Computer-Intensive Statistics

1 What is 'Computer-Intensive Statistics'?

'Computer-intensive statistics' is statistics that could only be done with *'modern'* computing resources, typically either

- Statistical inference on small problems which needs a lot of computation to do at all, or to do well. Quite small datasets can need complex models to explain, and even simple models can need a lot of computation for a realistic analysis (especially where dependence is involved).
- Statistical inference on 'huge' problems.

All of these terms are relative and change quite rapidly—according to the most commonly quoted version of *Moore's Law* (see section 6 and Ripley (2005)) computing power will quadruple during your doctoral studies.

One very important idea for doing statistical inference 'well' on analytically intractable statistical models (that is, most real-world ones) is to make use of *simulation*. So most of this module could be subtitled *simulation-based inference*, 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.*

These notes go beyond what will be covered in lectures—they are intended to give pointers to further issues and to the literature.

[1]Thanks to Anthony Davison, Martyn Plummer and Ruth Ripley for their comments.

2 Simulation-based Inference

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 is another.

The main reference for this section is Ripley (1987, §7.1).

Monte-Carlo tests

Suppose we have a fully-specified null hypothesis, and a test statistic T for which small values indicate departures from the null hypothesis. We can always simulate m samples $t_1, \ldots, t_m$ under the null hypothesis, and use these to obtain an indication of where the observed value T lies on the null distribution. For example, consider a dataset on the amounts of shoe wear in an experiment reported by Box, Hunter & Hunter (1978). There were two materials (A and B) that were randomly assigned to the left and right shoes of 10 boys.

Table 1: Data on shoe wear from Box, Hunter & Hunter (1978).

boy	A	B
1	13.2 (L)	14.0 (R)
2	8.2 (L)	8.8 (R)
3	10.9 (R)	11.2 (L)
4	14.3 (L)	14.2 (R)
5	10.7 (R)	11.8 (L)
6	6.6 (L)	6.4 (R)
7	9.5 (L)	9.8 (R)
8	10.8 (L)	11.3 (R)
9	8.8 (R)	9.3 (L)
10	13.3 (L)	13.6 (R)

A paired t-test gives a t-value of -3.3489 and two-tailed p-value of 0.85% for no difference between the materials. The sample size is rather small, and one might wonder about the validity of the t-distribution. An alternative for a randomized experiment such as this is to base inference on the permutation distribution of `d = B-A`. Figure 1 shows that the agreement is very good.

Monte Carlo tests[2] are a closely related (but not identical) idea. If the null hypothesis is true, we have $m + 1$ exchangeable samples from the null distribution, one natural and m by simulation. Thus the probability that T is the kth smallest or smaller is *exactly* $k/(m + 1)$ provided we can ignore ties. To do so we now assume that T has a continuous null distribution.

By choosing k and m suitably, say $(1, 19)$, $(5, 99)$, $(50, 999)$ we can derive an exact significance test at any desired level. Note that the experiment can be stopped early if k simulated

[2]Usually attributed to a published comment by George Barnard in 1963.

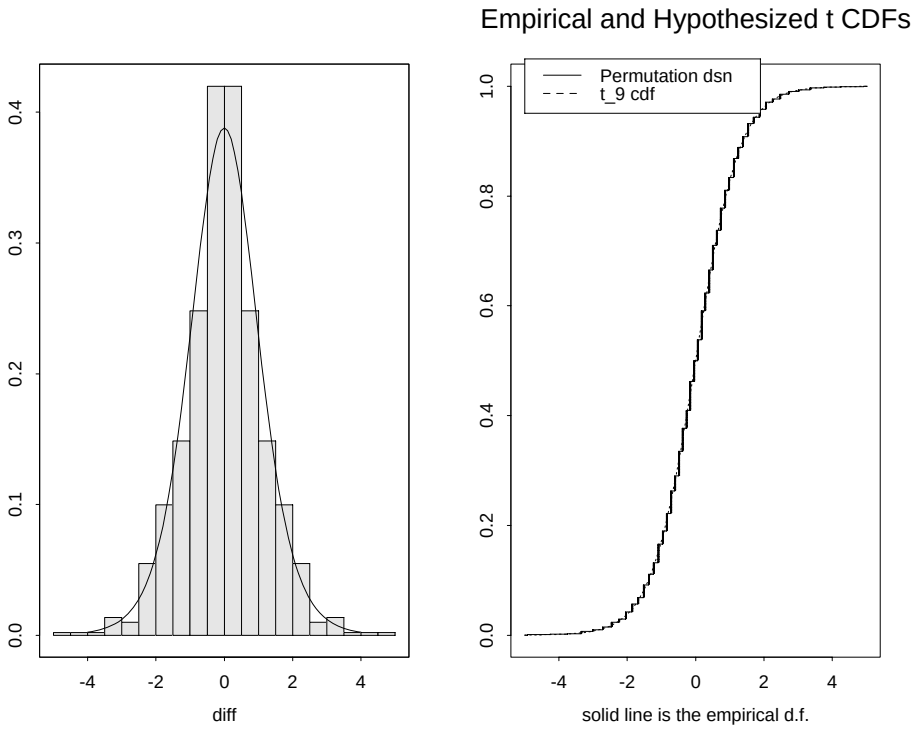

Figure 1: Histogram and empirical CDF of the permutation distribution of the paired t–test in the shoes example. The density and CDF of t_9 are shown overlaid. (Figure 5.5 of Venables & Ripley (2002).)

values less than T have been observed – if the null hypothesis is true this will take a mean of $2k$ trials. However, doing the test indicates some evidence against the null hypothesis, so we should not expect early stopping to be typical.

Power considerations

One common objection to Monte-Carlo tests is that different users will get different results. One answer is

> *So what?; they would have used different test statistics, or deleted different outliers, or chosen different significance levels or ....*

Effectively the actual significance level conditional on the simulations varies and only has average α. This will be reflected in a loss in power. Detailed calculations by Jöckel (1986) give

$$\frac{\text{power of MC test}}{\text{power of exact test}} \geq 1 - \frac{E|Z-\alpha|}{2\alpha} \approx 1 - \left[\frac{1-\alpha}{2\pi m\alpha}\right]^{1/2}$$

where $Z \sim \text{beta}(\alpha(m+1), (1-\alpha)(m+1))$. This is a *lower* bound, and ranges, for $\alpha = 5\%$, from 64% for $m = 19$ through 83% for $m = 99$ to 94.5% for $m = 999$. Asymptotic results show better behaviour if the statistic is asymptotically normal, for example.

Note that our discussion has been entirely about simple null hypotheses. There have been some suggestions about how to use Monte Carlo tests for composite null hypotheses, and of course there are many standard arguments to reduce composite null hypotheses to simple ones.

as named by Buckland (1984). These differ in a small but important detail from the bootstrap confidence intervals we will meet in section 3.

Monte Carlo tests are only defined for a single null hypothesis, so can not easily be inverted to form a confidence interval. Some pivotal quantity is needed. Suppose $\widehat{\theta}$ is a consistent estimator of θ with corresponding CDF F_θ. Let θ^* be a sample from $F_{\widehat{\theta}}$. We want to use the variation of θ^* about $\widehat{\theta}$ to infer the variation of $\widehat{\theta}$ about θ.

Suppose we have a location family. Then

$$\widehat{\theta} - \theta \sim F_0, \qquad \theta^* - \widehat{\theta} \sim F_0$$

so we can obtain upper and lower prediction limits for θ^* by

$$\begin{aligned} L &= F_{\widehat{\theta}}^{-1}(\alpha/2) = \widehat{\theta} + F_0^{-1}(\alpha/2) \\ U &= F_{\widehat{\theta}}^{-1}(1-\alpha/2) = \widehat{\theta} + F_0^{-1}(1-\alpha/2) \end{aligned}$$

either analytically or *via* simulation from the empirical CDF of θ^*. The conventional $(1-\alpha)$ confidence interval for θ is

$$\theta \in \left(\widehat{\theta} - F_0^{-1}(1-\alpha/2), \widehat{\theta} - F_0^{-1}(\alpha/2)\right) = (2\widehat{\theta} - U, 2\widehat{\theta} - L)$$

Thus we get a confidence interval for θ by reflecting the distribution of θ^* about $\widehat{\theta}$. This is the Monte-Carlo confidence interval. If the family is only locally a location family, the confidence interval is only approximately correct. With local scale families, the same arguments apply to $\log\theta$.

Note carefully the difference between this and the bootstrap. The bootstrap resamples (with replacement) from the data. These methods sample from the fitted distribution—sometimes they are called the *parametric bootstrap*, as distinct from the *non-parametric bootstrap*. If we had fitted a completely general class of distributions, the fitted distribution $F_{\widehat{\theta}}$ would be the empirical distribution function F^* which assigns mass $1/n$ to each data point. Then independent sampling from F^* is bootstrap resampling, and the Monte-Carlo confidence intervals are what are known as *basic* bootstrap intervals.

Monte Carlo Likelihood

as termed by Geyer & Thompson (1992). In general the likelihood equations for a canonical exponential family equate the observed value to the expectation of a sufficient statistic, the parameter controlling the expectation. The difficulty can arise in evaluating the expectation.

Consider the so-called Strauss model of spatial inhibition (Ripley, 1988, §4), which has pdf for n points $x_i \in D \subset \mathbb{R}^d$ of

$$f_c(x_1, \ldots, x_n) = a(c)c^{t(x_1,\ldots,x_n)} \tag{1}$$

where $t()$ computes the number of pairs of points closer than distance R, and $0 \le c \le 1$. The log-likelihood includes $\log a(c)$ and this is unknown.

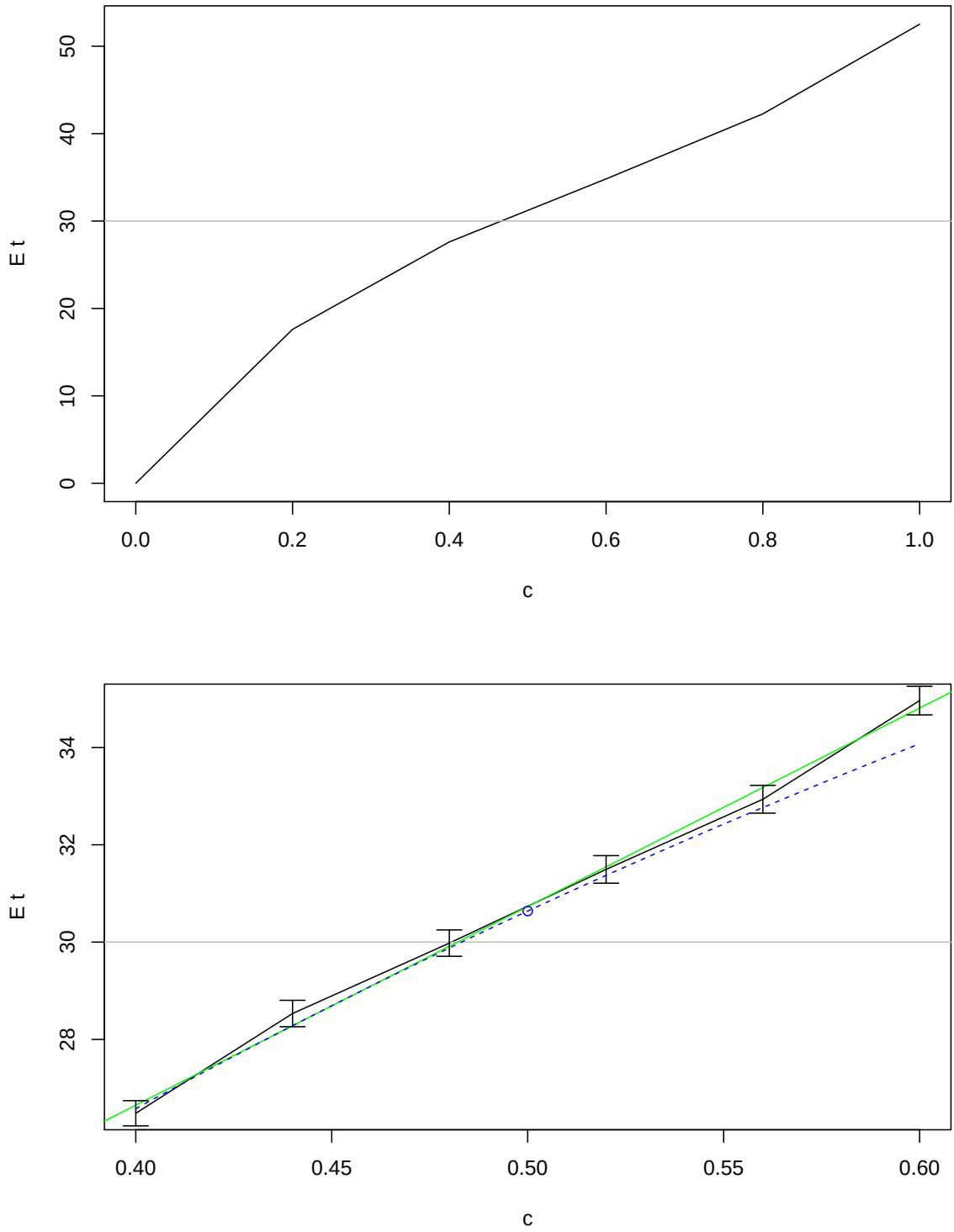

Figure 2: Fitting the Strauss model (1). The top figure shows the average of 100 simulations at five values of c. The lower figure shows the means and 95% confidence intervals based on 1000 simulations at six values of c, plus (green) the fitted regression line and (blue, dashed) values estimated by polysampling from 1000 simulations at $c = 0.5$. The grey line is the observed value.

However, we do not actually need $a(c)$, for the MLE of c satisfies

$$t(x_1, \ldots, x_n) = E_{\widehat{c}}[t(X_1, \ldots, X_n)] \tag{2}$$

where the right-hand side can not be expressed in a simple form. We can however estimate the RHS by simulation from the density (1), as in the preliminary material. An example is given in Figure 2, for a data set in which the observed value of $t(x_1, \ldots, x_n)$ was 30. Note that by importance sampling we can estimate the RHS of (2) for a range of c from simulations at one value, an idea sometimes known as *polysampling*. The idea is that

$$E_c[t(X_1, \ldots, X_n)] = E_{c_0}\left[t(X_1, \ldots, X_n)\frac{f_c(X_1, \ldots, X_n)}{f_{c_0}(X_1, \ldots, X_n)}\right]$$

so we can take a series of samples at $c = c_0$, replace the expectation on the RHS by an average over those samples, and thereby estimate the LHS for any c. The downside is that the estimator is likely to be a good estimator only for c near c_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. So-called *bridge sampling* (Meng & Wong, 1996) uses this idea for simulations at two values of c: see also Gamerman & Lopes (2006, pp. 242–3). Note that these ideas *do* need at least ratios of $a(c)$: see computer exercise 3.

Finding marginals and conditionals

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. 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. For example, the Lauritzen–Spiegelhalter message-passing algorithm's *raison d'être* is to be able to compute marginals after conditioning on evidence. 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.

