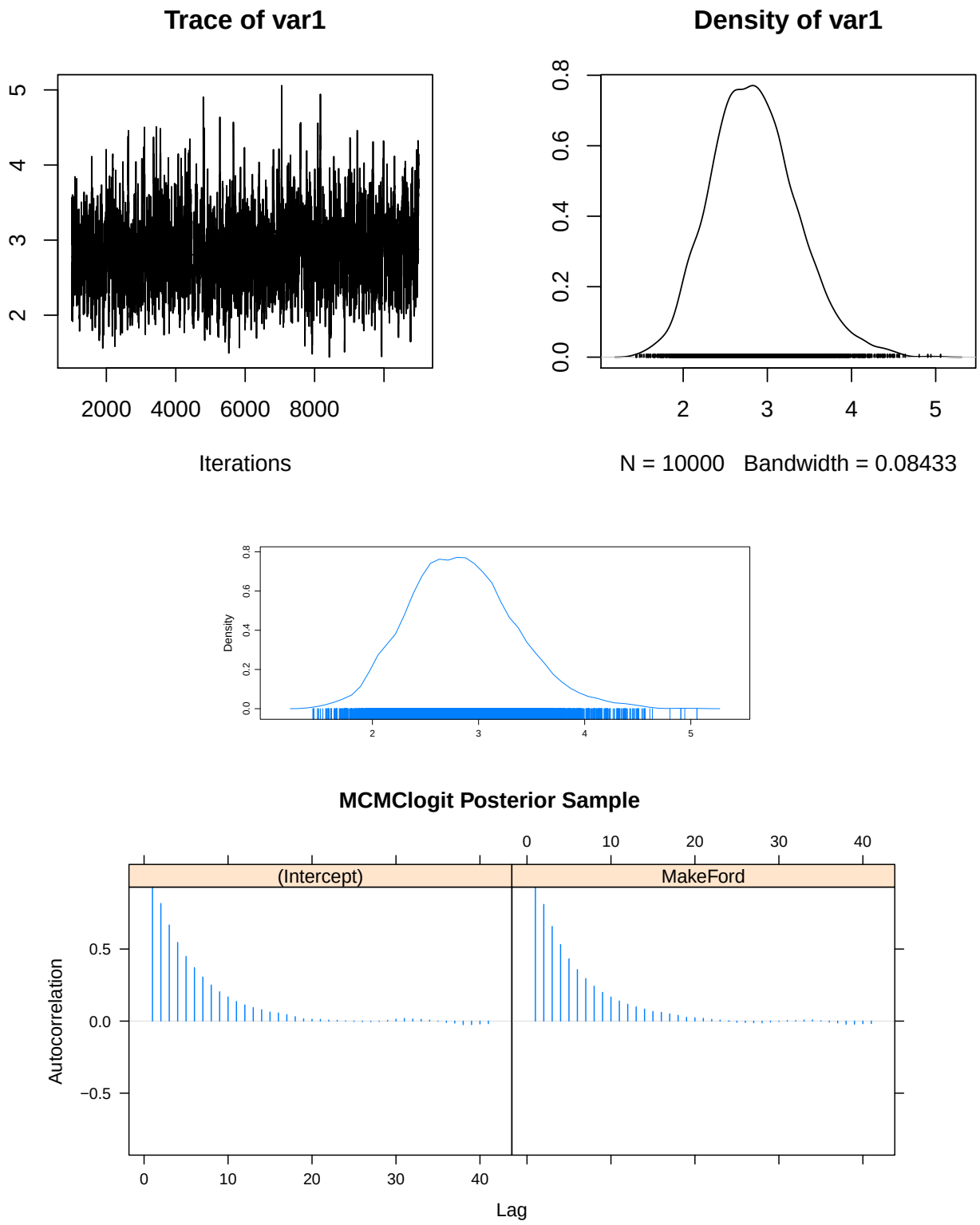


Figure 5: The output of `plot`, `densityplot` and `acfplot` for the logistic model fitted to the SUV data.

```
acfplot(Bfit)
densityplot(Bfit)
```

So now we can take a look at what are effectively 1000 independent samples of the odds:

```
odds <- exp(Bfit[, "MakeFord"])
plot(odds)
HPDinterval(odds)
```

which gives a 95% credible interval of $(5.8, 43.2)$.

Note that Bayesian intervals transform differently from classical ones: the posterior mean of the odds is 19.82 whereas $\exp \beta$ has mean 17.3, and exponentiating a 95% credible interval for β gives $(6.8, 44.5)$.

Notice that we did not specify a prior! However, someone did, and the default prior for `MCMClogit` is the improper uniform prior: this can be done because the posterior density is proper (and proportional to the likelihood). Then the MAP (*maximum a posteriori*) parameter estimate is equal to the MLE: one important difference from a classical analysis is that we use the marginal posterior density and not the profile likelihood when doing inference on the odds.

This analysis assumed product-binomial sampling. That's probably not the best assumption for this set of data—Poisson sampling is probably most appropriate. Of course, in a classical analysis the sampling does not affect the analysis: that may not hold in a Bayesian analysis with proper priors. Other analyses of this example are discussed in the practical class.

Biological assay

Venables & Ripley (2002, §7.2) explore the following example.

> Consider first a small example. Collett (1991, p. 75) reports an experiment on the toxicity to the tobacco budworm *Heliothis virescens* of doses of the pyrethroid *trans*-cypermethrin to which the moths were beginning to show resistance. Batches of 20 moths of each sex were exposed for three days to the pyrethroid and the number in each batch that were dead or knocked down was recorded. The results were

	Dose					
Sex	1	2	4	8	16	32
Male	1	4	9	13	18	20
Female	0	2	6	10	12	16

> The doses were in μg. We fit a logistic regression model using $\log_2$(dose) since the doses are powers of two.

The interest is in estimating the dose required for a particular probability p of death, especially that for $p = 0.5$ called LD50.

A frequentist analysis using `glm` is given in Venables & Ripley (2002), but here we consider a Bayesian analysis. For reference the classical results are, for $\log_2$ dose,

```
        Dose        SE
M: 2.238815 0.2039871
```

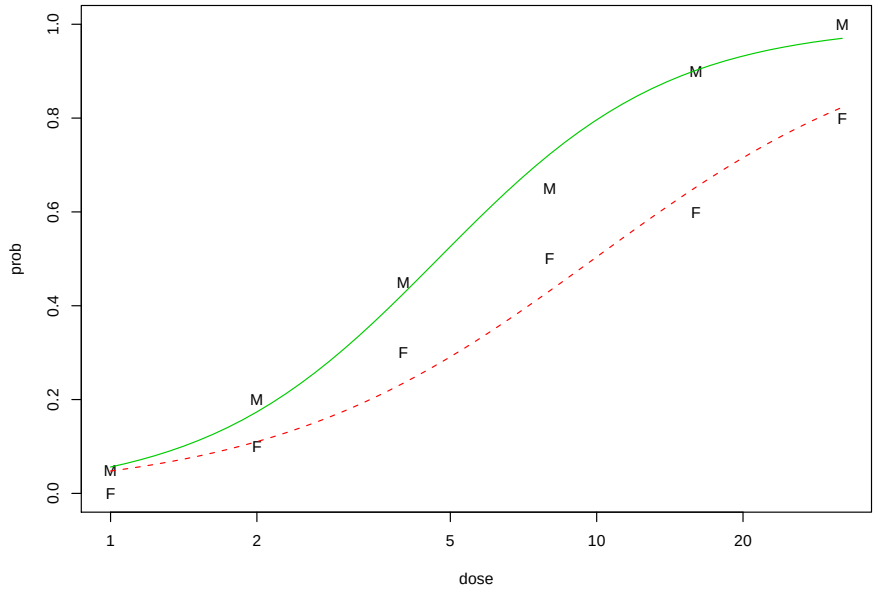


Figure 6: Probability of death of tobacco budworm moths *vs* dose (on log scale) of pyrethroid. The observed frequencies are marked by M (for male moths) and F (for female moths), together with fitted curves for separate logistic regressions.

```
F: 3.303997 0.2602836
```

We start by using R package `MCMCpack`: this works with Bernoulli and not binomial data, so first we disaggregate the results.

```
ldose <- rep(0:5, 2)
numdead <- c(1, 4, 9, 13, 18, 20, 0, 2, 6, 10, 12, 16)
sex <- factor(rep(c("M", "F"), c(6, 6)))
SF <- cbind(numdead, numalive = 20 - numdead)
resp <- rep(rep(c(1,0), 12), times = t(SF))
budworm <- data.frame(resp, ldose = rep(ldose, each = 20),
                      sex = rep(sex, each = 20))

library(MCMCpack)  # loads package 'coda'
fit <- MCMClogit(resp ~ sex*ldose, data = budworm)
summary(fit)
effectiveSize(fit)
plot(fit)
acfplot(fit) # suggests thinning
crosscorr.plot(fit)
fit <- MCMClogit(resp ~ sex*ldose, data = budworm, thin = 20)
summary(fit)
HPDinterval(fit)
```

Package `MCMCpack` is to some extent a black box: you get whatever MCMC scheme the authors decided to implement, and for some reason quite different schemes are used for logit (random-walk Metropolis) and the closely related probit regression (data augmentation *à la* Albert and Chib).

