Computer Intensive Methods

Stuart Coles
Gareth Roberts
Søren Jarner

January 14, 2002

Contents

Chapter 1

Introduction

1.1 Scope of course

The term ‘computer intensive methods’ means different things to different people. It is also a dynamic subject: what requires intensive computing today may be solvable with a pocket calculator tomorrow. Not so long ago, the calculation of normal probabilities to reasonable accuracy would have required considerable CPU time. An initial classification of computer intensive methods as applied to statistics is the following:

1. Computers for graphical data exploration.
2. Computers for data modelling.
3. Computers for inference.

There is obviously some overlap in these three, but in this course I intend to focus mostly on the third of the above. That is, we shall aim at understanding computer techniques which require innovative algorithms to apply standard inferences. However, especially for Bayesian inference, and its associated numerical technique, MCMC, 2 and 3 have become so closely linked that it makes sense to consider the two together. Hence Chapter 3 will really consider modern approaches to Bayesian statistical modelling and their associated computational techniques. On the other hand, some techniques used for computationally intensive maximum likelihood calculations such as the bootstrap (Chapter 4) are intrinsically non-parametric, and as such the numerical techniques can be treated totally in isolation of any modelling considerations. I’ll exclude material dealing explicitly with modern regression approaches, such as kernel regression, lowess, etc..

I see two roles of this type of course. The first is to gain some understanding and knowledge of the techniques and tools which are available (so long as you’ve got the computing power). The second is that many of the techniques (if not all of them) are themselves clever applications or interpretations of the interplay between probability and statistics. So, understanding the principles behind the different algorithms can often lead to a better understanding of inference, probability and statistics generally. In short, the techniques are not just a means to an end. They have their own intrinsic value as statistical exercises.

This is not a course on computing itself. We won’t get into the details of programming itself. Furthermore, this is not a course which will deal with specialised statistical packages often used in statistical computing. All the examples will be handled using simple S-plus functions - far from the most efficient way of implementing the various techniques. It is important to recognise that high-dimensional complex problems do require more efficient programming (commonly in C or Fortran). However the emphasis of this course is to illustrate the various methods and their application on relatively simple examples. A basic familiarity with S-plus will be assumed.

1.2 Computers as inference machines

It is something of cliché to point out that computers have revolutionized all aspects of statistics. In the context of inference there have really been two substantial impacts: the first has been the freedom to make inferences without the catalogue of arbitrary (and often blatantly inappropriate) assumptions which standard techniques necessitate in order to obtain analytic solutions — Normality, linearity, independence etc. The second is the ability to apply standard type models to situations of greater data complexity — missing data, censored data, latent variable structures.

1.3 References

I've stolen quite heavily from the following books for this course:

- *Stochastic simulation*, B. Ripley.
- *An introduction to the bootstrap*, B. Efron and R. Tibshirani.
- *Tools for statistical inference*, M. Tanner.
- *Markov chain Monte Carlo in Practice*, W. Gilks S. Richardson and D. Spiegelhalter.

By sticking closely to specific texts it should be easy to follow up any techniques that you want to look at in greater depth. Within this course I'll be concentrating very much on developing the techniques themselves rather than elaborating the mathematical and statistical niceties. You're recommended to do this for yourselves if interested.

1.4 Acknowledgement

These notes are adapted from the excellent course notes produced by Stuart Coles for a previous version of this course.

Gareth Roberts, 2000

Only minor modifications have been made in this version of the notes.

Søren Jarner, 2001

Chapter 2

Simulation

2.1 Introduction

In this chapter we look at different techniques for simulating from distributions and stochastic processes. Unlike all subsequent chapters we won't look much at applications here, but suffice it to say the applications of simulation are as varied as the subject of statistics itself. In any situation where we have a statistical model, simulating from that model generates realizations which can be analyzed as a means of understanding the properties of that model. In subsequent chapters we will see also how simulation can be used as a basic ingredient for a variety of approaches to inference.