Conditioning is an issue in the Bayesian setting known as *ABC*.[3] This is a rejection method of sampling from $P(\theta \mid Y = y)$ by first sampling from a distribution (maybe the prior) for θ, then from $P(Y \mid \theta)$ and accepting samples of θ for which $Y = y$. Of course in realistic scenarios this never happens, so instead samples are retained for which some summary of Y is close to that summary of y.

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. In the preliminary exercises you were asked to compare simulations of 71 points in a square with some data – this is a 142-dimensional problem and we have[4] sophisticated multi-dimensional ways to compare such patterns. For another example, ways to look for outliers in multidimensional datasets (Cook & Swayne, 2007) may screen 1,000s or more two-dimensional projections.

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[5] 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

[3] originally *Approximate Bayesian Computation*, http://en.wikipedia.org/wiki/Approximate_Bayesian_computation.

[4] including the human visual system.

[5] 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).

If the proposal used in ABC is not the prior, SIR can be used to reweight the sample.

Stochastic Approximation

An alternative is to solve equation (2) 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[6] 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.

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

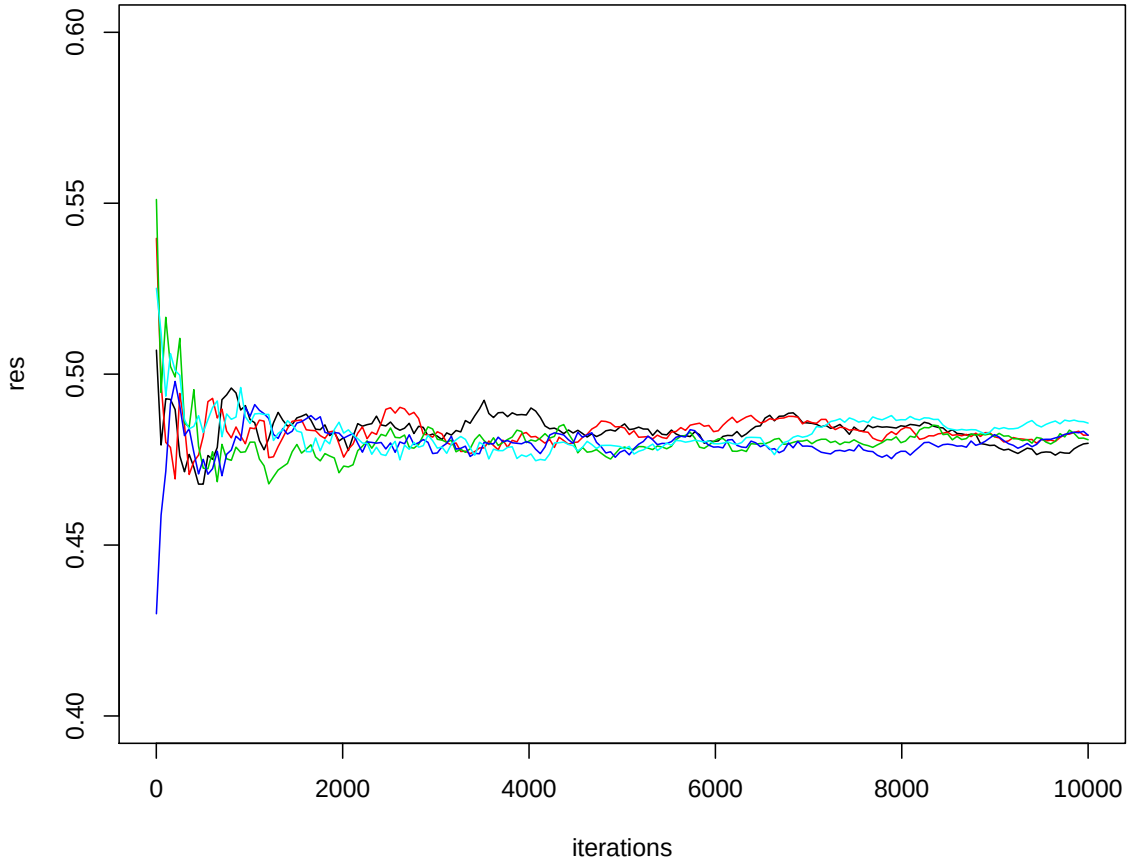

Figure 3: Five runs of fitting the Strauss model (1) by stochastic approximation from $U(0.4, 0.6)$ starting values with $\gamma = 0.7$.

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 a low-energy configuration of the atoms, it is natural[7] 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

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

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)

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

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

or try out http://www.stats.ox.ac.uk/~ripley/APTS2013/SimulatedAnnealing.R

3 Bootstrapping

Suppose we were interested in inference about the correlation coefficient θ of n IID[8] pairs (x_i, y_i) for moderate n, say 15 (as Efron (1982) apparently was). If we assume that the samples are jointly normally distributed, we might know that there is some approximate distribution theory (using Fisher's inverse tanh transform), but suppose we do not wish to assume normality?

We could do a simulation experiment: repeat R times sampling n pairs and compute their correlation, which gives us a sample of size R from the population of correlation coefficients. **But** to do so, we need to assume both a family of distributions for the pairs **and** a particular parameter value (or a distribution of parameter values).

The *bootstrap* procedure is to take m samples from $\boldsymbol{x}$ *with replacement* and to calculate $\widehat{\theta}^*$ for these samples, where conventionally the asterisk is used to denote a bootstrap resample. Note that the new samples consist of an integer number of copies of each of the original data points, and so will normally have ties. Efron's idea[9] was to assess the variability of $\widehat{\theta}$ about the unknown true θ by the variability of $\widehat{\theta}^*$ about $\widehat{\theta}$. For example, the bias of $\widehat{\theta}$ might be estimated by the mean of $\widehat{\theta}^* - \widehat{\theta}$.

Efron coined the term *the bootstrap* in the late 1970s, named after Baron Münchhausen who, it is said,

> *'finding himself at the bottom of a deep lake,*
> *thought to pull himself up by his bootstraps'*

or sometimes from a swamp or marsh by his 'pigtail'. (Fictional, of course, ca 1785, although the Baron was a historical person.)

and said

> *Bootstrap and jackknife algorithms don't really give you something for nothing.*
> *They give you something you previously ignored.*

How can we use the bootstrap resamples to do inference on the original problem, and under what circumstances is such inference valid? Note that the bootstrap resample is unlike the original sample in many ways, perhaps most obviously that (with very high probability) it contains ties.

Bootstrapping is most commonly used

- as part of a procedure to produce more accurate confidence intervals for parameters than might be obtained by classical methods (including those based on asymptotic normality). Often this is very similar to using more refined asymptotic statistical theory, and I once heard bootstrapping described as

  > *'a way to do refined asymptotics without employing the services of Peter Hall'*

- to alleviate biases.

[8] independent and identically distributed.

[9] Others have claimed priority: for example Simon claims in the preface of Simon (1997) to have discovered it in 1966. See also Hall (2003).

In part because it is so simple to describe and easy to do, bootstrapping has become popular and is not infrequently used when it is not valid. For a careful account by two authorities on the subject, see Davison & Hinkley (1997). For another viewpoint by an evangelist of bootstrapping for model validation, see Harrell (2001, Chapter 5). Efron & Tibshirani (1993), Shao & Tu (1995) and Chernick (2008) are complementary material. Hall (1992) covers the (asymptotic) theory. *Statistical Science* **18(2)** (May 2003) has several articles commemorating the *Silver Anniversary of the Bootstrap*.

Efron's original bootstrap is an IID sample of the same size as the original dataset from the *empirical distribution function*. so it is another example of simulation-based inference, using a *non-parametric* rather than *parametric* model of the data. The idea can easily be extended to other non-parametric models, and using a kernel-density estimate[10] of the underlying distribution is called a *smoothed bootstrap*. So that instead of assuming a particular parametric fit, we assume a particular non-parametric fit. This may be more flexible[11], but in both cases we have a *plug-in* estimator—that is we fit a single model from our family and act as if it were the true model.

As a very simple first example, suppose that we needed to know the median m of the `galaxies` data of Roeder (1990) shown in Figure 4. The obvious estimator is the sample median, which is 20 834 km/s. How accurate is this estimator? The large-sample theory says that the median is asymptotically normal with mean m and variance $1/4n\,f(m)^2$. But this depends on the unknown density at the median. We can use our best density estimators to estimate $f(m)$, but we can find considerable bias and variability if we are unlucky enough to encounter a peak (as in a unimodal symmetric distribution). The density estimates give $f(m) \approx 0.13$. Let us try the bootstrap:

```
1/(2*sqrt(length(gal))*0.13)
[1] 0.42474
> library(boot)
> gal.boot <- boot(gal, function(x,i) median(x[i]), R=1000)
> gal.boot
Bootstrap Statistics :
    original   bias    std. error
t1*   20.834 0.038747     0.52269
```

which was effectively instant and confirms the adequacy of the large-sample mean and variance for our example (if Efron's idea is correct). In this example the bootstrap resampling can be avoided, for the bootstrap distribution of the median can be found analytically (Efron, 1982, Chapter 10; Staudte & Sheather, 1990, p. 84), at least for odd n. The bootstrap distribution of $\widehat{\theta}_i$ about $\widehat{\theta}$ is far from normal (Figure 5). Choosing the median (a discontinuous function of the data) disadvantaged the simple bootstrap and e.g. a smoothed bootstrap would be preferred in practice.

Note that the bootstrap principle, that the variability of $\widehat{\theta}$ about the unknown true θ can be assessed by the variability of $\widehat{\theta}^*$ about $\widehat{\theta}$, is *not* always valid. For example, the bias of $\widehat{\theta}$ is not mirrored in the bias of $\widehat{\theta}^*$ when bootstrapping density estimation or curve fitting (Bowman &

[10] Note that sampling from a constant-bandwidth kernel-density estimate amounts to resampling from the original data and then adding a random variable drawn from the kernel as a density, so this is the procedure known as *jittering*.

[11] although large parametric families such as spline models for log densities and neural networks can be arbitrarily flexible.

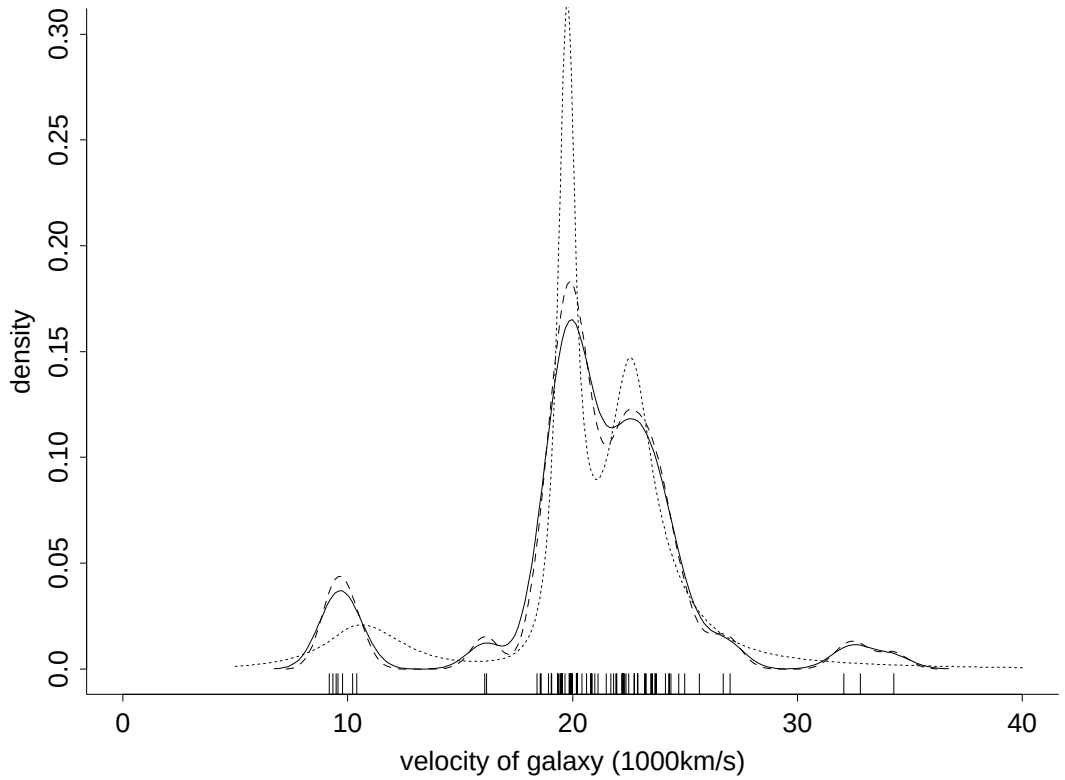

Figure 4: Density estimates for the 82 points of the `galaxies` data. The solid and dashed lines are Gaussian kernel density estimates with bandwidths chosen by two variants of the Sheather–Jones method. The dotted line is a logspline estimate. From Venables & Ripley (2002).

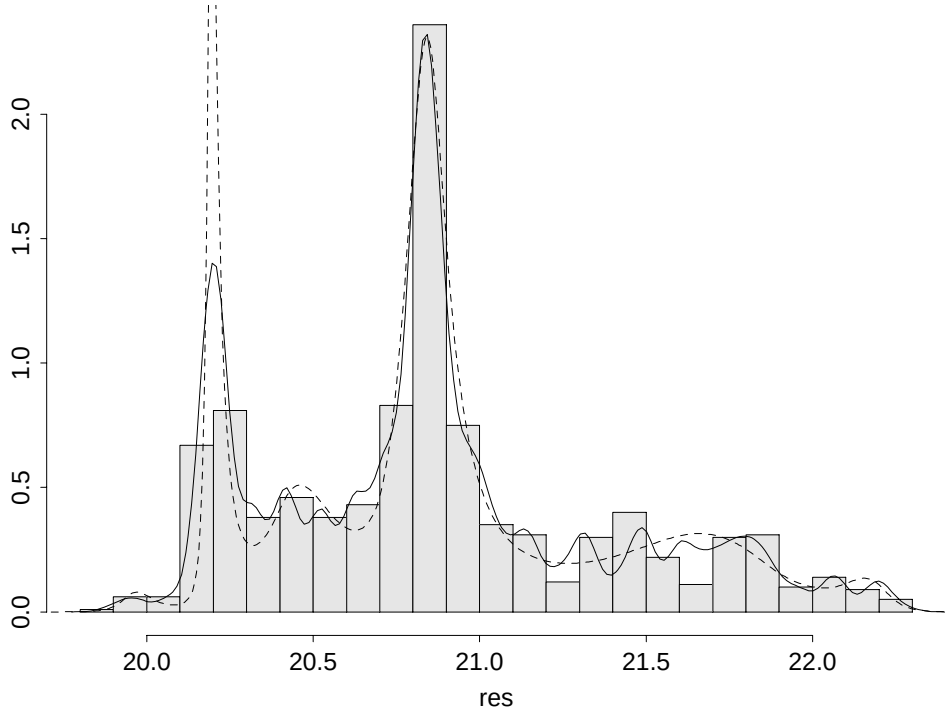

Figure 5: Histogram of the bootstrap distribution for the median of the `galaxies` data, with a kernel density estimate (solid) and a logspline density estimate (dashed). From Venables & Ripley (2002).