Now we look at the quantities of interest:

```
library(MASS)
contour(kde2d(fit[,1], fit[,3], n=50), xlab="alphaF", ylab="betaF")
points(fit[, c(1,3)], pch=20, cex=0.5, col=rgb(0,0,1,0.2))
## create some new objects, the LD50s in a form suitable for coda.
ld50F <- as.mcmc(2 ^ (-fit[,1]/fit[,3]) )
ld50M <- as.mcmc(2 ^ (-(fit[,1]+fit[,2])/(fit[,3] + fit[,4])) )
range(ld50M); range(ld50F)
ld50 <- mcmc(cbind(M=ld50M, F=ld50F))
plot(ld50)
acfplot(ld50)
HPDinterval(ld50)
```

There is an issue with LD50, pointed out by Gelman *et al.* (2004, p. 93): we are really only interested in positive slopes. In this example the chance of a negative fitted slope is negligible, but in theory LD50 is a non-linear function of the parameters, something simulation-based inference takes in its stride. So we can compute the samples for LD50 for female moths by

```
ld50F <- ifelse(fit[,3] > 0, 2^(-fit[,1]/fit[,3]), ifelse(fit[,1] > 0, 0, Inf))
```

(reporting a dose and not $\log_2$ dose).

Our second approach uses OpenBUGS *via* `BRugs`. We need to specify the BUGS model, which we will do in a file `budworm.bug`:

```
model {
   for(i in 1:6) {
      numdead[i] ~ dbin(p[i], 20)
      logit(p[i]) <- alphaM + betaM * ldose[i]
   }
   for(i in 7:12) {
      numdead[i] ~ dbin(p[i], 20)
      logit(p[i]) <- alphaF + betaF * ldose[i]
   }
   betaM ~ dnorm(0.0, 0.001)
   alphaM ~ dnorm(0.0, 0.001)
   betaF ~ dnorm(0.0, 0.001)
   alphaF ~ dnorm(0.0, 0.001)
}
```

This is simple rather than general, and specifies rather vague independent priors for the parameters. The syntax is deceptively similar to R, but note that `dnorm` has arguments mean and *precision* (reciprocal variance).

To run the MCMC simulation we use

```
library(BRugs)
inits <- list(list(alphaM = 0, betaM = 0, alphaF = 0, betaF = 0))
budworm.sim <-
    BRugsFit("budworm.bug", data = list("numdead", "ldose"), inits = inits,
             parametersToSave = c("alphaM", "alphaF", "betaM", "betaF"),
             numChains = 1,
             nBurnin = 5000, nIter = 5000, coda = TRUE, DIC = FALSE)
summary(budworm.sim)
plot(budworm.sim)
```

with printout

```
Iterations = 5001:10000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 5000

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

          Mean     SD Naive SE Time-series SE
alphaF -3.0897 0.5514 0.007797       0.016275
alphaM -2.9327 0.5755 0.008138       0.021347
betaF   0.9363 0.1663 0.002352       0.004735
betaM   1.3077 0.2249 0.003181       0.009128

2. Quantiles for each variable:

          2.5%     25%     50%    75%  97.5%
alphaF -4.2097 -3.4617 -3.0690 -2.701 -2.072
alphaM -4.1350 -3.2932 -2.9061 -2.534 -1.890
betaF   0.6221  0.8195  0.9328  1.049  1.276
betaM   0.9038  1.1532  1.2942  1.446  1.804
```

You should explore other starting points.

Looking at the posterior simulations shows a potential problem with naïve use of the Gibbs sampler—the intercept and slope are quite correlated. Only extreme correlations will give problems in a classical analysis of a GLM, but here quite modest correlations can slow down convergence of the automatically constructed Gibbs sampler. (The classic remedy in both cases is to centre the explanatory variables.)

Alternatively, using JAGS *via* `rjags`,

```
library(rjags)
## discard 5000, record 5000
inits <- list(list(alphaM = 0, betaM = 0, alphaF = 0, betaF = 0))
bd.jags <- jags.model("budworm.bug", inits = inits, n.chains = 1,
                      n.adapt = 5000)
vars <- c("alphaM", "alphaF", "betaM", "betaF")
bd.sims <- coda.samples(bd.jags, vars, n.iter = 5000)
summary(bd.sims)
plot(bd.sims)
```

Poisson change-point models

Consider the much-used data set of annual counts of British coal mining 'disasters' from 1851 to 1962.[22]

Looking at the data suggests that the rate of accidents decreased sometime around 1900, so a plausible first model is that the counts are independent Poisson with mean λ_1 before time τ

[22] These were derived from Jarrett (1979) and refer to explosions.

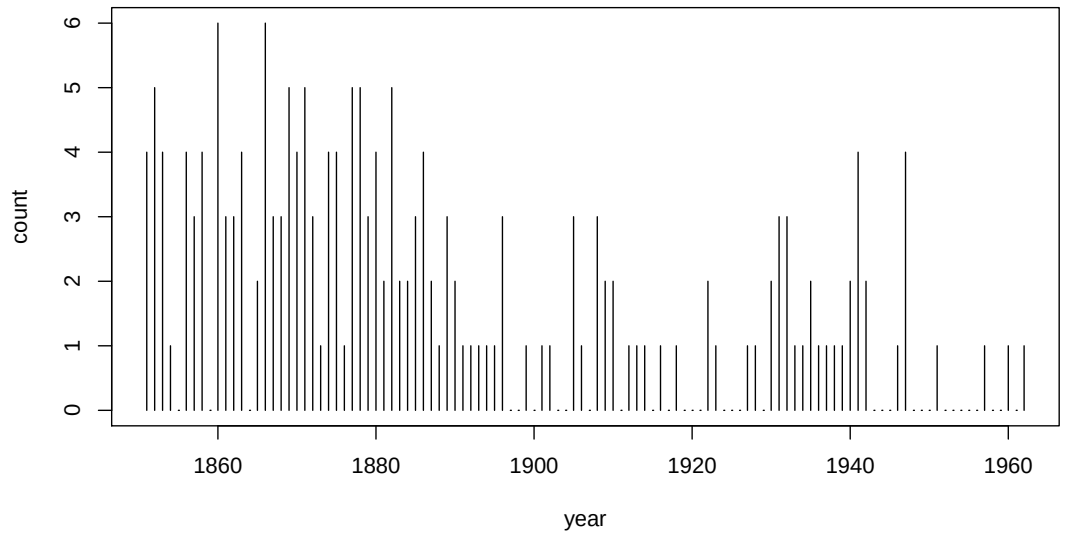


Figure 7: Number of explosions per year in British coal mines.

and mean λ_2 from time τ onwards, where we expect $\lambda_2 < \lambda_1$. This is the simplest possible case, and we could consider more than one changepoint.

For a Bayesian analysis we need a prior distribution on the three parameters $(\tau, \lambda_1, \lambda_2)$. If we take them as independent and of conjugate form, the posterior can be found analytically (Gamerman & Lopes, 2006, pp. 143ff), but a realistic prior will have a dependent distribution of (λ_1, λ_2). That is easy to do in the MCMC framework—for a more complex application to radiocarbon dating see Gilks *et al.* (1996, Chapter 25).