2.2 Issues in simulation

Whatever the application, the role of simulation is to generate data which have (to all intents and purposes) the statistical properties of some specified model. This generates two questions:

1. How to do it; and
2. How to do it efficiently.

To some extent, just doing it is the priority, since many applications are sufficiently fast for even inefficient routines to be acceptably quick. On the other hand, efficient design of simulation can add insight into the statistical model itself, in addition to CPU savings. We'll illustrate the idea simply with a well–known example.

2.3 Buffon's needle

We'll start with a simulation experiment which is intrinsically nothing to do with computers. Perhaps the most famous simulation experiment is Buffon's needle, designed to calculate (not very efficiently) an estimate of π. There's nothing very sophisticated about this experiment, but for me I really like the 'mystique' of being able to trick nature into giving us an estimate of π. There are also a number of ways the experiment can be improved on to give better estimates which will highlight the general principle of *designing* simulated experiments to achieve optimal accuracy in the sense of minimizing statistical variability.

Buffon's original experiment is as follows. Imagine a grid of horizontal parallel lines of spacing d, on which we randomly drop a needle of length l, with $l \leq d$. We repeat this experiment n times, and count R, the number of times the needle intersects a line. Denoting $\rho = l/d$ and $\phi = 1/\pi$, an estimate of ϕ is

$$\hat{\phi}_0 = \frac{\hat{p}}{2\rho}$$

where $\hat{p} = R/n$.

Thus, $\hat{\pi}_0 = 1/\hat{\phi}_0 = 2\rho/\hat{p}$ estimates π.

The rationale behind this is that if we let x be the distance from the centre of the needle to the lower grid line, and θ be the angle with the horizontal, then under the assumption of random needle throwing, we'd have $x \sim U[0,d]$ and $\theta \sim U[0,\pi]$. Thus

$$\begin{aligned} p &= \Pr(\text{needle intersects grid}) \\ &= \frac{1}{\pi}\int_0^{\pi} \Pr(\text{needle intersects } |\theta = \phi) d\phi \\ &= \frac{1}{\pi}\int_0^{\pi} (\frac{2}{d} \times \frac{l}{2}\sin\phi) d\phi \\ &= \frac{2l}{\pi d} = 2\rho\phi \end{aligned}$$

A natural question is how to optimise the relative sizes of l and d. To address this we need to consider the variability of the estimator $\hat{\phi}_0$. Now, $R \sim \mathrm{Bin}(n,p)$, so $\mathrm{Var}(\hat{p}) = p(1-p)/n$. Thus $\mathrm{Var}(\hat{\phi}_0) = 2\rho\phi(1-2\rho\phi)/4\rho^2 n = \phi^2(1/2\rho\phi - 1)/n$ which is minimized (subject to $\rho \leq 1$) when $\rho = 1$. That is, we should set $l = d$ to optimize efficiency.

Then, $\hat{\phi}_0 = \frac{\hat{p}}{2}$, with $\mathrm{Var}(\hat{\phi}_0) = (\phi/2 - \phi^2)/n$. It follows that $\mathrm{Var}\hat{\phi} \approx \pi^4 \mathrm{Var}(\hat{\phi}_0) \approx 5.63/n$.

Figure 2.1 gives 2 realisations of Buffon's experiment, based on 5000 simulations each. The S-Plus code to produce these is

```
buf<-function(n)
{
        ttt <- NULL
        ttt[1] <- 0
        x  <- runif(n)
        th <- runif(n, 0, pi)
        st <- sin(th)
        for(i in 1:n) {
                if(st[i] > x[i])
                        ttt[i + 1] <- ttt[i] + 1
                else ttt[i + 1] <- ttt[i]
        }
        ttt
        if(ttt[n + 1] > 0)
                plot((0:n)[ttt > 0], 2*(0:n)[ttt > 0]/ttt[ttt > 0], type = "l",
                        xlab = "number of simulations", ylab =
                        "proportion of hits")
        else print("no successes")
        abline(pi, 0)
}
```