Azzalini, 1997, pp. 44, 82). To quote lecture notes by Peter Hall

> *There is a "meta theorem" which states that the standard bootstrap, which involves constructing a resample that is of (approximately) the same size as the original sample, works (in the sense of consistently estimating the limiting distribution of a statistic) if and only if that statistic's distribution is asymptotically Normal.*
>
> *It does not seem possible to formulate this as a general, rigorously provable result, but it nevertheless appears to be true.*

Although unstated, it seems clear that this is not intended to cover semi-parametric problems such as density estimation. It hints at other exceptions, for example extreme-value statistics.[12] For more examples, see Davison *et al.* (2003).

The principle is valid often enough that users miss the exceptions and apply it uncritically. To quote the wisdom of Davison & Hinkley (1997, p. 4)

> *Despite its scope and usefulness, resampling must be carefully applied. Unless certain basic ideas are understood, it is all too easy to produce a solution to the wrong problem, or a bad solution to the right one. Bootstrap methods are intended to help avoid tedious calculations based on questionable assumptions, and this they do. But they cannot replace clear critical thought about the problem, appropriate design of the investigation and data analysis, and incisive presentation of conclusions.*

Perhaps the only point here that is particularly apposite to bootstrapping is the "all too easy". It is also easy to apply regression methods to datasets that have other structure (for example, were collected in groups, so mixed-effects models might be more appropriate), and indeed outside simple textbook problems choosing a suitable non-parametric resampling model is no easier than choosing a suitable parametric model. Think about survival problems or time series or spatial patterns or complex surveys for example. Even if missing data are present we have to model the way in which it might occur in other samples.

Contrast this with the embittered comments of Simon (1997)

> *The simple fact is that resampling devalues the knowledge of conventional mathematical statisticians, and especially the less competent ones. By making it possible for each user to develop her/his own method to handle each particular problem, the priesthood with its secret formulaic methods is rendered unnecessary.*

This seems more generally aimed at simulation-based inference than just resampling.

Performance assessment

which is the main part of what Harrell (2001) calls *model validation*. Suppose we have selected a model for, say, regression or classification. Then we expect the model to perform better on our dataset than in future, because both the model (variables used, transformations etc) and the

[12] for which the 'm-out-of-n bootstrap' when $m < n$ and $m/n \to 0$ provides an alternative with valid theory: see Politis *et al.* (1999).

parameter values have been chosen by looking at that dataset. Part of performance assessment is to predict how well the chosen procedure will do in real-world testing—most of the many approaches are discussed in Ripley (1996, §2.7).

One computer-intensive approach is *cross-validation*, to repeatedly keep back a small part of the dataset, do the model selection on the rest and then predict performance on the part held back and then (in some sense) average to estimate the 'out-of-sample' performance. That is a very general method and used sensibly gives very reliable results—but that has not stopped some developers of competitor methods giving it a bad press.

An extreme form is sometimes used: in *leave-one-out cross-validation* each case is left in left out in turn and compared with its prediction from the rest. This is more problematic than leaving out say 10% each time (Ripley, 1996).

To be concrete, suppose we have a classification procedure and the performance measure is the error rate, the proportion of examples incorrectly classified. Then the 'apparent' misclassification rate on the training data will clearly be biased downwards. Estimating how much bias was a simple (and early) application of the bootstrap. Take a series of new training sets by resampling the original, and fit a model to each new training set, and predict at the original training set. The problem here is that the new and original training sets are not distinct, and Efron (1983); Efron & Tibshirani (1997) proposed the '.632' bootstrap. This weights the apparent error rate and the error rate on those original examples which do not appear in the resampled training set as $0.368 : 0.632$. Here 0.632 is shorthand for $(1 - 1/e)$, the large-sample probability that a given example appears in the resampled training set.

Note that here we are trying to estimate what Harrell calls the *optimism* of the whole model fitting procedure. He rightly points out in a quote that

> *In spite of considerable efforts, theoretical statisticians have been unable to analyse the sampling properties of [usual multistep modeling strategies] under realistic conditions*

but then fallaciously goes on to conclude

> *that the modeling strategy must be completely specified and then bootstrapped to get consistent estimates of variances and other sampling properties.*

The fallacy is asserting that bootstrapping is the only game in town, whereas many other simulation-based inference methods could be (and have been) used.

Confidence intervals

One approach to a confidence interval for the parameter θ is to use the quantiles of the bootstrap distributions; this is termed the *percentile confidence interval* and was the original approach suggested by Efron. The bootstrap distribution in our example is quite asymmetric, and the intervals based on normality are not adequate. The 'basic' intervals are based on the idea that the distribution of $\widehat{\theta}^* - \widehat{\theta}$ mimics that of $\widehat{\theta} - \theta$. If this were so, we would get a $1 - \alpha$ confidence interval as

$$1 - \alpha = P(L \leq \widehat{\theta} - \theta \leq U) \approx P(L \leq \widehat{\theta}^* - \widehat{\theta} \leq U)$$

so the interval is $(\widehat{\theta} - U, \widehat{\theta} - L)$ where $L + \widehat{\theta}$ and $U + \widehat{\theta}$ are the $\alpha/2$ and $1 - \alpha/2$ points of the bootstrap distribution, say $k_{\alpha/2}$ and $k_{1-\alpha/2}$. (It will be slightly more accurate to estimate these as the $(R+1)\alpha/2$th and $(R+1)(1-\alpha/2)$th ordered values of a sample of size R of $\widehat{\theta}^*$.) Then the basic bootstrap interval

$$(\widehat{\theta} - U, \widehat{\theta} - L) = (\widehat{\theta} - [k_{1-\alpha/2} - \widehat{\theta}], \widehat{\theta} - [k_{\alpha/2} - \widehat{\theta}]) = (2\widehat{\theta} - k_{1-\alpha/2}, 2\widehat{\theta} - k_{\alpha/2})$$

which is the percentile interval reflected about the estimate $\widehat{\theta}$. (This is the same derivation as the Monte-Carlo confidence interval, applied to a non-parametric model.) In asymmetric problems the basic and percentile intervals will differ considerably (as here), and the basic intervals seem more rational. For our example we have

```
boot.ci(gal.boot, conf=c(0.90, 0.95), type=c("norm","basic","perc","bca"))

Level     Normal          Basic          Percentile          BCa
90%   (19.94, 21.65)  (19.78, 21.48)  (20.19, 21.89)  (20.18, 21.87)
95%   (19.77, 21.82)  (19.59, 21.50)  (20.17, 22.07)  (20.14, 21.96)
```

The BC_a intervals are an attempt to shift and scale the percentile intervals to compensate for their biases, apparently unsuccessfully in this example. The idea is that if for some unknown increasing transformation g we had $g(\widehat{\theta}) - g(\theta) \sim F_0$ for a *symmetric* distribution F_0, the percentile intervals would be exact. Suppose more generally that if $\phi = g(\theta)$,

$$g(\widehat{\theta}) - g(\theta) \sim N(-w\,\sigma(\phi), \sigma^2(\phi)) \qquad \text{with } \sigma(\phi) = 1 + a\,\phi$$

Let $U = g(\widehat{\theta})$. Then $U = \phi + (1 + a\,\phi)(Z - w)$ for $Z \sim N(0, 1)$ and hence

$$\log(1 + a\,U) = \log(1 + a\,\phi) + \log(1 + a\,(Z - w))$$

which is a pivotal equation in $\zeta = \log(1 + a\,\phi)$. Thus we have a α confidence limit for ζ, as

$$\zeta_\alpha = \log(1 + a\,u) - \log((1 + a\,(-z_\alpha - w))$$

(using $-z_\alpha = z_{1-\alpha}$) and hence one for ϕ as

$$\phi_\alpha = u + (1 + a\,u)\frac{w + z_\alpha}{1 - a\,(w + z_\alpha)}$$

Then the limit for θ is $\theta_\alpha = g^{-1}(\phi_\alpha)$. We do not know g, but if $u = g(\widehat{\theta})$ and $U^* = g(\widehat{\theta}^*)$

$$P^*(\widehat{\theta}^* < \theta_\alpha \,|\, \widehat{\theta}) = P^*(U^* < \phi_\alpha \,|\, u) = \Phi\left(w + \frac{\phi_\alpha - u}{1 + a\,u}\right) = \Phi\left(w + \frac{w + z_\alpha}{1 - a(w + z_\alpha)}\right)$$

Thus the α confidence limit for θ is given by the $\widehat{\alpha}$ percentile of the bootstrap distribution, where

$$\widehat{\alpha} = \Phi\Big(w + \frac{w + z_\alpha}{1 - a(w + z_\alpha)}\Big)$$

Thus if $a = w = 0$ the percentile interval is exact. This is very unlikely, but we can estimate a and w from the bootstrap samples. Now w essentially measures the offset of centre of the distribution, and

$$P^*(\widehat{\theta}^* < \widehat{\theta} \,|\, \widehat{\theta}) = P^*(U^* < u \,|\, u) = \Phi(w)$$

and so w can be estimated by

$$\widehat{w} = \Phi^{-1}\left(\frac{\#\{\widehat{\theta}^* \le \widehat{\theta}\}}{R+1}\right)$$

Estimating a is a little harder: we make a linear approximation to $\widehat{\theta}$ (as a function of the data points) and take one sixth its skewness (third moment divided by standard deviation cubed) in the bootstrap distribution.

For a smaller, simpler example, consider the data set on page 2.

```
> t.test(B - A)
95 percent confidence interval:
 0.133 0.687
> shoes.boot <- boot(B-A, function(x,i) mean(x[i]), R=1000)
> boot.ci(shoes.boot, type = c("norm", "basic", "perc", "bca"))

Level    Normal          Basic         Percentile         BCa
95%  (0.186, 0.644)  (0.180, 0.650)  (0.170, 0.640)  (0.210, 0.652)
```

There is a fifth type of confidence interval that `boot.ci` can calculate, which needs a variance v^* estimate of the statistic $\widehat{\theta}^*$ from each bootstrap sample. Then the confidence interval can be based on the basic confidence intervals for the *studentized* statistics $(\widehat{\theta}^* - \widehat{\theta})/\sqrt{v^*}$.

```
mean.fun <- function(d, i) {
  n <- length(i)
  c(mean(d[i]), (n-1)*var(d[i])/n^2)
}
> shoes.boot2 <- boot(B - A, mean.fun, R = 1000)
> boot.ci(shoes.boot2, type = "stud")

Level  Studentized
95%   (0.138, 0.718)
```

Some caution is needed here. First, despite three decades of work and lots of theory that suggest bootstrap methods are well-calibrated, we can get as large discrepancies as we have here. Second, this is only univariate statistics, and standard bootstrap resampling is only applicable to IID samples.

Note that the bootstrap distribution provides some diagnostic information on the assumptions being made in the various confidence intervals.

Theory for bootstrap confidence intervals

There are two ways to compare various types of bootstrap confidence intervals. One is empirical comparisons as above. Another is to work out the asymptotic theory to a high enough level of detail to differentiate between the methods.

The principal property of a confidence interval or limit is its coverage properties. We want for an upper confidence limit $\widehat{\theta}_\alpha$ that

$$P_\theta(\theta \le \widehat{\theta}_\alpha) = \alpha + O(n^{-a})$$

for a large a. Obviously we would like exact confidence limits (no remainder term), but they are in general unattainable, and so we aim for the best possible approximation.

Two points to note: a confidence interval or limit can have good coverage but be far from optimal (as measured by length, say), and a confidence interval can have good coverage but be far from equi-tailed.

We can consider a hierarchy of methods of increasing accuracy.

- A normal-based confidence interval, for example with standard deviation based on the bootstrap distribution, and with a bootstrap bias correction. This has $a = 1/2$ (in general, but we won't keep mentioning that).
- Basic bootstrap confidence limits. These have $a = 1/2$, *but* confidence intervals have $a = 1$ and are said to be *second-order accurate*.
- Percentile limits and intervals. The same story as the basic limits and intervals. As we have seen empirically, both of these tend to fail to centre the interval correctly, so achieve second-order accuracy for intervals at the expense of having unequal tails.
- BC_a limits and intervals both have $a = 1$, provided a and w are estimated to $O(n^{-1/2})$.
- The studentized method also has $a = 1$, provided the variance estimate used is accurate to $O(n^{-1/2})$.

To make the limitations of these results clearer, note that they apply equally to any continuous monotone transformation $\phi(\theta)$ (for example a log or arcsin or logistic transformation), but empirical studies (e.g. Davison & Hinkley, 1997, §5.7) show that using the right transformation can be crucial.

Double bootstrapping

Another idea is to re-calibrate a simpler confidence limit or interval, that is to use $\widehat{\theta}_\beta$ for some $\beta \neq \alpha$ to achieve more accurate coverage properties. How do we choose β? The double bootstrap uses a second tier of bootstrapping to estimate the coverage probability of $\widehat{\theta}_\beta$, and then solves for β on setting this estimate equal to α.

It transpires that applying this idea to the normal confidence interval produces the studentized confidence interval, but applied to the basic confidence interval it produces coverage accurate to $a = 2$.

Double bootstrapping appears to require large numbers of replications, say a million samples if we take 1000 in each of the tiers. This can be reduced to more manageable numbers by using *polysampling* (page 6, Davison & Hinkley (1997, §9.4.4) and Morgenthaler & Tukey (1991)), but computers may be fast enough these days.

Bootstrapping linear models

In statistical inference we have to consider what might have happened but did not. Linear models can arise exactly or approximately in a number of ways. The most commonly considered

form is

$$Y = X\beta + \boldsymbol{\epsilon}$$

in which only $\boldsymbol{\epsilon}$ is considered to be random. This supposes that in all (hypothetical) repetitions the same x points would have been chosen, but the responses would vary. This is a plausible assumption for a designed experiment and for an observational study with pre-specified factors.

Another form of regression is sometimes referred to as the *random regressor* case in which the pairs (x_i, y_i) are thought of as a random sample from a population and we are interested in the regression function $f(x) = E\{Y \mid X = x\}$ which is assumed to be linear. However, it is common to perform conditional inference in this case and condition on the observed xs, converting this to a fixed-design problem. For example, in the Scottish hill races dataset[13] the inferences drawn depend on whether certain races, notably Bens of Jura, are included in the sample. As they were included, conclusions conditional on the set of races seems most pertinent.

These considerations are particularly relevant when we consider bootstrap resampling. The most obvious form of bootstrapping is to randomly sample pairs (x_i, y_i) with replacement,[14] which corresponds to randomly weighted regressions. However, this may not be appropriate in not mimicking the assumed random variation and in some examples of producing singular fits with high probability. The main alternative, *model-based resampling*, is to resample the residuals. After fitting the linear model we have

$$y_i = x_i\widehat{\beta} + e_i$$

and we create a new dataset by $y_i = x_i\widehat{\beta} + e_i^*$ where the (e_i^*) are resampled with replacement from the residuals (e_i). There are a number of possible objections to this procedure. First, the residuals need not have mean zero if there is no intercept in the model, and it is usual to subtract their mean. Second, they do not have the correct variance or even the same variance. Thus we can adjust their variance by resampling the *modified residuals* $r_i = e_1/\sqrt{1 - h_{ii}}$ which have variance σ^2.