We will consider computing posterior distributions *via* MCMC in two ways. R package `MCMCpack` has a function `MCMCpoissonChange` with an MCMC scheme coded in C++, implementing the method of Chib (1998). This has independent gamma priors for the rates and beta priors for the transition point(s). The R code is simple:

```
## D is an integer vector of 113 counts.
library(MCMCpack)
fit <- MCMCpoissonChange(D ~ 1, m = 1, c0 = 1, d0 = 1,
                         burnin = 10000, mcmc = 10000)
plot(fit); par(mfrow=c(1,1))
summary(fit)
plotState(fit)
plotChangepoint(fit, start = 1851)
```

The arguments say that we are looking for `m = 1` changepoints, and specify a gamma$(1, 1)$ prior for the mean counts λ_i. In that approach, someone else has done all the work in designing and coding a suitable MCMC scheme—this is fast but not general.

Our second approach follows Albert (2009, §11.4) and uses vague priors. Rather than use specific code, we use BUGS and hence a Gibbs sampler somewhat tailored by the program to the problem. The first step is to tell BUGS what the model is using a model file containing

```
model {
  for (year in 1:N) {
       D[year] ~ dpois(mu[year])
```

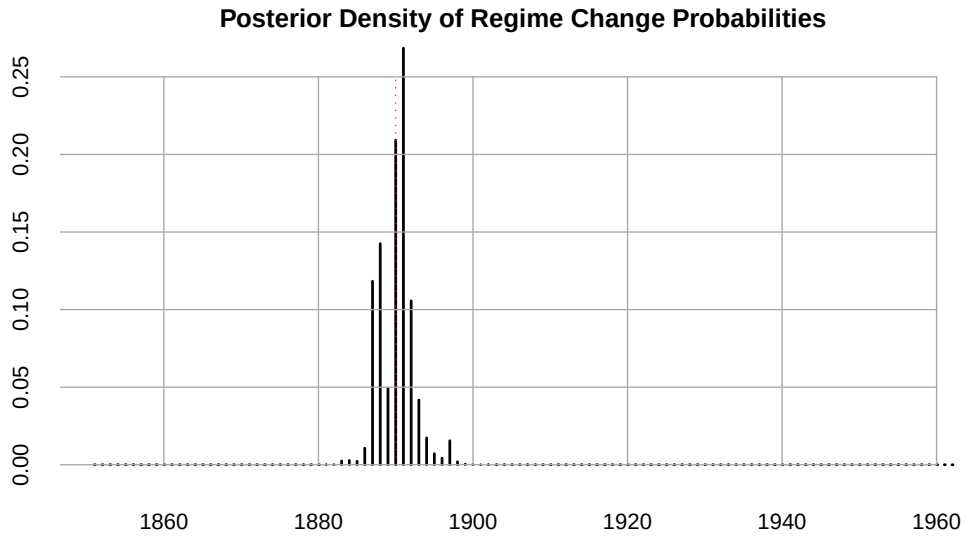


Figure 8: Posterior distribution for the year of change in disaster rate. The red dashed line marks the median.

```
        log(mu[year]) <- b[1] + step(year - changeyear) * b[2]
  }
  for (j in 1:2) { b[j] ~ dnorm(0.0, 1.0E-6) }
  changeyear ~ dunif(1,N)
}
```

This version uses a slightly different parametrization, with the first element of `b` as $\log \lambda_1$ and the second as $\log \lambda_2 - \log \lambda_1$. The data are the counts in `D` and the number of years, `N`.

We can then use function `BRugsFit` to ask OpenBUGS (*via* package `BRugs`) to simulate from the posterior distribution of this model, by e.g.

```
library(BRugs)
inits <- list(list(b=c(0,0), changeyear=50),
              list(b=rnorm(2), changeyear=30),
              list(b=rnorm(2), changeyear=70))
coalmining.sim <-
    BRugsFit(data = list("N", "D"), inits=inits, param=c("changeyear","b"),
             model="coalmining.bug",
             numChains = 3, nIter = 5000, DIC = FALSE, coda = TRUE)
```

As MCMC is an iterative scheme we have to supply initial values of the parameters: it is possible to supply (as a list of lists as here) separate starting values for each run, or a function that will give a random list result.

The result object can be summarized and plotted. The printout looks like

```
> summary(coalmining.sim)

Iterations = 1001:2000
Thinning interval = 1
Number of chains = 3
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
```

```
   plus standard error of the mean:

              Mean     SD Naive SE Time-series SE
b[1]         1.144 0.0941 0.001718       0.002286
b[2]        -1.233 0.1557 0.002842       0.003980
changeyear 39.584 2.1405 0.039079       0.052742

...
```

Regression models

Albert (2009, § 9.3) considers a dataset on the number of puffin nesting burrows in 38 study areas, with 4 explanatory variable. Although the data are counts, Albert treated this as a regression problem, and so initially will we. A classical analysis might be

```
library(LearnBayes); library(MASS)
fit <- lm(Nest ~ Grass + Soil + Angle + Distance, data = puffin)
summary(fit)
stepAIC(fit)
```

which drops `Grass` and almost drops `Angle`.

For a Bayesian analysis

```
library(MCMCpack)
Bfit <- MCMCregress(Nest ~ Grass + Soil + Angle + Distance, data = puffin,
                    burnin = 1000, mcmc = 25000, thin = 25)
summary(Bfit)
densityplot(Bfit)
```

and we will consider model selection in the next chapter. There we need a BUGS model, so we set it up here. The model file, `puffin.bug`, is

```
model{
    for(i in 1:38) {
         Nest[i] ~ dnorm(mu[i], tau)
         mu[i] <- alpha + beta1*Grass[i] + beta2*Soil[i] +
              beta3*Angle[i] + beta4*Distance[i]
    }
    beta0 ~ dnorm(0, 0.01)
    beta1 ~ dnorm(0, 0.01); beta2 ~ dnorm(0, 0.01)
    beta3 ~ dnorm(0, 0.01); beta4 ~ dnorm(0, 0.01)
    tau <- pow(sigma, -2)
    sigma ~ dunif(0, 10)
}
```

run by

```
library(BRugs)
inits <- list(list(beta0 = 0, beta1 = 0, beta2 = 0, beta3 = 0, beta4 = 0, sigma = 1))
vars <- c("beta1", "beta2", "beta3", "beta4", "sigma")
p1.sims <- with(puffin,
    BRugsFit("puffin.bug",
             data = list("Nest", "Grass", "Soil", "Angle", "Distance"),
```

```
                inits = inits, parametersToSave = vars, numChains = 1,
                nBurnin = 5000, nIter = 25000, nThin = 50,
                coda = TRUE, DIC = FALSE))
summary(p1.sims)
densityplot(p1.sims)
```

(a JAGS version is given on the problem sheets).

A Poisson regression model would appear to be more appropriate: the classical version would be

```
pfit <- glm(Nest ~ Grass + Soil + Angle + Distance, poisson, data = puffin)
summary(pfit)
```

but note that exhibits considerable over-dispersion and the dataset does have 13/38 zero counts (and none of one or two). If for the moment we ignore that, we could use

```
Bpfit <- MCMCpoisson(Nest ~ Grass + Soil + Angle + Distance,
                     data = puffin, burnin = 100, mcmc = 5000, thin = 5)
summary(Bpfit)
```

but we really need a better model (for example a negative binomial regression[23], perhaps with zero-inflation).

Hierarchical models

MCMC is widely used in hierarchical Bayesian models. Here is a simple example considered by Gelman *et al.* (2004, §§5.6, 6.8, 17.4).

The US Educational Testing Service investigated the effect of coaching for SAT-V tests in 8 schools. The tests are supposed to be insensitive to coaching, but these 8 schools claimed to have an effective coaching program. The SAT-V scores have a range of 200–800. This is a meta-analysis: for each school we have estimates of the mean effect of coaching and of the standard deviation of the effect estimate *via* a within-school analysis of covariance adjusting for some explanatory factors (scores on tests prior to coaching).