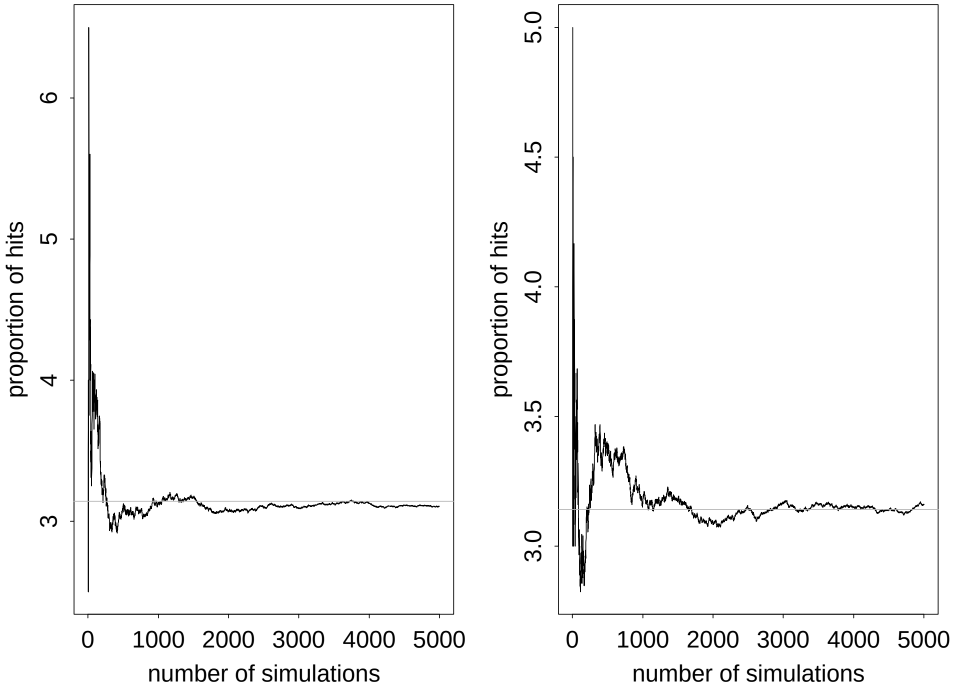

Figure 2.1: Two sequences of realisations of Buffon's experiment

Thus I've used the computer to simulate the physical simulations.

There are a catalogue of modifications which you can use which might (or might not) improve the efficiency of this experiment. These include:

1. Using a grid of rectangles or squares (which is best?) and basing estimate on the number of intersections with either or both horizontal or vertical lines.
2. Using a cross instead of a needle.
3. Using a needle of length longer than the grid separation.

So, just to re–iterate, the point is that simulation can be used to answer interesting problems, but that careful design may be needed to achieve even moderate efficiency.

2.4 Raw ingredients

The raw material for any simulation exercise is random digits: transformation or other types of manipulation can then be applied to build simulations of more complex distributions or systems. So, how can random digits be generated?

It should be recognised that any algorithmic attempt to mimic randomness is just that: a mimic. By definition, if the sequence generated is deterministic then it isn't random. Thus, the trick is to use algorithms which generate sequences of numbers which would pass all the tests of randomness (from the required distribution or process) despite their deterministic derivation. The most common technique is to use a *congruential generator*. This generates a sequence of integers via the algorithm

$$X_i = aX_{i-1} \pmod{M} \tag{2.1}$$

for suitable choices of a and M. Dividing this sequence by M gives a sequence U_i which are regarded as realisations from the Uniform $U[0,1]$ distribution. Ripley gives details of the number theoretic arguments which support this method, and gives illustrations of the problems which can arise by using inappropriate choices of a and M. We won't worry about this issue here, as any decent statistics package should have had its random number generator checked pretty thoroughly. The point worth remembering though is that computer generated random numbers aren't random at all, but that (hopefully) they look random enough for that not to matter.