I see bootstrapping as having little place in least-squares regression. If the errors are close to normal, the standard theory suffices. If not, there are better methods of fitting than least-squares! One issue that is often brought up is that of heteroscedasticity, which some bootstrap methods accommodate—but then so do Huber–White 'sandwich' estimators.

The distribution theory for the estimated coefficients in robust regression is based on asymptotic theory, so we could use bootstrap estimates of variability as an alternative. Resampling the residuals seems most appropriate for the `phones` data of Venables & Ripley (2002, p. 157)

```
library(MASS); library(boot)
fit <- lm(calls ~ year, data=phones)
summary(fit)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -260.059    102.607  -2.535   0.0189
```

[13] Venables & Ripley (2002, p. 8–10, 152–5).

[14] Davison & Hinkley (1997) call this *case-based resampling*. Shao & Tu (1995) call it the *paired bootstrap*, in contrast to the *residual bootstrap* we consider next.

```
year           5.041      1.658   3.041   0.0060

ph <- data.frame(phones, res=resid(fit), fitted=fitted(fit))
ph.fun <- function(data, i) {
  d <- data
  d$calls <- d$fitted + d$res[i]
  coef(update(fit, data=d))
}
ph.lm.boot <- boot(ph, ph.fun, R=999)
ph.lm.boot
    ....
     original   bias    std. error
t1* -260.0592  6.32092    100.3970
t2*    5.0415 -0.09289      1.6288

fit <- rlm(calls ~ year, method="MM", data=phones)
summary(fit)

Coefficients:
            Value    Std. Error t value
(Intercept) -52.4230   2.9159   -17.9783
year          1.1009   0.0471    23.3669

ph <- data.frame(phones, res=resid(fit), fitted=fitted(fit))
ph.rlm.boot <- boot(ph, ph.fun, R=999)
ph.rlm.boot
    ....
    original     bias  std. error
t1* -52.4231  3.232142   30.01894
t2*   1.1009 -0.013896    0.40648
```

(The `rlm` bootstrap is not nowadays computer-intensive, but took about 10 mins in S-PLUS for the third edition in 1997.) These results suggest that the asymptotic theory for `rlm` is optimistic for this example, but as the residuals are clearly serially correlated the validity of the bootstrap results is equally in doubt. Statistical inference really does depend on what one considers might have happened but did not.

Shao & Tu (1995, Chapters 7–8) and Davison & Hinkley (1997, Chapters 6–7) consider linear models and extensions such as GLMs and survival models.

How many bootstrap resamples?

Our examples have been based on 1000 resamples θ^*. Largely that figure was plucked from thin air: it was computationally feasible. Is it enough? To answer that question we need to consider the various sources of error. The Monte-Carlo error due to the resampling is one, and since the resampling is independent, the size of the Monte Carlo error will be $O_P(R^{-1/2})$ for R samples. On the other hand, the size of the confidence interval will be $O_P(n^{-1/2})$ and this suggests that to make the Monte Carlo error negligible we should take R to be some multiple of n. Some calculations (Davison & Hinkley, 1997, pp. 35–6) suggest a multiple of 10–40.

However, this is not all there is to it. If we want to compute confidence intervals, we need to be able to estimate fairly extreme quantiles, which suggests we need R around 1000. ($R = 999$ is popular as the $(R+1)p$th quantile is a data point for most popular p.) Moreover, the BC_a

method often needs considerably more extreme quantiles than the originals, and so can require very large bootstrap samples.

Further, there are other sources of (systematic) error, not least the extent to which the distribution of $\theta^* - \widehat{\theta}$ mimics that of $\widehat{\theta} - \theta$. It may be more important to choose a (monotone invertible) transformation $\phi(\theta)$ for which that approximation is more accurate, in particular to attempt variance stabilisation. Finally, if the samples were not independent, then simple bootstrapping will be inappropriate.

Diagnostics

It is (I hope) familiar that after fitting a parametric model such as a regression we look at diagnostics to see if the assumptions have been violated.

In one sense the problem is easier with bootstrap models as they are non-parametric and so make fewer assumptions. However *fewer* but often *more* critical ones, e.g. independent and identically distributed. So we would still like to know if there are observations that are particularly influential on our conclusions. That is hard enough for linear regression!

There is one general approach to bootstrap diagnostics, the use of *jackknife-after-bootstrap* (Davison & Hinkley, 1997, §3.10). We could consider dropping each observation j in turn and redoing the analysis (including all the resampling). However, we can avoid this by looking only at bootstrap resamples in which observation j does not occur and applying the jackknife method of the next section.

The Jackknife

The *jackknife*[15] is an older resampling idea, most often used to attempt bias reduction.

Consider an i.i.d. sample $X_1, \ldots, X_n \sim F$, and consider a statistic $\theta_n = \theta(F_n)$ for the empirical CDF F_n.. Then the 'true value' is $\theta(F)$. Any statistic which is independent of the ordering of the sample can be written in this form.

The jackknife estimates the bias from the n sub-samples of size $n-1$. Let $\widehat{\theta}_{(i)}$ denote the estimate from the sample omitting X_i. Then the estimate of bias is

$$\widehat{BIAS} = (n-1)\Big(\frac{1}{n}\sum \widehat{\theta}_{(i)} - \widehat{\theta}\Big)$$

It can be shown that if

$$\text{bias}(\widehat{\theta}) = E(\widehat{\theta}) - \theta(F) = a_1/n + a_2/n^2 + \cdots$$

then $\widehat{\theta} - \widehat{BIAS}$ has a bias of $O(n^{-2})$. For example, applied to the variance functional it replaces the divisor n by $n-1$. The assumption will be true for smooth functionals θ, but not, for example, for the median.

[15] a Tukey-ism for an idea of Quenouille: in American usage a jackknife is a pocket knife, most often one that has a single blade and folds away into its handle.

Another way to look at the jackknife is *via* the *pseudo-values* given by

$$\tilde{\theta}_i = \widehat{\theta} + (n-1)(\widehat{\theta} - \widehat{\theta}_{(i)})$$

Then these can be regarded as a new 'sample', and the mean and variance estimated from them. Then $\widehat{BIAS}$ is $\widehat{\theta}$ minus the mean of the pseudo-values.

The jackknife estimate of the variance of a statistic is the variance of the pseudo-values divided by n. The pseudo-values could be used to give a confidence interval, but that has proved to be no better than the normal-based confidence intervals based on the mean and variance of the pseudo-values.

Jackknife ideas have been fruitful for bias and (especially) variance estimation, for example in sample surveys—see Rao & Wu (1988) and Shao & Tu (1995, Chapter 6). (Note that this is another area where naïve applications of the bootstrap may be invalid.)

Note that the jackknife and leave-one-out cross-validation use the same resampling plan but a different analysis of the resamples: as a result leave-one-out cross-validation is often incorrectly called *jackknifing*.

Bootstrapping as simulation

Bootstrapping is 'just' simulation-based inference, sampling from a particularly simple model. As such it is subject to all the ways known in the simulation literature[16] to reduce variability. Davison & Hinkley (1997, Chapter 9) discuss many of them, but there are few examples in the many application studies using bootstrapping. If you use bootstrapping in your work, please take heed!

Software

Basic bootstrap resampling is easy, as you saw in the preliminary notes—e.g. just use the R function `sample`.

As for most uses of simulation-based inference the task for software falls into two halves

- Generate the simulations, in this case the resamples.
- Analyse the simulated data.

Once we move away from looking at univariate IID sampling, both become more complicated and less general.

R ships with a package `boot` which is support software for Davison & Hinkley (1997) written[17] largely by Angelo Canty. The workhorse here is function `boot`, which allows several types of resampling/simulation.

`sim = "ordinary"` which has `stype` as one of `"i"` (give indices into the data set), `"f"` (give frequencies for each item) and `"w"` (normalized to one)

[16] and sketched in the preliminary material.

[17] for S-PLUS, and ported to R by me.

`sim = "parametric"` see below.

`sim = "balanced"` stratification.

`sim = "permutation"` resampling without replacement.

`sim = "antithetic"` induce negative correlations in pairs of bootstrap resamples.

It is also possible to specify strata, and importance sampling weights.

`sim = "parametric"` is intended for parametric bootstrapping, but is a completely general mechanism. Here is how it is used for the smoothed bootstrap in the solutions to the preliminary exercises:

```
s <- 0.1 # the standard deviation of a normal kernel
ran.gen <- function(data, mle) {
    n <- length(data)
    rnorm(n, data[sample(n, n, replace = TRUE)], mle)
}
out3 <- boot(nerve, median, R = 1000, sim = "parametric",
             ran.gen = ran.gen, mle = s)
```

Here `mle` is an object representing the parameters to be passed to the simulation routine.

There are two other specialised simulation functions, `censboot` for censored data, and `tsboot` for time series.

The main analysis function is `boot.ci`.

Recent versions of package `boot` support parallel operations on multi-core computers. Although in general the re-sampling is not independent (it is for `sim = "ordinary"` and `sim = "parametric"`) once that is done the evaluation of the statistic for the re-samples (usually the time-consuming part) can be done in parallel.

4 Markov Chain Monte Carlo

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.

The previous paragraph assumes that these goals are achievable, but people do attempt to use MCMC in problems with millions of random variables. Almost inevitably there are some aspects of the process that mix slowly and some that mix fast, and so the choice of MCMC sampling scheme does often need to be linked to the questions of interest.

MCMC can be approached from several angles. The preliminary material took on one approach based on my personal experience, and for variety these notes take another.

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).

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

- 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

$$g(\theta) = \int K(\theta, \phi)\, g(\phi)\, d\phi, \quad \text{where} \quad K(\theta, \phi) = \int p(\theta \mid Y, Z)\, p(Z \mid \phi, Y)\, dZ \tag{3}$$

Under mild conditions we can solve (3) by successive substitution,[18] 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 (3) 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

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

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

[18] Start with some candidate g for $p(\theta \mid Y)$, and repeatedly use (3) 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.

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*[19] 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 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 standard 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.

Detailed balance

Data augmentation and the spatial birth-and-death processes of the preliminary notes 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[20] 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

[19] http://en.wikipedia.org/wiki/Particle_filter, Robert & Casella (2004, Chapter 14).
[20] for some value of 'formal'!

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,

$$\pi_i P_{ij} = \pi_j P_{ji} \tag{4}$$

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[21] 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.

Gibbs sampler

so named by Geman & Geman (1984) but published some years earlier by Ripley (1979) and as examples in earlier papers.

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.

This can easily be shown to satisfy detailed balance.

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[22] 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.

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

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

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 a simple exercise to show that this satisfies detailed balance and the stationary distribution is 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.

This satisfies detailed balance: taking rejection into account this requires

$$\pi(x)q(x, y)\alpha(x, y) = \pi(y)q(y, x)\alpha(y, x)$$

and both sides simplify to

$$\min\{\pi(x)q(x, y), \pi(y)q(y, x)\}$$

The original Metropolis *et al.* scheme had a symmetric transition kernel, so the move is accepted with probability $\min\{1, \pi(y)/\pi(x)\}$. 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 $\alpha(x, y) \equiv 1$).

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[23] 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 at http://www.stats.ox.ac.uk/~ripley/APTS2013/scripts/slice.R.

Other schemes

The only limit on the plethora of possible MCMC schemes is the ingenuity of developers. We saw another scheme, spatial birth-and-death processes, in the preliminary notes. A similar idea, the *reversible jump MCMC* of Green (1995), 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.

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

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.

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.)

One very useful concept is the *effective sample size*, the number of independent samples we would have had to have taken to get the same variance as H_N. We know (Ripley, 1987, p. 144) that

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

where σ^2, (ρ_s) and f are the variance, the autocorrelation sequence and spectral density[24] of the stationary time series $h(X_{mi})$. When you see 'Time-series SE' in the CODA summary, it is estimated using the RHS[25] of this formula.

Thus the ESS is, asymptotically,

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

[24] under at least one of its definitions!

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

Convergence diagnostics

Or 'How do we know when we are close to equilibrium?

This led to much heated discussion in the early 1990s, and several survey papers. The scale of the problem is often dramatically underestimated – more than 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.[26] 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.

Writing about this, Robert & Casella (2004) (which is a second edition) say (p. X)

> *We also spend less time on convergence control, because some of the methods presented [in the 1999 first edition] did not stand the test of time. The methods we preserved in Chapter 12 have been sufficiently tested to be considered reliable.*

and (p. 512)

> *Chapter 12 details the difficult task of assessing the convergence of an MCMC sampler, that is, the validity of the approximation $\theta^{(T)} \sim \pi(x)$, and, correspondingly, the determination of a "long enough" simulation time. Nonetheless, the tools presented in that chapter are mostly hints of (or lack of) convergence, and they very rarely give a crystal-clear signal that $\theta^{(T)} \sim \pi(x)$ is valid for all simulation purposes.*

Readers of other accounts (including their first edition) may come away with a very different impression.

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).

[26] on a 25MHz computer.

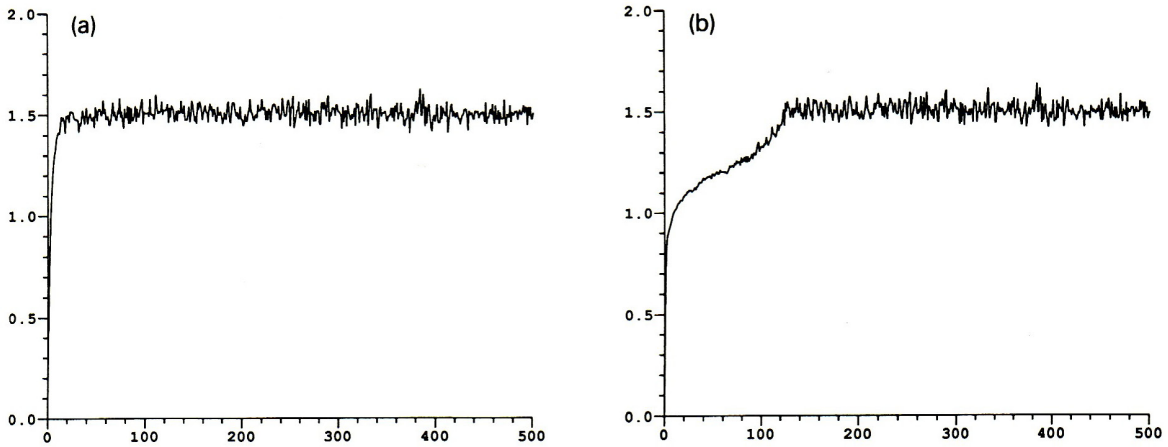

Fig. 2. Pseudolikelihood (a) and (asymptotic) maximum likelihood (b) estimates for 500 sweeps of Metropolis' method.