Gelman *et al.* (2004, Appendix C) provide R and WinBUGS code for several analyses based on Gibbs sampling. The model can be defined by the BUGS model file[24] `schools.bug`:

```
model {
    for (j in 1:J) {
        y[j] ~ dnorm(theta[j], tau.y[j])
        theta[j] ~ dnorm(mu.theta, tau.theta)
        tau.y[j] <- pow(sigma.y[j], -2)
    }
    mu.theta ~ dnorm(0, 1.0E-6)
    tau.theta <- pow(sigma.theta, -2)
    sigma.theta ~ dunif(0, 1000)
}
```

[23] A function to fit that is available in R package `bayesm`, and `JAGS` code is given by Jackman (2009, pp. 278–80).

[24] JAGS can use `length(y)` instead of `J`.

So there are `J = 8` schools, and each has a mean θ_i and precision τ_i, with the per-schools means being modelled as draws from a normal population. Thus the parameters are the 8 θ_i and the two hyperparameters for the population distribution of means. An alternative model we can consider is a t_ν distribution for the population of means, with a known or unknown ν.

Simulation-based inference makes it easy to draw inferences about non-linear functions of the parameters, for example of the largest effect $\max_i \theta_i$.

We can fit this model in OpenBUGS using different random starting values for each run by

```
library(BRugs)
J <- 8
y <- c(28, 8, -3, 7, -1, 1, 18, 12)
sigma.y <- c(15, 10, 16, 11, 9, 11, 10, 18)
data <- list("J", "y", "sigma.y")
inits <- function()
    list(theta = rnorm(J,0,100), mu.theta = rnorm(1,0,100),
         sigma.theta = runif(1,0,100))
parameters <- c("theta", "mu.theta", "sigma.theta")
schools.sim <- BRugsFit("schools.bug", data, inits, parameters,
                        numChains = 3,
                        nBurnin = 10000, nIter = 10000, nThin = 10,
                        DIC = FALSE, coda = TRUE)
```

and in JAGS by

```
library(rjags)
schools.jags <- jags.model("schools.bug", inits = inits,
                           n.chains = 3, n.adapt = 10000)
schools.sims2 <- coda.samples(schools.jags, parameters,
                              n.iter = 10000, thin = 10)
```

In both cases we can use `coda` to analyse the samples (and will do in the practical examples).

Mixed-effect linear models

Straightforward linear models of the form

$$Y = X\beta + \epsilon, \qquad \epsilon \sim N(0, \sigma^2 I)$$

have $p+1$ parameters $\theta = (\beta, \sigma^2)$. A Bayesian analysis needs a prior for θ, and for convenient priors the posterior can be found explicitly. However, if we allow non-IID errors (so $\epsilon \sim N(0, \kappa\Sigma(\psi))$) then simulation-based methods become much more convenient, and perhaps essential.

One common way for such structured variance matrices to arise is what is called in the classical literature *mixed effects* models. Suppose that

$$Y = X\beta + Z\gamma + \epsilon, \qquad \epsilon \sim N(0, \sigma^2 I)$$

where γ is regarded as random vector. In a Bayesian setting β is already regarded as a random vector so this is no real change, but impact comes from thinking of this hierarchically. In the simplest case, suppose we have two levels of units, say observations on classes in schools or

repeated measurements on individuals. Rather than the exchangeability that the IID assumption entails, we now have a multi-level invariance amongst groups of observations. With two levels of units, the Bayesian model has three groups of random variables

- data points Y_{ij}, observed at the lowest level,
- random effects η_i on level-one units, unobserved, and
- parameters in the distribution.

Note though that it is just a linear model for the observed data with a parametrized variance matrix of correlated errors.

This is fertile ground for use of a Gibbs sampling scheme to simulate from the posterior distribution given the observed data, so many schemes have been proposed. Here are the basic ideas.

(a) a grouped Gibbs sampler, in which all the variables in one of the three groups are updated at once. Generally the conditional distributions of the groups given the rest are simple to simulate from, although the variance parameters may need a Metropolis step. How well this works depends strongly on how well the hierarchical model mimics the real dependence structure.

(b) an 'all-at-once' Gibbs sampler. This flattens the hierarchical model to two levels, the linear coefficients and the distributional parameters, and alternates between them. Effectively it fits a weighted regression for known variance parameters, then simulates variance parameters conditional on the residuals from the weighted regression. It is in general easy to implement and quick to converge, but the flattened model can be much larger.

(c) a single-variable Gibbs sampler, updating one variable at a time. Again this is usually simple to implement, and simulating the individual regression parameters is fast. The problem is that the latter can be highly correlated and so the Gibbs sampler moves slowly: this can often be overcome by a linear transformation of the regression parameters, one which can be approximated by a pilot run.

(d) parameter expansion. All of these schemes can be slow to converge when estimated hierarchical variance parameters are near zero, since this will ensure that the corresponding random effects are estimated as rather similar and then at the next step the variance parameters is estimated as small. We can circumvent this by adding further parameters, e.g. a multiplicative effect on all the random effects in a group. For the SAT-V data the model becomes

$$\begin{aligned} Y_j &\sim N(\mu + \alpha\gamma_j, \sigma_j^2) \\ \gamma_j &\sim N(0, \sigma_\gamma^2) \\ p(\mu, \alpha, \sigma_\gamma) &\propto 1 \end{aligned}$$

(or some other prior on the parameters). The analysis discussed above was for the model with $\alpha \equiv 1$. Note that σ_γ^2 is no longer the random effects variance, rather $\alpha^2\sigma_\gamma^2$. (Some care is needed as aspects of the posterior here are improper—it would be better to use a proper prior for α.)

For more details on simulation-based Bayesian approaches to regression-like problems see Gelman & Hill (2007) and Gelman *et al.* (2004). Gamerman & Lopes (2006, §6.5) have complementary material.

5 Model criticism

Fitting a model is just part of statistical methodology: we also need to consider what Gelman *et al.* (2004, Chapter 6) call 'model checking and improvement', which covers topics such as

- *Sensitivity analysis*: to what extent do the conclusions depend on the prior or on particular aspects of the data (including single observations)?
- Do the inferences make sense? When validated against external information not included in the model formulation process. For example, we may be able to compare predictions made by the model on a validation set with the known results, or to compare the signs of coefficients with those predicted by subject-matter theory.
- Is the model consistent with the data?
- Does the model need to be expanded? Does it capture enough of the uncertainty?
- If we have more than one plausible model, how do we compare them?

Not much of this is specifically Bayesian (although sensitivity to the prior is), but the methods are somewhat different because of the use of simulation methods and because we are not using *plug-in*[25] methods but rather trying to take most of the uncertainty into account.

One overall comment: almost all Bayesian model criticism assumes that the model is true, whereas we have usable theory for what happens with maximum likelihood estimation when the true underlying probability distribution is not within the model space.

Sensitivity to the prior can often be assessed quite easily *via* importance sampling or SIR: changing the prior specification would just reweight the MCMC posterior samples.

Posterior predictive distributions

We have a posterior distribution $p(\theta \mid \mathcal{T})$ over the parameters in the model, and $\mathcal{T}$ denotes the given data (the *training set* in the terminology of pattern recognition).

We can use this to give a probability distribution over future observations $\tilde{y}$ by

$$p(\tilde{y} \mid \mathcal{T}) = \int p(\tilde{y} \mid \theta) p(\theta \mid \mathcal{T}) d\theta \tag{6}$$

which is often known as the *posterior predictive distribution*. Another way to consider this is to think of generating a new set of data of the same size (and at the same set of covariates) as the given data, and compare the dataset generated under the model with the original: systematic departures indicate a lack of fit of the model. And because we can generate many pseudo-replicate datasets, we can get some handle on whether departures are systematic or might have occurred by chance.

Almost all of classical statistical methods is concerned with comparing data with fitted models, and very much the same methods can be used here—the only difference is that the fitted model

[25] 'Plug-in' methods make a point estimate of parameters and then act as if the estimated values are the true values.

used is the posterior predictive distribution. For example, we can compare possible outliers with extremes generated from (6), and measure extremeness by a p value under the predictive distribution (rather than the plug-in distribution). *Residuals* are defined by comparing the observed value to the prediction under (6), that is of the form

$$y_i - E[Y_i \,|\, x_i, \mathcal{T}]$$

rather than $y_i - E[Y_i \,|\, x_i, \widehat{\theta}]$.