In subsequent sections then, we'll take as axiomatic the fact that we can generate a sequence of numbers $U_1, U_2, \ldots, U_n$ which may be regarded as n independent realisations from the $U[0,1]$ distribution.

2.5 Simulating from specified distributions

In this section we look at ways of simulating data from a specified univariate distribution F, on the basis of a simulated sample $U_1, U_2, \ldots, U_n$ from the distribution $U[0,1]$.

2.5.1 Inversion

This is the simplest of all procedures, and is nothing more than a straightforward application of the probability integral transform: if $X \sim F$, then $F(X) \sim U[0,1]$, so by inversion if $U \sim U[0,1]$,

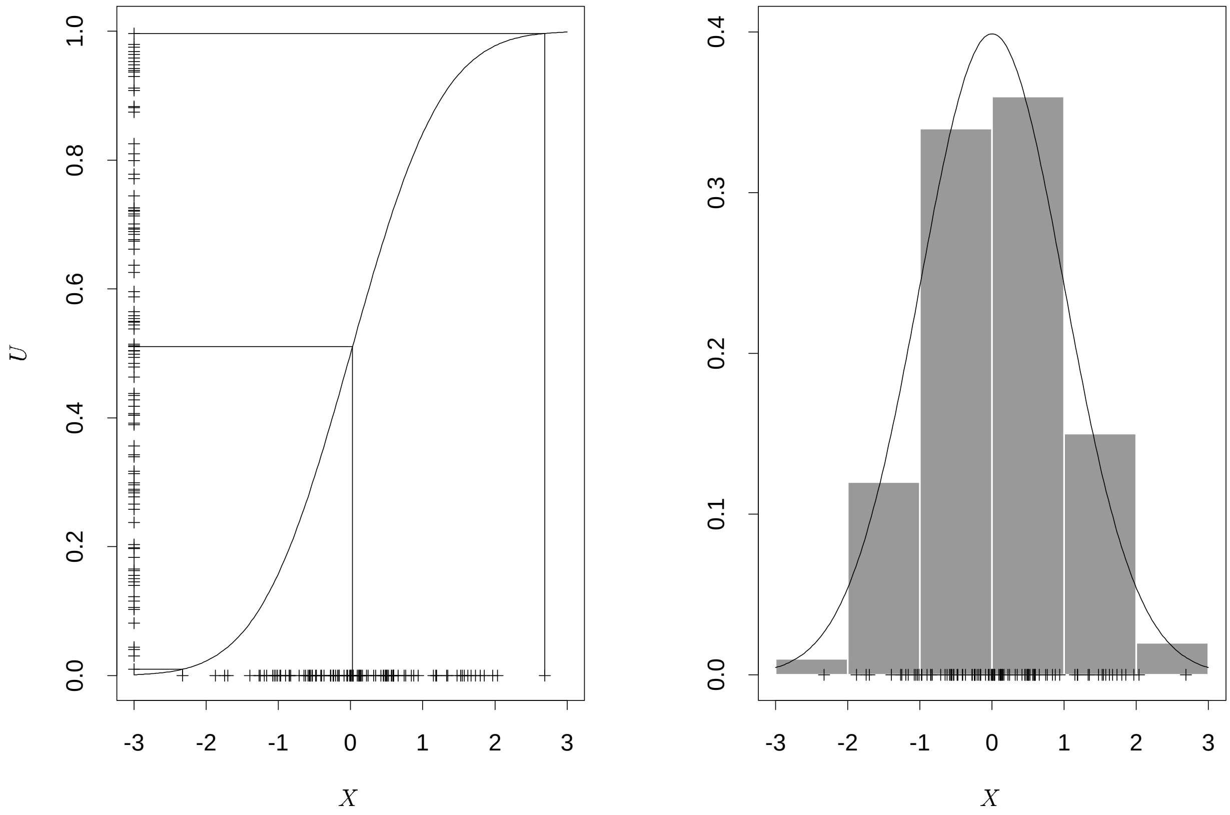

Figure 2.2: Left: The distribution function $F = \Phi$ for a standard normal distribution. The vertical crosses are 100 replicates of U and the horizontal crosses are the corresponding transformed values, $X = F^{-1}(U)$. Right: A histogram of the 100 replicates of X with the pdf for a standard normal distribution superimposed.

then $F^{-1}(U) \sim F$. Thus, defining $x_i = F^{-1}(u_i)$, generates a sequence of independent realisations from F. Figure 2.2 illustrates how this works.

This procedure is easily implemented in Splus with the following code:

```
dsim_function(n, inv.df)