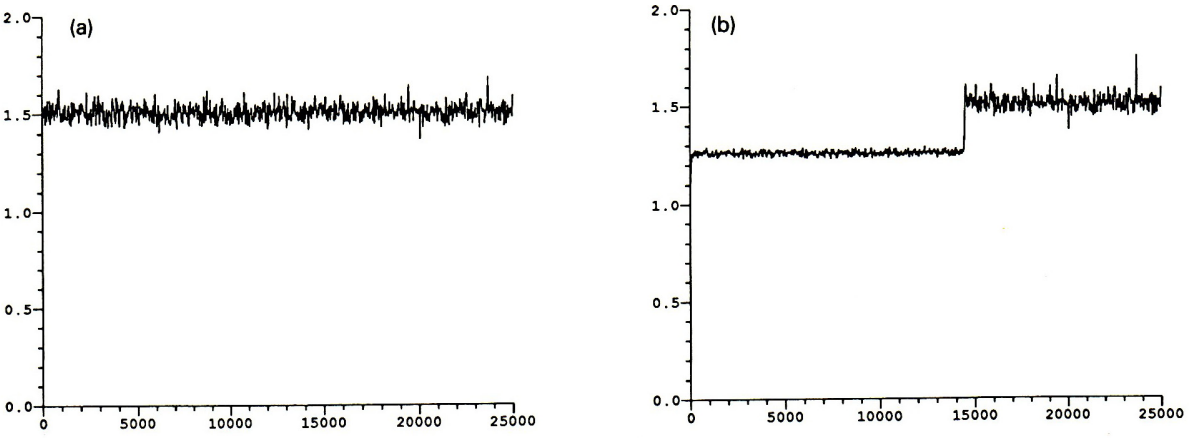

Fig. 3. As Fig. 2, but another sample with 25,000 sweeps!

Figure 6: 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).

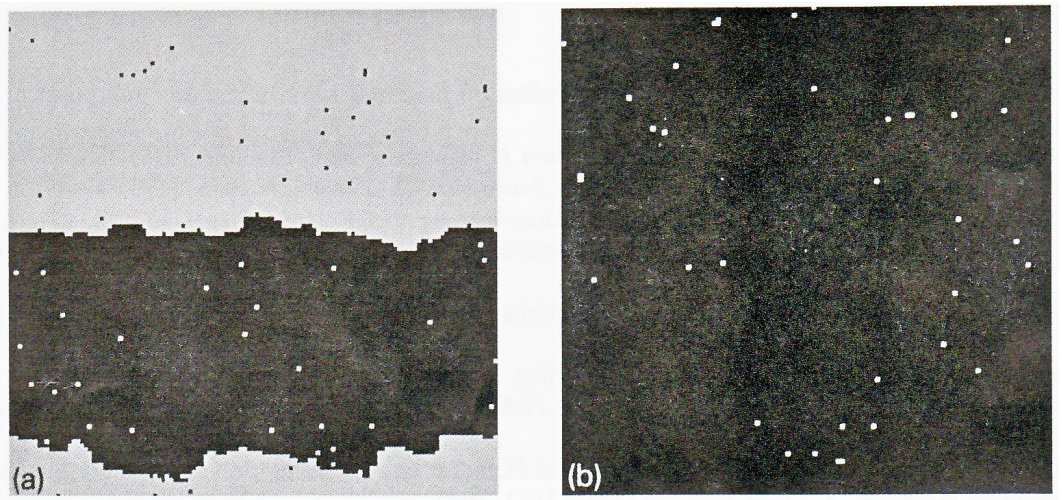

Fig. 4. The realization after 10,000 (a) and 25,000 (b) sweeps of Metropolis' method.

Figure 7: Two snapshots of the second MCMC simulation. From Ripley & Kirkland (1990).

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 designed 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.

- *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 to 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 B.

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.[27] 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 *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.

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),[28] and as a method of integration in Evans & Swartz (2000).

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

Software

Because MCMC is a meta-algorithm, there are very many specific applications and corresponding software including many R packages. (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 examples and labs 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

[27] 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.

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

producing output suitable for analysis by `coda`. The choice of models is slanted towards social science applications,[29] 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[30] 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 A) 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[31] (Lunn *et al.*, 2000) with a GUI interface: this was then re-implemented as OpenBUGS.[32] Ntzoufras (2009) describes how to use WinBUGS as a standalone Windows program. See Lunn *et al.* (2009) for some of the history of BUGS.

JAGS[33] is an Open Source program by Martyn Plummer written in C++ that re-implements the BUGS language. It is much more recent and does not (yet) 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. Despite its blurb, the examples in Jackman (2009) are done in JAGS—for he is a Mac user.

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!'*

[29] such as item response models

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

[31] which as its name suggests is for Windows only, `http://www.mrc-bsu.cam.ac.uk/bugs`: as a 32-bit program it runs under both 32- and 64-bit Windows. People have managed to run it on `ix86` Linux *via* WINE and on Mac OS X *via* WINE or CrossOver—but others have failed.

[32] `http://www.openbugs.info/`; this is Open Source but is *de facto* also restricted to `i386` Windows and Linux (although the `i386` version can be run on `x86_64` Linux). OpenBUGS is compiled by Oberon Microsystems' 'BlackBox' development system on 32-bit Windows: the `i386` Linux version is cross-compiled.

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

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[34] 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 interface packages `BRugs`, `R2OpenBUGS`, `R2WinBUGS` and `rjags`.

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`.

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

5 MCMC examples

This section sets the background for the examples of MCMC to be used in the practicals.

Binomial logistic regression

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.

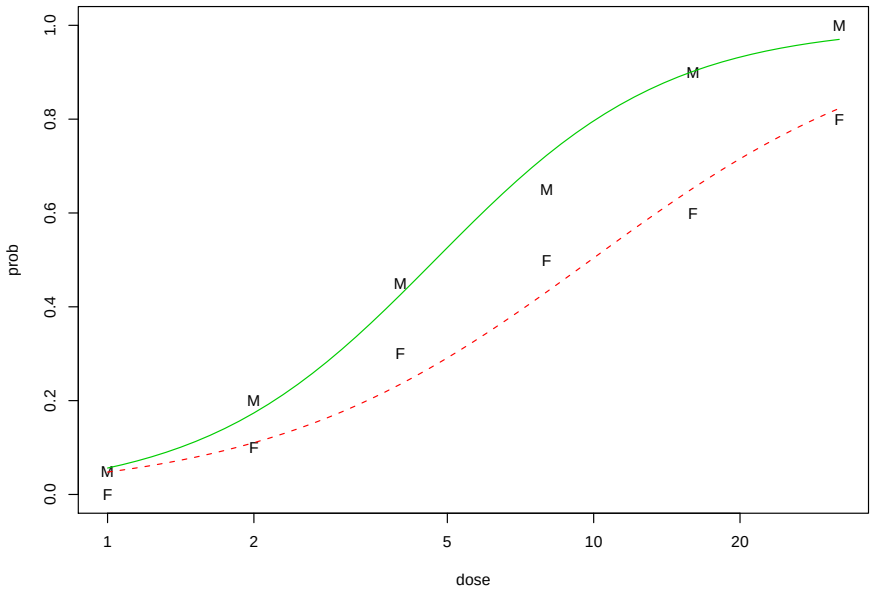

Figure 8: Probability of death of tobacco budworm moths *vs* dose (on log scale) of pyrethroid. The observed frequencies are marked by M (for male moths) and F (for female moths), together with fitted curves for separate logistic regressions.

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.

We start by using R package MCMCpack: this works with Bernoulli and not binomial data, so we must disaggregate the results. The default prior for β is an improper uniform prior, but others can be supplied – see ?MCMClogit.

```
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
fit <- MCMClogit(resp ~ sex*ldose, data = budworm, thin = 20)
summary(fit)
HPDinterval(fit)
```

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

This is illustrating the posterior distribution of the parameters, but that is not what we are interested in. With simulation-based inference it is trivial to transform the problem: simply transform the samples.

```
ld50F <- as.mcmc(2 ^ (-fit[,1]/fit[,3]) )
ld50M <- as.mcmc(2 ^ (-(fit[,1]+fit[,2])/(fit[,3] + fit[,4])) )
ld50 <- mcmc(cbind(M = ld50M, F = ld50F))
```

gives the posterior distribution of LD50 (for the original scale of the dose).

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 more accurately 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))
```

Even though this is a discontinuous non-linear transformation of the model fit, transforming the samples remains just a matter of writing a suitable R function.

Our second approach uses JAGS via `rjags`. We need to specify the JAGS model, which we will do in a file `budworm.jags`:

```
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(rjags)
inits <- list(list(alphaM = 0, betaM = 0, alphaF = 0, betaF = 0))
vars <- c("alphaM", "alphaF", "betaM", "betaF")
budworm.jags <- jags.model("budworm.jags", inits = inits, n.chains = 1,
                           n.adapt = 500)
budworm.sim <- coda.samples(budworm.jags, vars, n.iter = 10000)
summary(budworm.sim)
plot(budworm.sim)
effectiveSize(bd.sims)
```

with printout

```
terations = 501:10500
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
alphaF -3.0863 0.5485 0.005485        0.02216
alphaM -2.9135 0.5593 0.005593        0.01914
betaF   0.9342 0.1657 0.001657        0.00677
betaM   1.3014 0.2166 0.002166        0.00747

2. Quantiles for each variable:

          2.5%     25%    50%    75%  97.5%
alphaF -4.2345 -3.4368 -3.060 -2.717 -2.073
alphaM -4.0372 -3.2812 -2.890 -2.544 -1.856
betaF   0.6329  0.8232  0.928  1.041  1.279
betaM   0.8957  1.1553  1.295  1.441  1.744
```

```
> effectiveSize(bd.sims)
  alphaF   alphaM    betaF    betaM
624.3288 799.2533 645.4253 773.8282
```

We should explore other starting points, and will do so in the practical.

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.)

For a third approach, consider a random-walk Metropolis scheme. We start with the aggregated classical fit, and use its asymptotic distribution to suggest the (multivariate normal) distribution of the random-walk step.

```
w <- rep(20, 12)
fit <- glm(numdead/w ~ sex*ldose, weights = w, family = binomial)
X <- model.matrix(fit)
logpost <- function(beta)
    sum(dbinom(numdead, w, plogis(X %*% drop(beta)), log = TRUE))

library(LearnBayes)
scale <- 0.25
fit2 <- rwmetrop(logpost, list(var = vcov(fit), scale = scale),
                 coef(fit), m = 1000)
fit2$accept
sims <- as.mcmc(fit2$par) # make a coda object.
```

This assumes a flat prior, but priors are easily incorporated into `logpost`. The tuning constant `scale` re-scales the size of the step: this needs to be chosen to get a reasonable acceptable rate (often 10–50%) and large effective sample size—and will be in the practical.

Poisson change-point models

Consider the much-used data set of annual counts of British coal mining 'disasters' from 1851 to 1962.[35]

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 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 for (λ_1, λ_2). That is easy to do in the MCMC framework by re-weighting—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++, im-

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

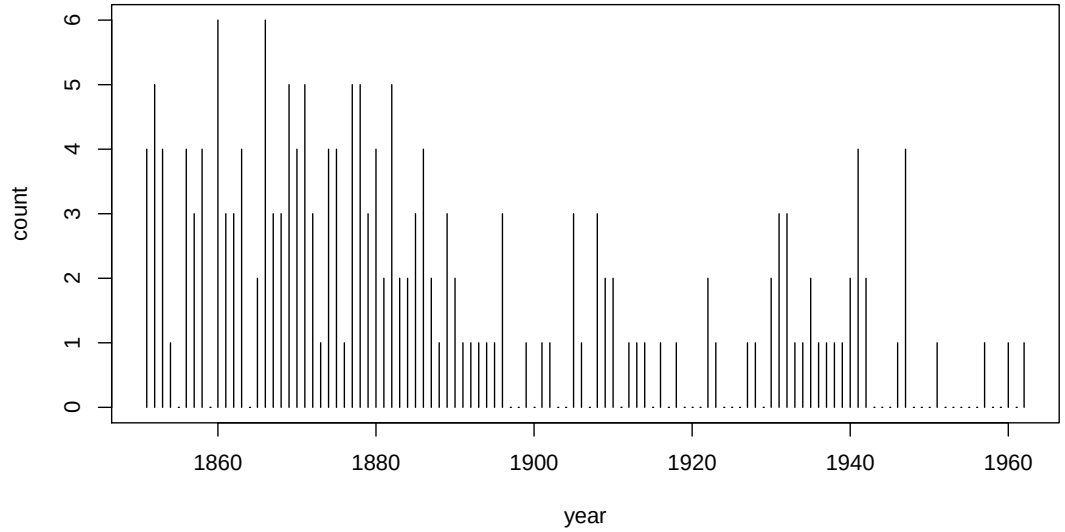

Figure 9: Number of explosions per year in British coal mines.

plementing 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 N = 112 counts.
library(MCMCpack)
fit <- MCMCpoissonChange(D ~ 1, m = 1, c0 = 1, d0 = 1,
                         burnin = 10000, mcmc = 10000,
                         marginal.likelihood = "Chib95") # bug workaround
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. `MCMCpack` produces samples ready for analysis by R package `coda`.

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

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

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`. The change-year is modelled as continuous, but we could also model it with a discete distribution.

We can then use `rjags` to ask JAGS to simulate from the posterior distribution of this model,

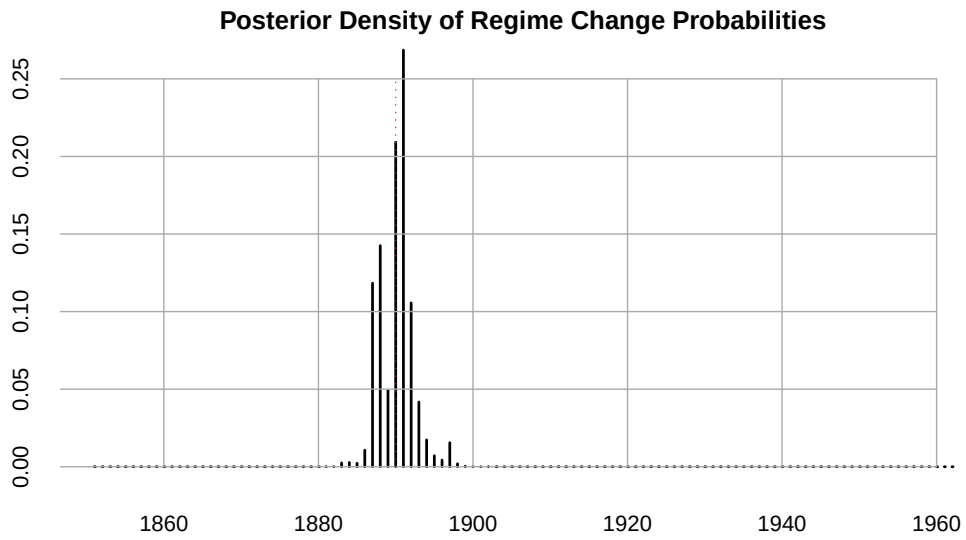

Figure 10: Posterior distribution for the year of change in disaster rate. The red dashed line marks the median.

by e.g.

```
library(rjags)
inits <- list(list(b = c(0, 0) , changeyear = 50),
              list(b = rnorm(2), changeyear = 30),
              list(b = rnorm(2), changeyear = 70))
cm.jags <- jags.model("coalmining.jags", inits = inits, n.chains = 3)
coalmining.sim <- coda.samples(cm.jags, c("changeyear","b"), n.iter = 5000)
```

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 list result.

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

```
> summary(coalmining.sim)

Iterations = 1001:6000
Thinning interval = 1
Number of chains = 3
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
b[1]        0.9711  0.2448 0.001999       0.003086
b[2]       -1.4779  0.4891 0.003993       0.023831
changeyear 58.7470 26.9659 0.220176       0.105376

...
```

Survival

Parametric survival models are not easily fitted in a Bayesian setting, and we consider fitting a Weibull accelerated life model to a subset of the Australian AIDS data of Venables & Ripley (2002, §13.5). To reduce computation time we consider only the 1116 patients from NSW and ACT (an enclave within NSW). To take account of the introduction of Zidovudine (AZT), time was run at half speed from July 1987.