In the case of hierarchical models with a hyperprior, we might want to consider just which posterior distribution to choose – for example if we have data from experiments on a group of animals, we can consider either (pseudo-)replicating the experiment on the same animals or on another sample of animals chosen from the superpopulation of animals. Normally most such choices are appropriate for some questions about model fit: but different choices for different questions.

Model comparison

Comparison of models in a Bayesian setting is a specialized topic, mainly because it is pretty close to non-Bayesian. To quote Geisser (1993, p. 91)

> The question then is what we do once we select the best model? Presumably it affords the "best" single description amongst those entertained. Should we now use it for prediction? If we do, we know it is not optimal under any loss function except one that reflects a principle of parsimony that states one should only use one of the models for prediction.

That *can* happen for reasons of speed, but is rare.

What theory suggests is that we use so-called *Bayesian model averaging*. Assume m (finite) models, exactly one of which is true. In the Bayesian formulation, models are compared via $P\{M \,|\, \mathcal{T}\}$, the posterior probability assigned to model M:

$$P\{M \,|\, \mathcal{T}\} \propto p(\mathcal{T} \,|\, M) p_M,$$

where the so-called *marginal likelihood* is

$$p(\mathcal{T} \,|\, M) = \int p(\mathcal{T} \,|\, M, \theta) p(\theta) \,\mathrm{d}\theta$$

(using the posterior predictive distribution at the given dataset under model M), so the ratio in comparing models M_1 and M_2 is proportional to $p(\mathcal{T} \,|\, M_2)/p(\mathcal{T} \,|\, M_1)$, known as the *Bayes factor*.

It is often assumed (and often implicitly) that models have equal prior probabilities p_M.

However, a formal Bayesian approach then averages predictions from models, weighting by $P\{M \,|\, \mathcal{T}\}$, unless a very peculiar loss function is in use.

Suppose we just use the Bayes factor as a guide. The difficulty is in evaluating $p(\mathcal{T} \,|\, M)$ (see for example Carlin & Louis (2009, §4.4 & §4.5)). Asymptotics are not very useful for Bayesian

methods, as the prior on θ is often very important in providing smoothing, yet asymptotically negligible. But using some (rather dubious) asymptotics leads to the criterion variously known as BIC or SBC ('Schwarz's Bayesian Criterion').

$P\{M \mid \mathcal{T}\}$ is just another marginal distribution, but one that is hard to compute accurately by MCMC methods (naïve computation does depend on having samples that explore the whole parameter space, but for some models, including linear regression, other tricks can be used—a comprehensive discussion can be found in Ntzoufras (2009, §11.3)). It also suffers from conceptual problems: it assumes the model is true, and may depend on aspects of model fit which are not interesting to the questions in hand.

Gelman *et al.* (2004, pp. 185–6) give two examples, entitled

- *An example in which Bayes factors are helpful.*
  (A simple genetic counselling example with objective priors and no nuisance parameters.)
- *An example in which Bayes factors are a distraction.*
  The latter is a variation on the SAT example, and highlights two problems—we need a proper prior to compute the marginal likelihood (and comparisons can depend heavily on the precise choice of proper but diffuse prior) and there are dimensionality effects, so the weights given to different aspects of the fit for 8 schools is rather different than if there had been 80.

Another issue is *Bartlett's paradox*, which points out that the Bayes factor is unduly sensitive to the degree of vagueness assumed for the priors (and these need to be proper).

Consider again the data on puffin nesting. We need a proper prior to compute marginal likelihoods, so refit by

```
library(MCMCpack)
Bfit <- MCMCregress(Nest ~ Grass + Soil + Angle + Distance,
                    data = puffin,  B0 = 0.1,
                    burnin = 100, mcmc = 5000, thin = 5,
                    marginal.likelihood = "Chib95")
attr(Bfit, "logmarglike")
```

Now suppose we repeat this for other models

```
-118.88: Grass + Soil + Angle + Distance
-113.85: Soil + Angle + Distance
-120.12: Grass + Angle + Distance
-117.23: Grass + Soil + Distance
-126.20: Grass + Soil + Angle
-145.23: Grass + Soil
-122.80: Grass + Angle
-130.22: Grass + Distance
-123.29: Soil + Angle
-112.12: Soil + Distance
-116.91: Angle + Distance
-142.30: Grass
-141.67: Soil
-119.46: Angle
-127.29: Distance
```

For a uniform prior on models the posterior is primarily weighted to the models with formulae `Nest ~ Soil + Distance` (84%) and `Nest ~ Soil + Angle + Distance` (15%)—but the choice of prior does matter.

An alternative approach is to include in the model an indicator variable for whether or not the term should be included. Thus the model becomes

$$y = \beta_0 + \sum_{j=1}^{p} z_j \beta_j x_j$$

where $z_1 \dots z_p$ are 0/1-valued indicators. These can be given a prior distribution (e.g. independent fair coin tosses) and their (joint) posterior will give information on which model is favoured, and the posterior predictive distribution averages over all possible subset models. We can run this in BUGS with model file `puffin2.bug`:

```
model{
    for(i in 1:38) {
         Nest[i] ~ dnorm(mu[i], tau)
         mu[i] <- beta0 + beta1*Grass[i] + beta2*Soil[i] +
                  beta3*Angle[i] + beta4*Distance[i]
    }
    beta0 ~ dnorm(0, 0.01)
    b1 ~ dnorm(0, 0.01); b2 ~ dnorm(0, 0.01)
    b3 ~ dnorm(0, 0.01); b4 ~ dnorm(0, 0.01)
    z1 ~ dbern(0.5); z2 ~ dbern(0.5); z3 ~ dbern(0.5); z4 ~ dbern(0.5)
    beta1 <- z1*b1; beta2 <- z2*b2; beta3 <- z3*b3; beta4 <- z4*b4
    tau <- pow(sigma, -2)
    sigma ~ dunif(0, 10)
}
```

run ten times (for a couple of minutes) by

```
inits <- function() list(beta0 = 0, b1 = 0, b2 = 0, b3 = 0, b4 = 0, sigma = 1,
                         z1 = 1, z2 = 1, z3 = 1, z4 = 1)
vars <- c("beta1", "beta2", "beta3", "beta4", "sigma", "z1", "z2", "z3", "z4")
p2.sims <- with(puffin,
    BRugsFit("puffin2.bug",
             data = list("Nest", "Grass", "Soil", "Angle", "Distance"),
             inits = inits, parametersToSave = vars, numChains = 10,
             nBurnin = 1000, nIter = 5e4, nThin = 50,
             coda = TRUE, DIC = FALSE, BRugsVerbose = FALSE))
summary(p2.sims)
plot(p2.sims[[1]]) # look at one run
pis <- do.call(rbind, p2.sims[, 6:9])
Fr <- as.data.frame(xtabs(~ ., pis))
Fr[sort.list(-Fr$Freq), ]
```

with result (from one particular set of 10,000 observations)

```
   z1 z2 z3 z4 Freq
11  0  1  0  1 6729
5   0  0  1  0 1663
13  0  0  1  1 1154
15  0  1  1  1  345
14  1  0  1  1   51
9   0  0  0  1   39
12  1  1  0  1    9
7   0  1  1  0    6
6   1  0  1  0    4
```

(Ntzoufras, 2009, Chapter 11) gives a reasonably comprehensive account of Bayesian model comparison. The other accounts are sketchy: Albert (2009) covers selection in regression, and there are some comments in Carlin & Louis (2009).

References

Aarts, E. and Korst, J. (1989) *Simulated Annealing and Boltzmann Machines*. John Wiley and Sons.

Albert, J. (2009) *Bayesian Computation with R*. Second Edition. New York: Springer.

Albert, J. and Chib, S. (1993) Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association* **88**, 669–679.

Asmussen, S. and Glynn, P. W. (2007) *Stochastic Simulation. Algorithms and Analysis*. New York: Springer.

Berthelsen, K. K. and Møller, J. (2003) Likelihood and non-parametric Bayesian MCMC inference for spatial point processes based on perfect simulation and path sampling. *Scandinavian Journal of Statistics* **30**, 549–564.

Carlin, B. P. and Louis, T. A. (2009) *Bayesian Methods of Data Analysis*. CRC Press.