{
        u <- runif(n)
        inv.df(u)
}
```

where `inv.df` is the user–supplied inverse distribution function F^{-1}. For example, to simulate from the exponential distribution we have $F(x) = 1 - \exp(-\lambda x)$, so $F^{-1}(u) = -\lambda^{-1}\log(1-u)$. Thus defining

```
inv.df_function(x,lam=1) -(log(1-x))/lam
```

we can simulate from the exponential distribution (unit exponential as default). Figure 2.3 shows a histogram of 1000 standard exponential variates simulated with this routine.

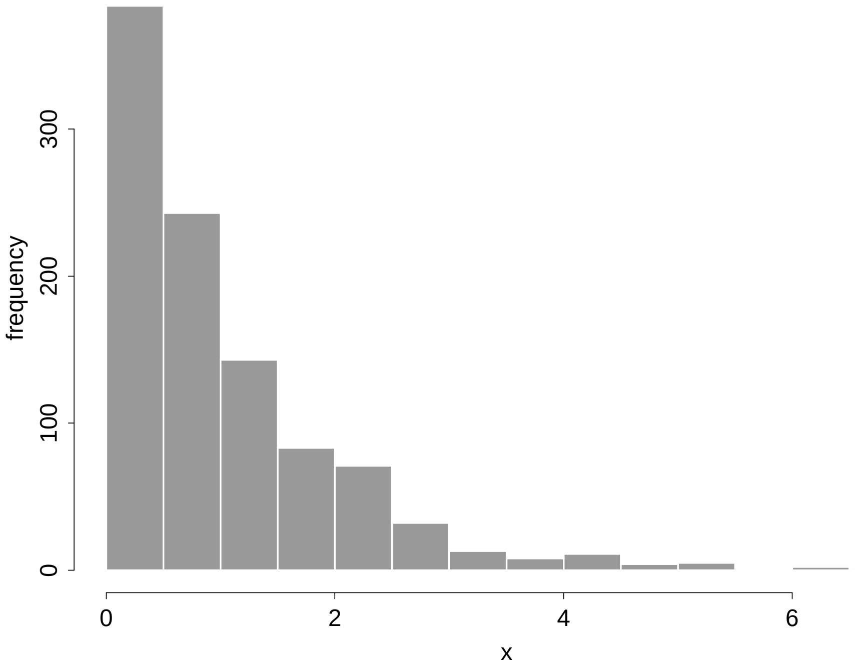

Figure 2.3: Histogram of 1000 simulated unit exponential variates

This procedure works equally well for discrete distributions, provided we interpret the inverse distribution function as

$$F^{-1}(u) = \min\{x|F(x) \geq u\} \tag{2.2}$$

The procedure then simply amounts to searching through a table of the distribution function. For example, the distribution function of the Poisson(2) distribution is

x	F(x)
0	0.1353353
1	0.4060058
2	0.6766764
3	0.8571235
4	0.9473470
5	0.9834364
6	0.9954662
7	0.9989033
8	0.9997626
9	0.9999535
10	0.9999917

so, we generate a sequence of standard uniforms $u_1, u_2, \ldots, u_n$ and for each u_i obtain a Poisson (2) variate x where $F(x-1) < u_i \leq F(x)$. So, for example, if $u_1 = 0.7352$ then $x_1 = 3$.