```
library(MASS); library(survival)
Aidsp <- make.aidsp() # MASS ch13 script
fit <- survreg(Surv(survtime + 0.9, status) ~ T.categ + age,
               data = Aidsp, subset = (state == "NSW"))
summary(fit)
```

This model has 9 coefficients in the linear prediction, and one (σ) for the shape of the Weibull, which is estimated to be very close to exponential.

We can make use of an improved version of the code in Albert (2009):

```
library(LearnBayes)
weibullpost <- function(theta, data)
{  ## this version is 4x faster than the original
    p <- length(theta); shape <- theta[p]; beta <- theta[-p]
    lp <- data[, -(1:2), drop=FALSE] %*% beta
    zi <- (log(data[,1]) - lp)/shape
    Si <- -exp(zi)
    fi <- zi + Si - log(shape)
    ind <- data[,2] == 1
    lterm <- Si; lterm[ind] <- fi[ind]
    sum(lterm)
}
start <- c(coef(fit), shape = fit$scale)
mf <- model.frame(fit)
d <- cbind(mf[[1]], model.matrix(terms(fit), mf))
fit0 <- laplace(weibullpost, start, d)
```

This computed the posterior density for the parameters (β, σ) for a vague (improper) prior, and then finds the posterior mode (which is here the MLE). One might need to handle the constraint $\sigma > 0$ by for example a $\log$ transformation, but in this example σ is almost surely close to 1 so we use a simpler (and faster) version.

Simulation is then done by a random-walk Metropolis algorithm as before:

```
proposal <- list(var = fit0$var, scale = 0.5)
bf <- rwmetrop(weibullpost, proposal, fit0$mode, 1000, d)
bf$accept
res <- bf$par
colnames(res) <- names(start)
library(coda)
res <- as.mcmc(res)
plot(res)
acfplot(res)
effectiveSize(res)
```

This results in about 45% acceptance in the Metropolis step and apparently reasonable convergence well within 1000 steps. However, there is a high autocorrelation, so the effective

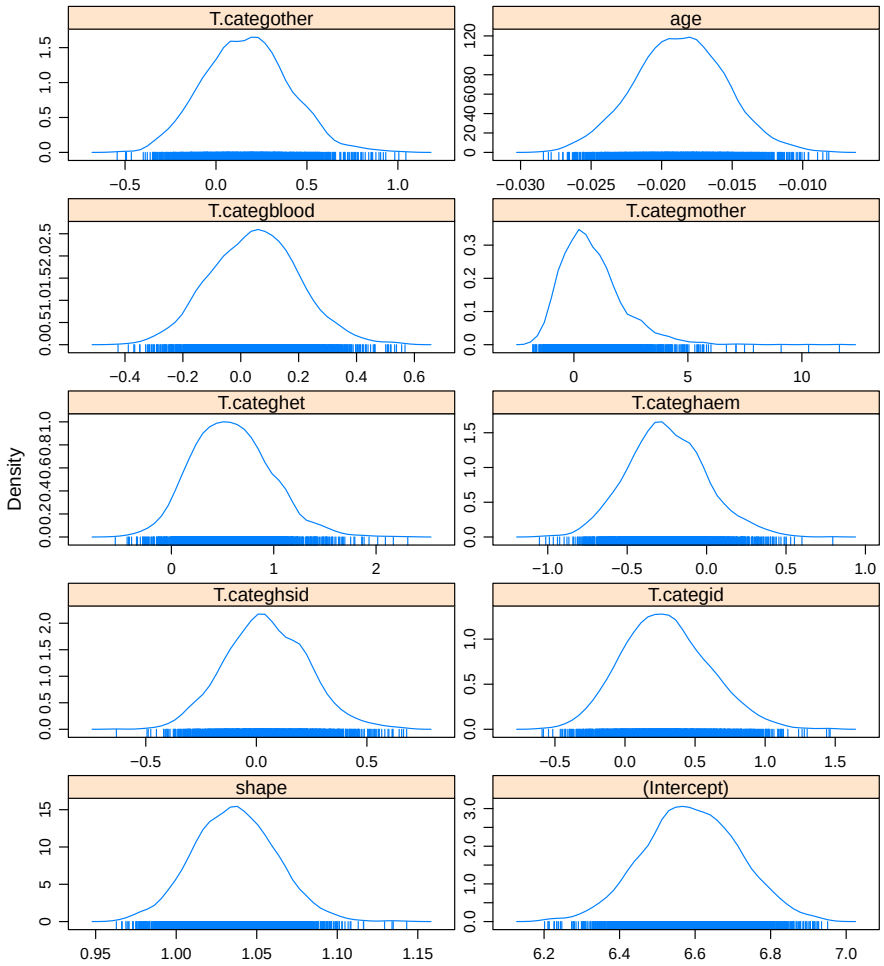

Figure 11: Kernel density estimates of the univariate posteriors from MCMC estimation of a Weibull survival model for the NSW/ACT AIDS data.

sample size is 10–30. More steps (and more time) are needed to get adequate coverage of the posterior density:

```
res <- rwmetrop(weibullpost, proposal, fit0$mode, 4e4, d)$par
colnames(res) <- names(start)
res <- window(as.mcmc(res), thin = 40)
effectiveSize(res)
acfplot(res)
densityplot(res)
summary(res)
```

Regression models

Albert (2009, § 9.3) considers a dataset on the number of puffin nesting burrows in 38 study areas with 4 explanatory variables.

Although the data are counts, Albert treated this as a regression problem, and so initially will we. For a Bayesian analysis *via* MCMC we can use

```
library(LearnBayes) # for the dataset
```

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

or as a JAGS model. A general way to set up regression models in `rjags` is

```
library(rjags)
X <- model.matrix(~ Grass + Soil + Angle + Distance, data = puffin)
inits <- list(list(beta = rep(0, 5), sigma = 1))
p.jags <- jags.model("puffin.jags", data = list(Nest = puffin$Nest, X = X),
                     inits = inits, n.chain = 1)
p.sims <- coda.samples(p.jags, c("beta", "sigma"), n.iter = 10000, thin = 10)
```

with model file `puffin.jags` given by

```
model{
    for(i in 1:38) { Nest[i] ~ dnorm(mu[i], sigma^-2) }
    mu <- X %*% beta
    for(i in 1:5) { beta[i] ~ dnorm(0, 0.01) }
    sigma ~ dunif(0, 10)
}
```

(See the scripts for how to get the samples labelled by the regression coefficient names and not just e.g `beta[3]`.)

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 = 1000, mcmc = 25000, thin = 25)
summary(Bpfit)
```

but we really need a better model. Package `MCMCpack` runs out of road here, but we can turn to JAGS.

Changing to a Poisson regression in JAGS is a simple change: the model file is `puffin2.jags`:

```
model{
    for(i in 1:38) { Nest[i] ~ dpois(exp(eta[i])) }
    eta <- X %*% beta
    for(i in 1:5) { beta[i] ~ dnorm(0, 0.01) }
}
```

run by

```
inits <- function() list(beta = c(2, rep(0, 4)))
p2.jags <- jags.model("puffin2.jags", data = list(Nest = puffin$Nest, X = X),
                      inits = inits, n.chain = 3)
p2.sims <- coda.samples(p2.jags, "beta", n.iter = 10000, thin = 25)
summary(p2.sims)
effectiveSize(p2.sims)
```

To change to a negative binomial regression, we need to know a bit more about how that is done classically (Venables & Ripley, 2002, §7.4) :

```
library(MASS)
nbfit <- glm.nb(Nest ~ Grass + Soil + Angle + Distance, data = puffin,
                maxit = 50)
summary(nbfit)
```

This uses a parametrization of the negative binomial that makes the model a GLM: this differs from `rnbinom` in R, which JAGS follows. So we can either use a model file like

```
model{
    for(i in 1:38) {
        Nest[i] ~ dnegbin(p[i], theta)
        lambda[i] <- exp(eta[i])
        p[i] <- theta/(theta + lambda[i])
    }
    eta <- X %*% beta
    for(i in 1:5) { beta[i] ~ dnorm(0, 0.01) }
    theta ~ dunif(1, 50) # a guess, the classical fit is about 9
}
```

or we can make use of the characterization of the negative binomial as a gamma-distributed mixture of Poissons as

```
model{
    for(i in 1:38) {
        Nest[i] ~ dpois(exp(eta[i]) * e[i]/theta)
        e[i] ~ dgamma(theta, 1)
    }
    eta <- X %*% beta
    for(i in 1:5) { beta[i] ~ dnorm(0, 0.01) }
    theta ~ dunif(1, 50)
}
```

We can handle zero-inflation (where the count is a mixture of two distributions, one of which is always zero) in a similar way.

Hierarchical 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 (in steps) 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 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.

Package `rjags` has the option

```
load.module("glm")
```

which makes further algorithms available for (generalized) linear models. At one point that made for appreciably faster convergence, but it seems at present often to be counter-productive.

6 Large datasets

What is 'large' about large datasets or the 2010s buzzword, *Big Data*?

- Many cases. This is perhaps the most common, for example
  - Oxford's library catalogue has 5 million items, the British Library's 13 million and the Library of Congress contains 21 million books and 141 milion items in total.
  - Insurance companies have records for all their customers, and all those they have offered quotes to, and they tend to share information with other companies. So there is a database with about 70 million records on US drivers.
  - Medical registries (in countries which have them) will have pretty much complete coverage of particular diseases (e.g. haemophilia) and typically these are of up to tens of thousands of subjects.
  - Inventories and sales records are often for many thousands of items.
  - Banks have records for every customer (several million in large countries) which they use to target their marketing.
  - Social media sites have lots of (mainly) small items. E.g. Twitter has 1–200 million 'tweets' per day of on average 40 bytes, and Facebook has around 1 billion posts per day.

  A variant on this is recording at high frequency for a long time, e.g. flight recorders, tick-by-tick financial trading.

- Many pieces of information per case. This tends to be rarer, but with genome-wide screening it is common to have thousands (and sometimes tens of thousands) of pieces of information for each of a modest number (perhaps 100) subjects.

  I used to work on group studies in fMRI.[36] These have at most tens of subjects, and maybe thousands of brain images each of a hundred thousand voxels for each subject.

  Remote sensing provides another example—complex images on few occasions. As does a project[37] to image (in 3D) mammalian brains at high enough resolution to see connections between neurons (of which there are around 10^{14} for a human). Each image of a 30nm slice from an electron microscope is about 20 billion pixels and it would need about 10^{11} slices for a human. So at a byte per pixel that is 2×10^{20} bytes of information: but the dataset will not be available (even for a mouse brain) in your lifetime.

- Many cases and many pieces of information on each. This is currently unusual, and the only one I have encountered is CRM.[38] You probably all have a 'loyalty' card—that is a means to bribe you to collect information about you. So the consortium owning the card know the buying patterns of every customer but also associations between items. You have probably used sites such as Amazon and been offered suggestions, maybe

  > People who purchased this book also purchased . . . '

[36] functional Magnetic Resonance Imaging.
[37] lead by Jeff Lichtman
[38] 'Customer Relationship Management'.

Perhaps national censuses come into this category: although full censuses usually have a modest number of questions, it is common to ask more of a, say, 10% sample and perhaps even follow those up yearly.

A typical dataset can be thought of as a rectangular table: the most common case is a 'long' table with the second and third bullet points corresponding to 'wide' and 'both long and wide' tables.

The largest dataset I have been involved with was one planned by my colleagues in genetics who are envisaging 20TB[39] of raw data. Note that Twitter is generating around 1TB/year.

And what is meant by 'large'? Unwin *et al.* (2006, Chapter 1) trace some history and quotes the following table (from Huber in 1992 and Wegman in 1995)

Size	Description	Bytes
Tiny	Can be written on a blackboard	10^2
Small	Fits on a few pages	10^4
Medium	Fills a floppy disk	10^6 (1MB)
Large	Fills a tape	10^8 (100MB)
Huge	Needs many tapes	10^{10} (10GB)
Monster		10^{12} (1TB)

Some more examples:

> Eighteen months ago, Li & Fung, a firm that manages supply chains for retailers, saw 100 gigabytes of information flow through its network each day. Now the amount has increased tenfold.
> (Economist, 2010 Feb 25)

> Tesco, a British retailer, collects 1.5 billion nuggets of data every month and uses them to adjust prices and promotions. Williams-Sonoma, an American retailer, uses its knowledge of its 60m customers (which includes such details as their income and the value of their houses) to produce different iterations of its catalogue.
> (Economist, 2011 May 26.)

> As a file on disk, the Netflix Prize data (a matrix of about 480,000 members' ratings for about 18,000 movies) was about 65Gb in size. But not every member rated every movie: there were only about 99 million ratings.
> (Bryan Lewis' blog, 2011 May 31)

Hardware considerations

Currently we are towards the end of the transition from '32-bit' to '64-bit' computing. These transitions happen every (quite a) few years—Windows went from 16-bit to 32-bit with Windows 95.[40]

What is it that was '32-bit'? Several things. Almost all current computers work with *bytes*, units of 8-bits. So '32-bits' refers to the way that bytes are addressed—by using a 32-bit

[39] 20 terabytes, about 20,000 GB or 2×10^{13} bytes or 40 entry-level hard discs.
[40] in 1995, with some 32-bit support in previous versions.

pointer we can address 2^{32} separate bytes, that is about 4 billion. What are these bytes? At least

- Virtual memory for a user process (the address space).
- Virtual memory for the operating system.
- Bytes in a file.

However, because programmers used signed integers which can hold numbers $-2^{31} \ldots 2^{31}-1$, most effective limits for 32-bit systems were 2GB.

File sizes of 2GB seemed large until recently, but with current disc sizes of around 500GB, 2GB files are no longer rare. Most 32-bit OSes have a means to support larger files, and R has made use of those facilities for some years.

These days 2GB of RAM is entry-level,[41] so it is reasonable to expect to use 2GB of memory in a single process. In fact 32-bit OSes limit per-process user address spaces to that or a bit more, and it is that address limit that pushed the move to 64-bit OSes. (Most CPUs currently on sale are 32/64-bit.) Multi-user servers with 128GB–1TB of RAM are no longer rare.

Until April 2013, R had a limit of $2^{31} - 1$ elements for a vector (which for numeric data takes 16GB of address space), but that was raised to 2^{52} in version 3.0.0 on 64-bit platforms.

To put this in some perspective, I moved to Oxford in 1990 and we equipped an MSc teaching lab with 20MHz 8MB Sun workstations running a 32-bit OS, and had about 600MB of disc space for all the graduate students. Our current teaching lab has 3.1GHz quad-core machines with 8GB RAM running 64-bit Windows 7 and we have about 2TB of primary user disc space.