Casella, G., Lavine, M. and Robert, C. (2001) Explaining the perfect sampler. *American Statstician* **55**, 299–305.

Chen, M.-H., Shao, Q.-M. and Ibrahim, J. G. (2000) *Monte Carlo Methods in Bayesian Computation*. New York: Springer.

Chib, S. (1998) Estimation and comparison of multiple change-point models. *J. Econometrics* **86**, 221–241.

Chib, S. and Greenberg, E. (1995) Understanding the Metropolis–Hastings algorithms. *American Statistician* **49**, 327–335.

Collett, D. (1991) *Modelling Binary Data*. London: Chapman & Hall.

Congdon, P. (2003) *Applied Bayesian Modelling*. Chichester: Wiley.

Congdon, P. (2005) *Bayesian Models for Categorical Data*. Chichester: Wiley.

Congdon, P. (2006) *Bayesian Statistical Modelling*. Second Edition. Chichester: Wiley.

Congdon, P. (2009) *Applied Bayesian Hierarchical Models*. London: Chapman & Hall.

Dagpunar, J. (2007) *Simulation and Monte Carlo. With Applications in Finance and MCMC*. Chichester: Wiley.

Evans, M. and Swartz, T. (2000) *Approximating Integrals via Monte Carlo and Deterministic Methods*. Oxford: Oxford University Press.

Gamerman, D. and Lopes, H. F. (2006) *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Second Edition. London: Chapman & Hall/CRC Press.

Geisser, S. (1993) *Predictive Inference: An Introduction*. New York: Chapman & Hall.

Gelman, A., Carlin, J. B., Stern, H. S. and Rubin, D. B. (2004) *Bayesian Data Analysis*. Second Edition. Chapman & Hall/CRC Press.

Gelman, A. and Hill, J. (2007) *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.

Gelman, A. and Rubin, D. B. (1992) Inference from iterative simulation using multiple sequences (with discussion). *Statistical Science* **7**, 457–511.

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

Geweke, J. (1992) Evaluating the accuracy of sampling-based approaches to calculating posterior moments. In *Bayesian Statistics 4*, eds J. M. Bernado, J. O. Berger, A. P. Dawid and A. F. M. Smith, pp. 169–193. Oxford: Clarendon Press.

Geyer, C. (1999) Likelihood inference for spatial point processes. In *Stochastic Geometry. Likelihood and Computation*, eds O. E. Barndorff-Nielsen, W. S. Kendall and M. N. M. van Lieshout, Chapter 3, pp. 79–140. London: Chapman & Hall/CRC.

Gilks, W. R., Richardson, S. and Spiegelhalter, D. J. (1996) *Markov Chain Monte Carlo in Practice*. London: Chapman & Hall.

Green, P. J. (1995) Reversible junp Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika* **82**, 711–732.

Grenander, U. and Miller, M. (1994) Representations of knowledge in complex systems (with discussion). *Journal of the Royal Statistical Society series B* **56**, 549–603.

Hastings, W. K. (1970) Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **57**, 97–109.

Heidelberger, P. and Welch, P. D. (1983) Simulation run length control in the presence of an initial transient. *Operations Research* **31**, 1109–1144.

Holmes, C. C. and Held, L. (2006) Bayesian auxiliary models for binary and multinomial regression. *Bayesian Analysis* **1**, 145–168.

Jackman, S. (2009) *Bayesian Analysis for the Social Sciences*. New York: Wiley.

Jarrett, R. G. (1979) A note on the interval between coal-mining disasters. *Biometrika* **66**, 191–3.

Jerrum, M. (1995) A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Structures and Algorithms* **7**, 157–165.

Kendall, W. S. (2005) Notes on perfect simulation. In *Markov Chain Monte Carlo. Innovations and Applications*, eds W. S. Kendall, F. Liang and J.-S. Wang, pp. 93–146. Singapore: World Scientific.

Kirkpatrick, S., Gelatt, Jr, C. D. and Vecchi, M. P. (1983) Optimization by simulated annealing. *Science* **220**, 671–680.

Kushner, H. J. and Lin, G. G. (2003) *Stochastic Approximation and Recursive Algorithms and Applications*. Second Edition. New York: Springer-Verlag.

Lancaster, T. (2004) *An Introduction to Modern Bayesian Econometrics*. Oxford: Blackwell.

Lauritzen, S. and Spiegelhalter, D. J. (1988) Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society series B* **50**, 157–224.

Liu, J. S. (2001) *Monte Carlo Strategies in Scientific Computing*. New York: Springer.

Lunn, D., Spiegelhalter, D., Thomas, A. and Best, N. (2009) The BUGS project: Evolution, critique and future directions (with discussion). *Statistics in Medicine* **28**, 3049–3082.

Lunn, D. J., Thomas, A., Best, N. and Spiegelhalter, D. (2000) Winbugs – a Bayesian modelling framework: concepts, structure, and extensibility. *Statistics and Computing* **10**, 325–337.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. (1953) Equations of state calculations by fast computing machines. *Journal of Chemical Physics* **21**, 1087–1091.

Neal, R. M. (2003) Slice sampling. *Annals of Statistics* **31**, 705–767.

Ntzoufras, I. (2009) *Bayesian Modelling Using WinBUGS*. Hoboken: Wiley.

Pincus, M. (1970) A Monte-Carlo method for the approximate solution of certain types of constrained optimization problems. *Operations Research* **18**, 1225–1228.

Propp, J. and Wilson, D. (1996) Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms* **9**, 223–252.

Raftery, A. E. and Lewis, S. M. (1992) One long run with diagnostics: Implementation strategies for Markov chain Monte Carlo. *Statistical Science* **7**, 493–497.

Ripley, B. D. (1979) Algorithm AS137. Simulating spatial patterns: dependent samples from a multivariate density. *Applied Statistics* **28**, 109–112.

Ripley, B. D. (1987) *Stochastic Simulation*. New York: John Wiley and Sons.

Ripley, B. D. (1988) *Statistical Inference for Spatial Processes*. Cambridge: Cambridge University Press.

Ripley, B. D. (2005) How computing has changed statistics. In *Celebrating Statistics: Papers in Honour of Sir David Cox on His 80th Birthday*, eds A. C. Davison, Y. Dodge and N. Wermuth, pp. 197–211. Oxford University Press.

Ripley, B. D. and Kirkland, M. D. (1990) Iterative simulation methods. *Journal of Computational and Applied Mathematics* **31**, 165–172.

Robert, C. P. and Casella, G. (2004) *Monte Carlo Statistical Methods*. Second Edition. New York: Springer.

Robert, C. P. and Casella, G. (2010) *Introducing Monte Carlo Methods with R*. New York: Springer.

Roberts, G. O. and Rosenthal, J. S. (1998) Markov-chain Monte Carlo: Some practical implications of theoretical results. *Canadian Journal of Statistics* **26**, 5–31.

Roberts, G. O. and Tweedie, R. L. (2005) *Understanding MCMC*. New York: Springer.

Smith, A. F. M. and Gelfand, A. E. (1992) Bayesian statistics without tears: a sampling–resampling perspective. *American Statistician* **46**, 84–88.

Snijders, T. A. B. (2001) The statistical evaluation of social network dynamics. In *Sociological Methodology – 2001*, eds M. Sobel and M. Becker, pp. 361–395. Boston and London: Basil Blackwell.

Snijders, T. A. B. (2006) Statistical methods for network dynamics. In *Proceedings of the XLIII Scientifc Meeting, Italian Statistical Society*, pp. 281–296. Padova: CLEUP.

Spiegelhalter, D., Best, N., Thomas, A. and Lunn, D. (2010) *Bayesian Analysis using the BUGS language*. Chapman & Hall.

Tanner, M. A. (1996) *Tools for Statistical Inference*. Third Edition. Springer-Verlag.

Tanner, M. A. and Wong, W. H. (1987) The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association* **82**, 528–540.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth Edition. New York: Springer-Verlag.

Yu, B. and Mykland, P. (1998) Loooking at Markov samplers through cusum path plots: A simple diagnostic idea. *Statistics and Computing* **8**, 275–286.

A Markov processes and chains

This appendix provides a brief recap of some parts of stochastic process theory.