DBMSs

Large amounts of data are not usually stored in simple files but in *databases*. Generally a *database* is thought of as the actual numeric or character data plus metadata.

Although some people use Excel spreadsheets as databases, professional-level uses of databases are *via* DataBase Management Systems (DBMS), which are designed to efficiently retrieve (and in some cases update) parts of the data. DBMSs lie behind a great deal of what we experience: when a call centre says 'I will just bring up your details',[42] what they are actually doing is using a DBMS to extract records from several tables in one or more databases. Also, when you ask for a page on many websites, it is retrieved from various tables in a DBMS.

DBMSs vary greatly in scale, from personal (such as Microsoft Access, MySQL) through department-level servers (Microsoft SQL Server, MySQL, PostgreSQL) to enterprise-wide (Oracle, DB2 and upscaled department-level systems).

Most of these systems work with SQL (Structured Query Language)[43] to access parts of the data. There is a series of ISO standards for SQL, but unfortunately most of the DBMS vendors have their own dialects.

[41] thanks to the memory usage of Windows Vista/7.
[42] usually followed by 'the system is rather slow today'.
[43] http://en.wikipedia.org/wiki/SQL.

It has been a long time coming, but it will become increasingly common for statisticians to be working with data stored in a DBMS. So some knowledge of SQL will become increasingly valuable.

There are several R packages which interface with DBMSs. The most universal is `RODBC`, which uses a standard called ODBC to talk to the DBMS and virtually all DBMSs have ODBC drivers. Then there are direct interfaces in packages `RMySQL`, `ROracle` and `RPostgreSQL`. A DBMS we have not yet mentioned is SQLite, which as its name suggests is a lightweight SQL engine that can be incorporated into other applications[44], and has been in packages `RSQLite` (and can also be accessed *via* ODBC). The Bioconductor project use SQLite to distribute much of their genomic metadata as binary databases.

Strategies for handling large datasets

The increase in volume of data available has been driven by automated collection, but computer power is growing faster than human activity. So in many fields we have already reached or are close to having all the data that will be relevant. For example, the motor insurance databases are as large as they are ever going to be, the medical registries have all current cases of rare diseases (and new cases are by definition rare), and so on.

So already some of the strategies needed in the past are no longer required.

Fifteen years ago, Bill Venables and I heard a talk at a conference about some new capabilities in a software package for 'large data' regressions, illustrated by a simulated regression problem with 10,000 cases. Over dinner we discussed if we had even seen such a problem in real life (no) and if we ever would (we thought not). The issue is that large regression problems are almost never homogeneous—they lump together data from different groups (e.g. different centres in a clinical trial). So the first strategy is

> *Divide the dataset into naturally occurring groups, analyse each group separately and do a meta-analysis.*

In a random regression problem each case adds the same amount of (Fisher) information, so collecting more cases reduces the variance of the estimator of the parameters at a known rate. As the regression model is false,[45] there will be a fixed amount of bias in its predictions irrespective of the sample size and for large enough sample sizes the bias will dominate the variance. So

> *With homogeneous datasets we can often achieve close to maximally accurate*[46] *results using a small sample of the dataset.*

It is hard to give detailed guidance as large homogeneous datasets are so rare, but it seems exceptional to need to do a regression on more than 1000 cases. Such problems may exist, but all those we have been offered as counter-examples have crumbled on close examination.

Do be careful in sampling heterogeneous datasets. We once had a motor insurance database of 700,000 cases to which we were fitting binomial and gamma generalized linear models.

[44] it is used by several OSes.

[45] apart from in a perfectly constructed simulation

[46] `http://en.wikipedia.org/wiki/Accuracy`.

Because the Fisher information per case is not uniform in such models, some observations are much more important than others, and we found that using a 10% random sample was giving much less good predictions than the whole dataset, and we needed to use a stratified sample. But that was in 2001 and the computers used had about 256MB of RAM, so sampling would no longer be needed. In fact a very important strategy is

Get a bigger computer, or even several of them.

As a student it may be hard to appreciate that the time of analysts (and particularly trained statisticians) is very valuable, and computers are cheap. If someone with a large dataset or a computer-intensive method does not have use of a multi-core computer with at least 32GB of RAM 24 hours/day, then their time is not being valued correctly.

If these strategies are not sufficient, we need to consider how to do the actual computations. In some statistical problems, only some summaries of the data are required to do the computation. It is tempting to think that this is the case in statistical models with low-dimensional sufficient statistics, but that is not in general the case as 'data' is not necessarily regarded as random in the model. Consider first a regression problem with response vector Y and an $n \times p$ data matrix X. The sufficient statistics are X^TY, but that is not enough information to find the parameter estimates. We can however find the parameter estimates from X^TX and X^TY, involving dimensions of size p but not n. However, to compute residuals, we need to go back to the data matrix X.

Now consider fitting a GLM. As you know, the commonest method is Iteratively (Re)Weighted Least Squares, which involves solving problems equivalent to

$$(X^TWX)b = X^TWY$$

where W is a diagonal matrix which varies for each iteration. We can use the summarization approach here, provided we are prepared to make multiple passes over the data. Note to the *cognoscenti*: we do not really solve these normal equations as stated, as that would be needlessly inaccurate. Rather one can make use of a row-wise QR decomposition, most often using Givens rotations.

This leads to another general strategy:

Consider algorithms needing multiple passes over the data.

These are almost inevitably slower, but can need fewer resources at any one time and so may be feasible. This is how programs from long ago like SAS[47] work, and there is an R package `biglm` which takes this approach for linear models and GLMs.

Another general strategy is to

Make use of multiple CPUs by parallelizing the computations.

Increasingly computers are coming with multiple CPU cores—even basic machines have two CPUs on a single chip and up-market servers have (in 2013) 4–10 cores/chip and two, four or more such chips. This trend is bound to continue for quite a while, and computer systems containing 256 or more CPU cores are now fairly common. Programming parallel computations is not easy, and the available hardware is beginning to run far ahead of available software. There are at least two fundamental problems

[47] still using a design from the 1960s and 70s.

- *Location of data.* Many statistical algorithms need repeated use of the same pieces of data and of data created from earlier computations (as we have seen e.g. for GLMs). Moving that data around between multiple computers is a bottleneck. Even where the CPUs are in the same computer or even on the same chip, moving data around can be an issue since modern CPUs get their speed by maintaining two or three levels of local cache memory.
- *Synchronization.* Somehow calculations need to be arranged so that CPUs are not waiting for too long for other CPUs to finish.

There are also loosely connected 'clusters' of separate computers, including using idle time on people's desktops.

The one relevant area where a lot of work has been done on parallelization is numerical linear algebra—experience is that gains with just two CPUs are often small, but with 4 or 8 or more it is possible to get a substantial speedup. *However*, it is also possible for the data-passing issues to dominate so that using multiple CPUs is several times *slower* than using just one. It is not easy to anticipate when this might happen, as the authors of the mixed-effect models packages for R such as `lme4` have found. Indeed, the reason that parallel computation is not currently enabled for vector and matrix arithmetic in R is the difficulty in determining when it is worthwhile (and on Mac OS X and Windows with the standard implementation, OpenMP, the problems need to be very large[48] for it to be worthwhile).

The most trivial form of parallel computation, using several CPUs for separate simulations, is particularly well suited to the methods of this module. This is sometimes known as *'embarrassingly parallel'* programming. For example, the $M = 4999$ case of the double bootstrap example took 120s on one core of a 16-core Intel Xeon Linux server. Running 16 processes in parallel *via* package `parallel` took 16s elapsed time, 193s CPU time. (The machine has hyperthreading, and using 32 processes took 13s elapsed.) There are worthwhile speedups even on a Windows laptop: my hyperthreaded dual-core Intel Core i7 took 26s using 4 processes for $M = 999$ *vs* 45s using one.

Note that it may become increasingly important to use multiple CPUs, as the conventional folk-wisdom version of Moore's Law ('computer power doubles every 18 months') has slowed down considerably. My mid-2007 home desktop was a 2.4GHz Intel Core 2 Duo[49] — its late-2012 replacement is a 3.1GHz Intel Core i7 with 4 real cores and 8 virtual cores, each of which is ca 1.7 times as fast.

These days 'supercomputers' are massively parallel. E.g. the current leader (June 2013, `http://www.top500.org/`) has 3,120,000 cores (in 12-core Intel Xeon CPUs and 60-core Xeon Phi coprocessors). And those are 'only' 2.2Ghz CPUs.

GPUs

Thanks to the insatiable appetite for more realistic computer games, the graphics cards in some computers contain one or several separate CPUs called GPUs, and nowadays these have a highly parallel architecture. Conversely, low-end computers and most laptops have simple

[48] Perhaps vector lengths of 100,000 or matrices of milions of elements.
[49] The Warwick lab machines are 2.2GHz Core 2 Duos.

on-board graphics. But that is changing as OSes make more use of visual effects, and most recent low-end CPU chips contain very capable GPUs, and even laptops often have separate GPUs.

There have been moves to harness such GPUs for more general computation, and recent generations have reasonable floating-point performance. The best known is Nvidia's *CUDA* and its *Tesla* 'personal supercomputer'. These have many hundreds of cores, which demands a very different way to think about organizing computations.

So far people are mainly trying to use GPUs with R to speed up linear algebra, e.g. specialized BLAS routines and packages `cudaBayesreg` and `gputools`. The real-world performance gains in linear algebra are not dramatic: one Tesla C2050 (which has 448 cores and shipped in May 2010) shows in http://www.microway.com/pdfs/TeslaC2050-Fermi-Performance.pdf only a few times the performance of a quad-core Intel Core i7 CPU.

For some specialist tasks GPUs can be alarmingly fast—for cracking passwords see
`http://www.pcpro.co.uk/blogs/2011/06/01/how-a-cheap-graphics-card-could-crack-your-password-in-under-a-second/`.

Visualization

Visualization of large datasets is an important topic – see Unwin *et al.* (2006) for the viewpoints of the Augsburg school. One dataset explored in their Chapter 11 is discussed at http://www.public.iastate.edu/~hofmann/infovis/ (with videos).

Sampling can help here, unless the aim is to spot outliers and other exceptional cases.

Large datasets in R

R is often criticized for needing too much memory as it holds its working data in memory (and often several copies of it). However the people who make this criticism most often have an axe to grind or a favourite package to promote. Note that

- The capacity of computers is growing faster than the size of datasets (in most fields).
- 'Memory' here means virtual memory not RAM and on OSes which manage VM well (not Windows) this can be a very effective way to store data on disc.
- We have already mentioned the strategy of holding data in a DBMS and using that to extract pieces as needed.

Nevertheless, some users are condemned to work with 32-bit Windows and there are several R packages which aim to make working with large datasets easier—although some are merely amateur implementations of virtual memory or DBMSs.

Package `biglm` uses the multiple pass approach for linear models and GLMs.

Packages `bigmemory`[50], `filehash` and `ff` all provide ways to store R objects on disc.

[50] not currently available for Windows, the one place it might be needed.

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.

Asanovic, K. and ten others (2006) The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

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

Bowman, A. and Azzalini, A. (1997) *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations*. Oxford: Oxford University Press.

Box, G. E. P., Hunter, W. G. and Hunter, J. S. (1978) *Statistics for Experimenters*. New York: John Wiley and Sons.

Buckland, S. T. (1984) Monte Carlo confidence intervals. *Biometrics* **40**, 811–817.

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.

Chernick, M. R. (2008) *Bootstrap Methods. A Practioner's Guide*. Second Edition. New York: Wiley.

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.

Cook, D. and Swayne, D. F. (2007) *Interactive and Dynamic Graphics for Data Analysis*. New York: Springer.

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

Davison, A. C. and Hinkley, D. V. (1997) *Bootstrap Methods and Their Application*. Cambridge: Cambridge University Press.

Davison, A. C., Hinkley, D. V. and Young, G. A. (2003) Recent developments in bootstrap methodology. *Statistical Science* **18**, 141–157.

Efron, B. (1982) *The Jackknife, the Bootstrap, and Other Resampling Plans*. Philadelphia: Society for Industrial and Applied Mathematics.

Efron, B. (1983) Estimating the error rate of a prediction rule. improvements on cross-validation. *Journal of the American Statistical Association* **78**, 316–331.

Efron, B. and Tibshirani, R. (1993) *An Introduction to the Bootstrap*. New York: Chapman & Hall.

Efron, B. and Tibshirani, R. (1997) Improvements on cross-validation: The .632 bootstrap method. *Journal of the American Statistical Association* **92**, 548–560.

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.

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.

Geyer, C. J. and Thompson, E. A. (1992) Constrained Markov chain maximum likelihood for dependent data (with discussion). *Journal of the Royal Statistical Society series B* **54**, 657–699.

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.

Hall, P. (1992) *The Bootstrap and Edgeworth Expansion*. Springer-Verlag.

Hall, P. (2003) A short pre-history of the bootstrap. *Statistical Science* **18**, 158–167.

Harrell, Jr., F. E. (2001) *Regression Modeling Strategies, with Applications to Linear Models, Logistic Regression and Survival Analysis*. New York: Springer-Verlag.

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.

Jöckel, K.-H. (1986) Finite sample properties and asymptotic efficiency of Monte Carlo tests. *Annals of Statistics* **14**, 336–347.

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.

Meng, X. L. and Wong, W. H. (1996) Simulating ratios of normalizing constants via a simple identity: a theoretical exploration. *Statistica Sinica* **6**, 831–860.

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.

Morgenthaler, S. and Tukey, J. W. eds (1991) *Configural Polysampling. A Route to Practical Robustness*. John Wiley and Sons.

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.

Politis, D. N., Romano, J. P. and Wolf, M. (1999) *Subsampling*. New York: Springer-Verlag.

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.

Rao, J. N. K. and Wu, C. F. J. (1988) Resampling inference with complex survey data. *Journal of the American Statistical Association* **83**, 231–241.

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. (1996) *Pattern Recognition and Neural Networks*. 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.

Roeder, K. (1990) Density estimation with confidence sets exemplified by superclusters and voids in galaxies. *Journal of the American Statistical Association* **85**, 617–624.

Shao, J. and Tu, D. (1995) *The Jackknife and the Bootstrap*. New York: Springer.

Simon, J. L. (1997) *Resampling: The New Statistics*. Second Edition. Resampling Stats.

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.

Staudte, R. G. and Sheather, S. J. (1990) *Robust Estimation and Testing*. New York: John Wiley and Sons.

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.

Unwin, A., Theus, M. and Hofmann, H. (2006) *Graphics of Large Datasets. Visualizing a Million*. Springer.

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 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 on Windows *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 or http://sourceforge.net/projects/mcmc-jags/files/Manuals/3.x.

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 puffins example

```
model{
    for(i in 1:38) { Nest[i] ~ dnorm(mu[i], sigma^-2) }
    mu <- X %*% beta
    for(i in 1:5) { beta[i] ~ dnorm(0, 0.01) }
    sigma ~ dunif(0, 10)
}
```

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[51] 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. 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[52] cycles are allowed.