Suppose we have a stochastic process $\{X_t\}$ taking values in a discrete or continuous space $\mathcal{X}$ with $t \in T$, either discrete time or continuous time. Then the process is said to be *Markov* if the past and future are independent given the present state X_t, or equivalently if the future $\{X_s, s > t\}$ depends on the (weak) past $\{X_s, s \leq t\}$ only through X_t.

A *Markov chain* is less unambiguously defined, but usually means a Markov process with a discrete state space $\mathcal{X}$, and often with discrete time too. See `http://en.wikipedia.org/wiki/Markov_chain`. Of course, books about MCMC tend to define it as a discrete-time Markov process.

For those with the necessary probability background, introductory accounts of Markov processes for MCMC can be found in Gamerman & Lopes (2006, Chapter 4) and Robert & Casella (2004, Chapter 6), and a summary in Jackman (2009, Chapter 4).

A stochastic process is *stationary* if the joint distribution of a finite set of $X_{\tau+t_i}$ does not depend on τ: a stationary process is necessarily defined for all times in $(-\infty, \infty)$ or the integers.

Continuous-time Markov chains proceed by waiting an exponentially-distributed time (with rate depending on X_t) and then jumping to another state. These jumps can pile up so an infinite number are taken in a finite time, but we will have no use of such processes. For the normal case, if we look at the process only immediately it jumps we have a discrete-time process (the *jump chain*), and we simulate a Markov process via the exponentially-distributed waiting times and the jump chain.

Apart from the spatial birth-and-death process, all the Markov processes we need to consider are in discrete time (but usually with a continuous state space).

Detailed balance

Data augmentation and spatial birth-and-death processes provide 'mechanistic' approaches to developing an MCMC algorithm, but in general MCMC algorithms can be unrelated to any hypothesized stochastic generative mechanism. Especially in such cases, we need to be able to show formally that we do indeed have a Markov process with the desired stationary distribution, and that the stationary distribution is the limiting distribution.

A key concept is *detailed balance*, which is connected to *reversibility* of the Markov process. Reversibility just means that the joint distribution of the process at a series of times is unchanged if the direction of time is reversed—clearly this only makes sense for a stationary process as for any other Markov process the convergence towards equilibrium reveals the direction of time.

For a discrete-time discrete-state-space Markov process reversibility entails

$$P(X_t = i, X_{t+1} = j) = P(X_{t+1} = i, X_t = j) = P(X_t = j, X_{t+1} = i)$$

so if (π_i) is the stationary distribution,

$$\pi_i P_{ij} = \pi_j P_{ji} \tag{7}$$

for transition matrix P_{ij}. This equation is known as *detailed balance*.

If we know there is a unique stationary distribution, and we can show detailed balance for our distribution π, we have shown that it is the unique stationary distribution. If we also know[26] that the Markov process converges to its stationary distribution, we have a valid MCMC sampling scheme.

Similar considerations apply to continuous-state-space Markov processes, e.g. detailed balance can apply to the density of the stationary distribution.

[26] e.g. by showing it is aperiodic and irreducible, and for continuous state-spaces Harris recurrent.

B The BUGS language

The BUGS language is an R-like language for specifying a class of Bayesian models. It was originally developed for classic BUGS and is used with some variations by its descendants WinBUGS, OpenBUGS and JAGS. WinBUGS and OpenBUGS provide another way to specify these models *via* a point-and-click interface called 'DoodleBUGS'.

This appendix gives some introductory notes on the BUGS language, enough to understand the examples for this module. For more serious use, consult the manual of the variant of BUGS you use, `http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/manual14.pdf`, `http://www.openbugs.info/Manuals/Manual.html`[27] or `http://sourceforge.net/projects/mcmc-jags/files/Manuals/2.0`.

BUGS models are graphical models containing several types of nodes, specified in a `model` statement in the script file.

The *stochastic* nodes are random variables whose distribution is specified *via* a formula containing `~`: the left-hand side of the formula is a variable name and the right-hand side is a call to a BUGS function which specifies the prior distribution, often with parameters which are values of other nodes. Thus in the schools example

```
model {
    for (j in 1:J) {
        y[j] ~ dnorm(theta[j], tau.y[j])
        theta[j] ~ dnorm(mu.theta, tau.theta)
        tau.y[j] <- pow(sigma.y[j], -2)
    }
    mu.theta ~ dnorm(0, 1.0E-6)
    tau.theta <- pow(sigma.theta, -2)
    sigma.theta ~ dunif(0, 1000)
}
```

we (and the BUGS model compilers) can identify the stochastic nodes *via* the lines given as formulae. We often (as here) want parameter values which are computed as functions of the values at other nodes: these are called *deterministic* nodes[28] and are specified by R-like assignment lines. Then there are (obviously) *constants*, and finally some values that are not specified: JAGS figures out for itself that the latter are data values, but BUGS needs to be told (in the interface we are using *via* a named R list argument, originally *via* a data file). Each deterministic or stochastic node depends on others (called its *parents*), so the nodes form a directed graph. In JAGS this graph must be acyclic—in BUGS under some circumstances[29] cycles are allowed.

The various versions of BUGS have restrictions on what distributions can be used where: these are not shared by JAGS.

Deterministic nodes are a notational device forced by some of the restrictions of the original BUGS language—in JAGS they can be avoided, so we can write

[27] which also ships with `BRugs` as HTML linked from file `BRugs\OpenBUGS\Manuals\Manual.html`

[28] or *logical* nodes, especially in the BUGS manuals.

[29] e.g. for defining autoregressive and ordered priors

```
model {
    for (j in 1:length(y)) {
        y[j] ~ dnorm(theta[j], sigma.y[j] ^ -2)
        theta[j] ~ dnorm(mu.theta, sigma.theta ^ -2)
    }
    mu.theta ~ dnorm(0, 1.0E-6)
    sigma.theta ~ dunif(0, 1000)
}
```

Deterministic nodes are not observables, so you cannot supply data values for them. Note that as in the coal-mining example

```
log(mu[year]) <- b[1] + step(year - changeyear) * b[2]
```

functions can appear on the left-hand side of declarations of deterministic nodes. However, this is restricted to known *link* functions `log`, `logit`, `probit` and `cloglog` and the corresponding inverse functions[30] can be used directly, e.g. for JAGS we could use

```
model {
  for (year in 1:N) {
       D[year] ~ dpois(exp(b[1] + step(year - changeyear) * b[2]))
  }
  for (j in 1:2) { b[j] ~ dnorm(0.0, 1.0E-6) }
  changeyear ~ dunif(1,N)
}
```

As we see in these examples, one-dimensional arrays can be specified, and 2 or more dimensions too (but like R as a convenient way to specify a univariate index). There are some multivariate distributions (e.g. multinomial, Dirichlet, multivariate normal and t, Wishart), but otherwise arrays are handled by R-like `for` loops (and the braces are essential, unlike R). In BUGS the indices of arrays must be simple expressions, but the values of deterministic nodes defined to compute more complex expressions can be used. As in R missing indices indicate all elements, and ranges such as `m:n` are allowed.

Parentheses, the arithmetic operators `+ - * /` plus a fairly short list of built-in functions are allowed for deterministic nodes. The list includes the link functions and their inverses, `abs`, `sqrt`, `pow`, `cos`, `sin`, `loggam`, `logfact`, `phi` (Φ, the standard normal CDF, not the density), `round`, `trunc`, `step` (a Heaviside step function at 0), `max`, `min`, `mean`, `sd`, `sum`, `prod`, `rank`[31] and `sort`. There are also some matrix functions such as `t`, `inverse` and `logdet`, plus in JAGS matrix multiplication (`%*%`).

Note (as in the first version of the schools example) there is no exponentiation operator in BUGS, unlike JAGS, so the `pow` function must be used. There are some facilities for extending the language by adding distributions or functions *via* compiled code: in addition OpenBUGS has a `dgeneric` distribution that allows simulation by inversion.

The original intention was that the `model` block be purely declarative, but in BUGS it has been extended to allow data transformations (which can equally well be done in an R interface). JAGS does not allow these, but allows a separate `data` block.

[30] `exp`, `ilogit`, `phi` and `icloglog`

[31] but with different arguments in OpenBUGS and JAGS

Censoring and truncation

All the examples shown so far are valid R code, and the functions `write.model` in packages `BRugs` and `R2WinBugs` exploit this to write a BUGS model file from an R expression.