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[53] can be used directly, so we could also use

```
model {
  for (year in 1:length(D)) {
       D[year] ~ dpois(exp(b[1] + step(year - changeyear) * b[2]))
```

[51] or *logical* nodes, especially in the BUGS manuals.
[52] e.g. for defining autoregressive and ordered priors
[53] `exp`, `ilogit`, `phi` and `icloglog`

```
    }
    for (j in 1:2) { b[j] ~ dnorm(0.0, 1.0E-6) }
    changeyear ~ dunif(1, length(D))
  }
```

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`[54] and `sort`. There are also some matrix functions such as `t`, `inverse` and `logdet`, plus in JAGS matrix multiplication (`%*%`).

Note that 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.

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 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.

[54] but with different arguments in OpenBUGS and JAGS

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, in some dialects, `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, 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), Jackman (2009), Lancaster (2004) and Ntzoufras (2009).

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(p.jags))
     RealSlicer ConjugateNormal ConjugateNormal ConjugateNormal
        "sigma"       "beta[5]"       "beta[4]"       "beta[3]"
ConjugateNormal ConjugateNormal
      "beta[2]"       "beta[1]"
> unlist(list.samplers(p2.jags))
RealSlicer RealSlicer RealSlicer RealSlicer RealSlicer
 "beta[5]"  "beta[4]"  "beta[3]"  "beta[2]"  "beta[1]"
```

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.

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).

Random number generators

Neither `rjags` nor `BRugs` use the R random number generators, rather those in JAGS and OpenBUGS 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.

B 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.

Tuesday Practical

Scripts for the exercises can be found at
http://www.stats.ox.ac.uk/~ripley/APTS2013/scripts
as well as JAGS model files for the second and third practicals.

Ex 1 We do something similar to figure 1, using 9999 random permutations.

```
library(MASS)
shoes
with(shoes, t.test(A, B, paired = TRUE))
d <- with(shoes, A - B)
t.test(d)
tperm <- replicate(9999, {
    a <- 2*rbinom(10, 1, 0.5) - 1
    t.test(a*d)$statistic
})

op <- par(mfrow = c(1, 2))
truehist(tperm, xlab = "diff", xlim = c(-5,5))
lines(density(tperm), lty = 2)
x <- seq(-5, 5, 0.1)
lines(x, dt(x,9))
plot(ecdf(tperm), xlim = c(-5,5), do.points = FALSE)
lines(x, pt(x,9), lty = 3)
par(op)
```

How can you use these data to perform a Monte Carlo test?

Ex 2 Dataset `cd4` in package `boot` provides 20 'before' and 'after' measurements of CD4 counts on HIV-positive patients. This exercise is based on Davison & Hinkley (1997, practicals 2.3 and 5.5).

(a) Read the help page `?cd4` for the background.

(b) Find a 90% confidence interval for the correlation (as in the preliminary material).

(c) Let us start by finding a 90% Monte-Carlo confidence interval based on bivariate normality. As ever, there is more than one way to do compute this! Here's an approach making use of the facilities of package `boot`:

```
library(MASS); library(boot)
cd4.rg <- function(data, mle) mvrnorm(nrow(data), mle$m, mle$v)

cd4.mle <- list(m = colMeans(cd4), v = var(cd4))
cd4.boot <- boot(cd4, corr, R = 999,
                 sim = "parametric", ran.gen = cd4.rg, mle = cd4.mle)
cd4.boot
boot.ci(cd4.boot, type = c("norm", "basic"), conf = 0.9)
boot.ci(cd4.boot, type = c("norm", "basic"), conf = 0.9, h = atanh, hinv = tanh)
```

The `"norm"` type uses a normal approximation with the mean and variance of the simulations, whereas `type = "basic"` is the 'Monte-Carlo confidence interval' in the sense of Buckland. This is a problem which is far from a location family, so we also consider the scale given by Fisher's transformation (as in the preliminary material).

(d) Now find a bootstrap confidence interval. `corr` is a function in package `boot` to compute weighted correlations.

```
cd4.boot <- boot(cd4, corr, stype = "w", R = 999)
cd4.boot
boot.ci(cd4.boot, conf = 0.9)
boot.ci(cd4.boot, conf = 0.9, h = atanh, hinv = tanh)
```

(e) The last part gave a warning about being unable to compute Studentized intervals. We can remedy that by

```
corr.fun <- function(d, w = rep(1, n))
{
    x <- d[, 1]; y <- d[, 2]
    n <- length(x); w <- w/sum(w)
    m1 <- sum(x*w); m2 <- sum(y*w)
    v1 <- sum(x^2*w) - m1^2; v2 <- sum(y^2*w) - m2^2
    rho <- (sum(x*y*w) - m1*m2)/sqrt(v1 * v2)
    i <- rep(1:n, round(n*w))
    us <- (x[i] - m1)/sqrt(v1); xs <- (y[i] - m2)/sqrt(v2)
    L <- us*xs - 0.5*rho*(us^2 + xs^2)
    c(rho, sum(L^2)/n^2)
}
cd4.boot <- boot(cd4, corr.fun, stype = "w", R = 999)
boot.ci(cd4.boot, type = "stud", conf = 0.9)
boot.ci(cd4.boot, type = "stud", conf = 0.9,
        h = atanh, hdot = function(r) 1/(1-r^2), hinv = tanh)
```

but you will need to consult Davison & Hinkley (1997, practical 2.3) to understand the calculations.

(f) We can use the double bootstrap to see how well we have done at getting 90% coverage. This will take a minute or so (depending on your machine): if you have less patience decrease `R` and `M` to 499. This makes use of two cores: you can use more (and more simulations) if you have them.

```
page(nested.corr) # a function in the 'boot' package
cd4m <- unname(as.matrix(cd4))
R <- 999; M <- 999

## on Mac or Linux use parallel = "multicore"
system.time(cd4.nest <- boot(cd4m, nested.corr, R = R, stype = "w",
                             t0 = corr(cd4), M = M,
                             parallel = "snow", ncpus = 2))

op <- par(pty = "s", xaxs = "i", yaxs = "i")
qqplot((1:R)/(R+1), cd4.nest$t[, 2], pch = ".", asp = 1,
       xlab = "nominal", ylab = "estimated")
abline(a = 0, b = 1, col = "grey")
par(op)
```

Now work out what corrections are needed to get a more accurate 90% interval.

Ex 3 This replicates part of figure 2.

(a) First we quickly get a rough idea of where the MLE is by plotting the RHS of (2) against θ (estimating the expectation by 100 simulations).

```
library(spatial)
towns <- ppinit("towns.dat")
tget <- function(x, r = 3.5) sum(dist(cbind(x$x, x$y)) < r)
t0 <- tget(towns)
c <- seq(0, 1, 0.2)
## res[1] = 0
res <- c(0, sapply(c[-1], function(c)
    mean(replicate(100, tget(Strauss(69, c = c, r = 3.5))))))
plot(c, res, type = "l", ylab = "E t")
abline(h = t0, col = "grey")
```

Here `tget` calculates the number of R-close pairs.

(b) This suggests zooming in to $(0.4, 0.6)$. We can do more runs: computers are fast enough these days. The scripts show how to do this in parallel.

```
c <- seq(0.4, 0.6, len = 6)
do_one <- function(c, R = 1000)
{
    z <- replicate(R, tget(Strauss(69, c = c, r = 3.5)))
    list(mean = mean(z), se = sd(z)/sqrt(R))
}
res <- lapply(c, do_one) # or a parallel version
means <- sapply(res, `[[`, 1)
sds <- sapply(res, `[[`, 2)
plot(c, means, type = "l", ylab = "E t")
abline(h = t0, col = "grey")
abline(lm(means ~ c))
arrows(c, means-1.96*sds, c, means+1.96*sds,
       angle = 90, code = 3, length = 0.1, xpd = TRUE)
```

(c) Now make use of these results to estimate the MLE of c, and give some indication of the inaccuracy. Choose some more simulations to do to get a more accurate result for the same amount of computation.

(d) How well can we do with polysampling? We do need to estimate the ratio of normalizing constants:

```
c0 <- 0.50
rs <- replicate(1000, tget(Strauss(69, c = c0, r = 3.5)))
c <- seq(0.4, 0.6, len = 20)
res <- numeric(length(c))
for(i in seq_along(c))
    res[i] <- mean(rs * (c[i]/c0)^rs)/mean((c[i]/c0)^rs)
points(c0, mean(rs), col = "blue"); lines(c, res, col = "blue")
```

To understand this, consider the importance sampling identity

$$E_c\, t(X) = E_{c_0}\, t(X) \frac{f(X;c)}{f(X;c_0)} = \frac{a(c)}{a(c_0)}\, E_{c_0}\, t(X) \left(\frac{c}{c_0}\right)^{t(X)}$$

For the first term we have

$$1/a(c) = \int c^{t(x)}\, dx = \int \frac{c^{t(x)}}{f(x;c_0)}\, f(x;c_0)\, dx = 1/a(c_0) E_{c_0} \left(\frac{c}{c_0}\right)^{t(X)}$$

so

$$E_c\, t(X) = E_{c_0}\, t(X) \left(\frac{c}{c_0}\right)^{t(X)} \Big/ E_{c_0} \left(\frac{c}{c_0}\right)^{t(X)}$$

(e) Now let us try stochastic approximation. The sequence (a_n) was chosen by trial-and-error.

```
R <- 1000
doit <- function(ave = FALSE, gam = 0.7) {
    res <- numeric(R)
    c <- runif(1, 0.4, 0.6) # initial guess.
    for(i in seq_len(R)) {
        a <- 0.5*(i+5)^-gam
        err <- tget(Strauss(69, c = c, r = 3.5))/t0  - 1
        c <- c - a*err
        res[i] <- c
    }
    if(ave) cumsum(res)/(1:R) else res
}
res <- doit()
plot(res, type = "l", ylim = c(0.4, 0.6))
for(i in 2:5) lines(doit(), col = i)
```

The scripts contain a parallel version, which will run faster but you will need to wait to see all the runs at once.

That was a näive form of Robbins–Munro. Clearly we can get a more accurate estimate by local averaging, and in what is known as Polyak–Ruppert averaging we can get optimum convergence rates, e.g.

```
res <- doit(TRUE)
plot(res, type = "l", ylim = c(0.4, 0.6))
for(i in 2:5) lines(doit(TRUE), col = i)
```

Experiment with other choices of $a_n = An^{-\gamma}$: what is a good choice will depend on whether averaging is done.

Again, the script illustrates a parallel version.

Ex 4 We continue the LD50 example from the lectures. Using `MCMCpack` we had

```
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))
summary(glm(resp ~ sex*ldose, family = binomial, data = budworm))

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, mcmc = 1e5, thin = 20)
summary(fit)
HPDinterval(fit)
```

Now we can explore the posterior distribution of LD50: this uses translucent points so that a build-up of colour indicates density.

```
## plot beta (slope) vs alpha (intercept) for females
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 no hint of negative slopes, or we would have to work harder (see the lecture notes).

Ex 5 Now we try out JAGS *via* `rjags`. We use the model file `budworm.jags` given in the notes.

```
library(rjags)
inits <- list(list(alphaM = 0, betaM = 0, alphaF = 0, betaF = 0))
## discard 500, record 10000
bd.jags <- jags.model("budworm.jags", inits = inits, n.chains = 1,
                      n.adapt = 500)
vars <- c("alphaM", "alphaF", "betaM", "betaF")
bd.sims <- coda.samples(bd.jags, vars, n.iter = 10000)
summary(bd.sims)
plot(bd.sims)
effectiveSize(bd.sims)
```

Now experiment with multiple starting points. To do so we set up a function for `inits`. If we give JAGS more than 4 chains we need to specify the RNG (and optionally the seeds)

```
inits <- function()
   list(alphaM = rnorm(1,0,10), betaM = rnorm(1),
        alphaF = rnorm(1,0,10), betaF = rnorm(1))
inits2 <- lapply(1:5, function(i) c(inits(),
                                    .RNG.name = "base::Mersenne-Twister",
                                    .RNG.seed = i))
bd.jags <- jags.model("budworm.jags", inits = inits2,
                      n.chains = 5, n.adapt = 500)
bd.sims <- coda.samples(bd.jags, vars, n.iter = 500)
summary(bd.sims)
plot(bd.sims)
gelman.plot(bd.sims)
densityplot(bd.sims) # and so on
```

Ex 6 For the random-walk Metropolis sampler we can use

```
w <- rep(20, 12)
fit <- glm(numdead/w ~ sex*ldose, weights = w, family = binomial)
X <- model.matrix(fit)
library(LearnBayes)
logpost <- function(beta)
    sum(dbinom(numdead, w, plogis(X %*% drop(beta)), log = TRUE))

scale <- 0.25
fit2 <- rwmetrop(logpost, list(var = vcov(fit), scale = scale),
                 coef(fit), m = 1000)
fit2$accept
sims <- as.mcmc(fit2$par) # make a coda object.
colnames(sims) <- names(coef(fit))
effectiveSize(sims)
```

Experiment with changing the tuning constant `scale`.

Ex 7 There is code in the lecture notes for two appoaches to the change-point problem for coal-mining disasters. Try them out. The data are[55]

```
D <- c(4,5,4,1,0,4,3,4,0,6,3,3,4,0,2,6,3,3,5,4,5,
       3,1,4,4,1,5,5,3,4,2,5,2,2,3,4,2,1,3,2,1,1,1,1,
       1,3,0,0,1,0,1,1,0,0,3,1,0,3,2,2,0,1,1,1,0,1,0,
       1,0,0,0,2,1,0,0,0,1,1,0,2,3,3,1,1,2,1,1,1,1,2,
       4,2,0,0,0,1,4,0,0,0,1,0,0,0,0,0,1,0,0,1,0,1)
```

Compare the speed of the two approaches.

[55] This version from Gamerman & Lopes (2006, p. 145) has the correct total, unlike some others in the literature.

Thursday Practical

Ex 8 Consider the Australian AIDS survival example sketched in the lecture notes.

Try the code there, and experiment with tuning `scale`. To set the problem up you will need

```
library(MASS)
make.aidsp <- function() {
  cutoff <- 10043 # 1987-07-01 with origin 1960-01-01
  btime <- pmin(cutoff, Aids2$death) - pmin(cutoff, Aids2$diag)
  atime <- pmax(cutoff, Aids2$death) - pmax(cutoff, Aids2$diag)
  survtime <- btime + 0.5*atime
  status <- as.numeric(Aids2$status)
  data.frame(survtime, status = status - 1, state = Aids2$state,
    T.categ = Aids2$T.categ, age = Aids2$age, sex = Aids2$sex)
}
Aidsp <- make.aidsp()
library(survival)
fit <- survreg(Surv(survtime + 0.9, status) ~ T.categ + age,
               data = Aidsp, subset = (state == "NSW"))
summary(fit)
```

This gives a classical accelerated-life survival model. The lecture notes have a Bayesian MCMC simulation for this model.

Use the `mcmc` function from package `coda` to convert the simulations to `coda` objects, and explore the diagnostics of the latter package.

Would there be any advantage in centring the explanatory variables in this problem? If you are unsure, try it.

Ex 9 We now turn to the puffin nesting data.

(a) Run a regression model using `MCMCpack`.

(b) Repeat using JAGS.

(c) Run a Poisson regression model using `MCMCpack`.

(d) Repeat using JAGS.

(e) Try both ways of fitting a negative binomial regression. Which runs faster? Which converges faster?