However, there are exceptions used to indicate censoring and truncation, in ways that differ between the BUGS dialects. In OpenBUGS as used by `BRugs` we could write

```
y ~ dnorm(mu, tau) C(lo, hi)
```

where `lo` and `hi` are nodes or constants (and if one it omitted it indicates a half-infinite interval). This represents *censoring* (values outside the restriction are known about, but the exact value is not observed): the function `T()` has the same semantics but a different interpretation, representing *truncation* (values outside the restriction will never be seen).

In WinBUGS `C()` is replaced by `I()` with no counterpart for truncation.

JAGS has `T()` but not `C()`: censoring is handled *via* its `dinterval` function with a separate indicator variable: see the manual.

Lexical restrictions

The following restrictions apply in classic BUGS: it is often unclear what is allowed in later dialects.

Node names must start with a letter, contain letters, numbers or periods and not end in a period. They are case sensitive, and in Classic BUGS limited to 32 chars. (Although unstated, it seems that 'letter' means the English letters `A-Z` and `a-z`.) OpenBUGS also disallows two successive periods. JAGS allows alphanumeric characters plus a period, starting with a letter.

Numeric constants are specified either as decimals, possibly including a decimal point, or exponential notation such as `1.0E-6` with the exponent marked by `E` or, perhaps, e. (Classic BUGS requires a decimal point in exponential notation.)

Examples of BUGS code

Classic BUGS came with two volumes of examples, available at the same site as the manual. This became three volumes for WinBUGS and OpenBUGS[32], and there are versions of the Classic BUGS examples adapted for JAGS on the latter's site.

Gelman & Hill (2007, p. 11) comment (their capitalization):

> *The two volumes of online examples for Bugs give some indication of the possibilities — it fact it is common practice to write a Bugs script by starting with an example with similar features and then altering it step-by-step to fit the particular problem at hand.*

Other extensive sources of examples of BUGS code are several recent textbooks such as Congdon (2006, 2003, 2005, 2009), Lancaster (2004), Ntzoufras (2009) and Jackman (2009).

[32] installed as part of `BRugs` in directory `BRugs\OpenBUGS\Examples`

Model compilation

Given the model specification and the data, the first task is to identify all the unobserved stochastic nodes (note that some of the 'data' might be missing values) and group them into blocks for a Gibbs sampler. Then the conditional distribution of each block given the remaining nodes is a distribution with parameters which are deterministic functions of other nodes. This phase implicitly sets an order in which the blocks will be sampled.

The next step (which JAGS refers to as part of *initialization*) is to choose an algorithm to sample the conditional distribution of each block. In `rjags` we can find out what sampling algorithms were chosen by

```
> unlist(list.samplers(schools.jags))
ConjugateNormal ConjugateNormal ConjugateNormal ConjugateNormal
     "theta[8]"      "theta[7]"      "theta[6]"      "theta[5]"
ConjugateNormal ConjugateNormal ConjugateNormal ConjugateNormal
     "theta[4]"      "theta[3]"      "theta[2]"      "theta[1]"
     RealSlicer ConjugateNormal
  "sigma.theta"      "mu.theta"
```

Note that where conjugate priors are used the model compiler is able to recognize the analytical form of the posterior and so use a standard method to sample from a known distribution. Slice sampling is used as a general-purpose sampler from a univariate continuous distribution: here there is an analytical form for the posterior but it is not currently in JAGS' repertoire.

Either slice or Metropolis (with random-walk proposals) samplers may be selected to sample from a single node: both are implemented adaptively, that is tune their parameters for increased efficiency during the burn-in phase of sampling (which is then no longer a Markov Chain, since many or all past values are used in choosing the sampling algorithm and hence the next move).

WinBUGS allows some control over the choice of samplers, for example to disable some of the possibilities. More detailed control is available in package `BRugs`: see the help on `modelEnable` and `modelSetAP`. There is some discussion of the OpenBUGS sampling algorithms in file `BRugs/OpenBUGS/Developer/SamplingAlgorithms.html` in package `BRugs`, but lack of documentation makes these functions 'for experts only'.

Model compilation in BUGS automatically adds a deterministic node called `deviance`, which can be monitored. The name is somewhat misleading, for it is minus twice the log-likelihood of the observed stochastic nodes (the data) at the current values of their parent nodes (essentially, the current parameter values).

Random number generators

Neither `BRugs` nor `rjags` use the R random number generators, rather those in OpenBUGS and JAGS respectively which are by default initialized from the wall clock. In `rjags` you can set both the generator to be used (it contains four based on those used by R: see `?RNGkind`) and the seed for each chain *via* the `inits` argument of `jags.model`, e.g.

```
inits2 <- lapply(1:5, function(i) c(inits(),
                                    .RNG.name = "base::Mersenne-Twister",
```

```
                                .RNG.seed = i))
```

selects the 'Mersenne Twister' random number generator for each chain with seeds `1`, `2`, ..., `5`. Unless you set `.RNG.name`, JAGS uses a different type of generator for each chain and hence runs out of types after four chains.

WinBUGS and OpenBUGS each have an item on their Model menu to set the random number seed(s). `BRugs` has functions `modelGetSeed` and `modelSetSeed`: however, the details needed to make this useful are lacking in the documentation.

C Using CODA

The `coda` package in R is based on a set of S-PLUS functions, designed to take the output from the standalone BUGS and (later) JAGS programs and provide a menu-driven interface for those unfamiliar with the S language: that interface is still available *via* `codamenu()`. See also the article by the authors in the March 2006 issue of *R News* at `http://www.r-project.org/doc/Rnews/Rnews_2006-1.pdf`.

For our purposes the package is more useful as providing means to work with R objects of classes `"mcmc"` and `"mcmc.list"`. The first is used to represent a single MCMC run, and the second a collection of parallel MCMC runs (of the same length). All the sampling functions we use from packages `MCMCpack`, `BRugs` and `rjags` return objects of one of these classes, and they can also be constructed from matrices by functions `mcmc` and `as.mcmc`: there are examples in the practicals.

A single run is a multiple time series with observations at each time on each *monitored* variable, and class `"mcmc"` closely resembles the `"ts"` class, in that it has methods for `start()`, `end()` and `window()`. Rather than having a 'frequency', `"mcmc"` objects have a thinning parameter extracted by `thin()`, and can be re-thinned by the `window()` function. (Recall that *thinning* is sampling every $m \geq 1$ steps, so it is a different name for the concept of time-series frequency.)

Individual runs can be extracted from class `"mcmc.list"` by `[[ ]]` indexing, and a subset of runs by `[ ]` indexing, as one would expect for a list. Using `[ ]` on a single run operates on the matrix of values, which has a (named) column for each monitored parameter.

Printing a CODA object prints all the values with a header (similar to a time series): using `summary()` gives a compact summary. One part of the latter which may need further explanation is the column `Time-series SE`: this is an estimate of the standard error of the quoted mean estimate taking autocorrelation into account, and is used by function `effectiveSize` to compute the equivalent sample size of independent samples (summed across runs).

A wide range of plotting facilities is provided. There are methods for the `plot` function, and these (by default) call `traceplot` and `densplot` to produce line plots and density plots respectively. There are methods for several of the plotting functions in package `lattice`, such as `xyplot`, `densityplot`, `qqmath` and `levelplot`, and also a function `acfplot` for lattice plots of autocorrelations. Function `autocorr.plot` is another way to plot autocorrelations, and `cumuplot` another way to plot the series as cumulative means.

Function `HPDinterval` extracts *H*ighest *P*osterior *D*ensity intervals for each monitored parameter (assuming monotonic tails of the posterior density).

Functions `gelman.diag` (Gelman & Rubin, 1992), `geweke.diag` (Geweke, 1992), `heidel.diag` (Heidelberger & Welch, 1983) and `raftery.diag` (Raftery & Lewis, 1992) implement some of the convergence diagnostics: `gelman.plot` and `geweke.plot` have corresponding graphical representations.

You may see messages from `coda` about '`algorithm did not converge`' from `glm.fit`. These result from using `glm` as part of the estimation of the spectral density at zero, used for the `Time-series SE` estimates and by some of the convergence diagnostics.