The limitation on the efficiency of this procedure is due to the necessity of searching through the table, and there are various schemes to optimize this aspect.

Returning to the continuous case, it may seem that the inversion method is sufficiently universal to be the only method required. In fact, there are many situations in which the inversion method is either (or both) complicated to program or excessively inefficient to run. The inversion method is only really useful if the inverse distribution function is easy to program and compute. This is not the case, for example, with the Normal distribution function for which the inverse distribution function, Φ^{-1}, is not available analytically and slow to evaluate numerically. To deal with such cases, we turn to a variety of alternative schemes.

2.5.2 Rejection sampling

The idea in rejection sampling is to simulate from one distribution which is easy to simulate from, but then to only accept that simulated value with some probability P. By choosing P correctly, we can ensure that the sequence of accepted simulated values are from the desired distribution. The technique is illustrated in Figure 2.4.

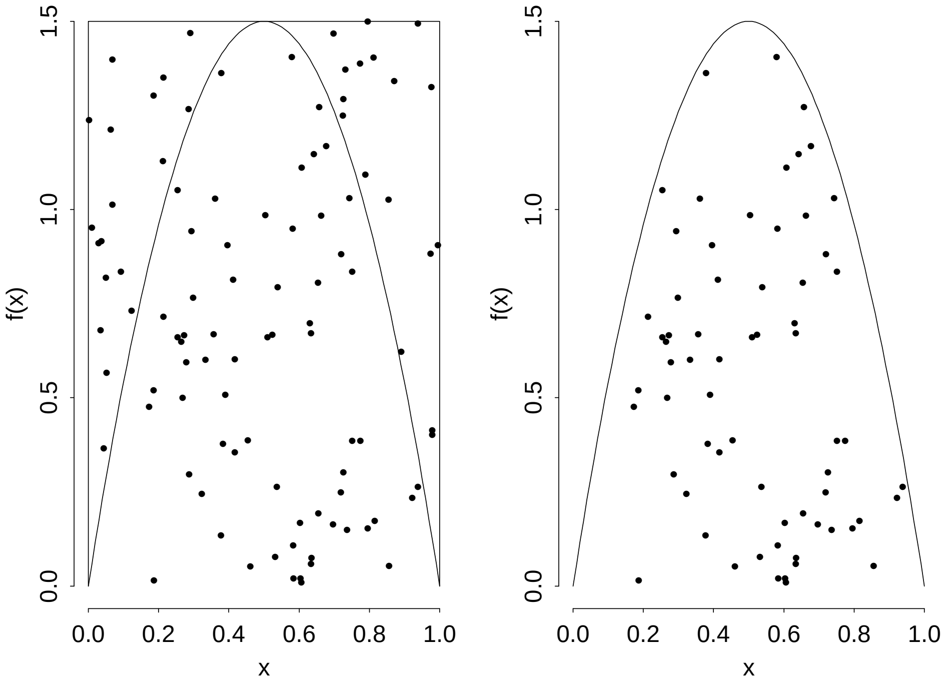

Figure 2.4: Simulation by rejection sampling

Suppose we wish to simulate a sample from the Beta(2,2) distribution. Inversion in this case would require non–linear solution of the inverse distribution function. Instead we bound the density function $f(x)$ by a rectangle, and simulate points (x_i, y_i) uniformly over the rectangle. We then reject those points which do not lie under $f(x)$. The x–coordinates of the remaining points will be a sample from $f(x)$. These are shown in Figure 2.5.

The efficiency of this method depends on how many points are rejected, which depends in turn

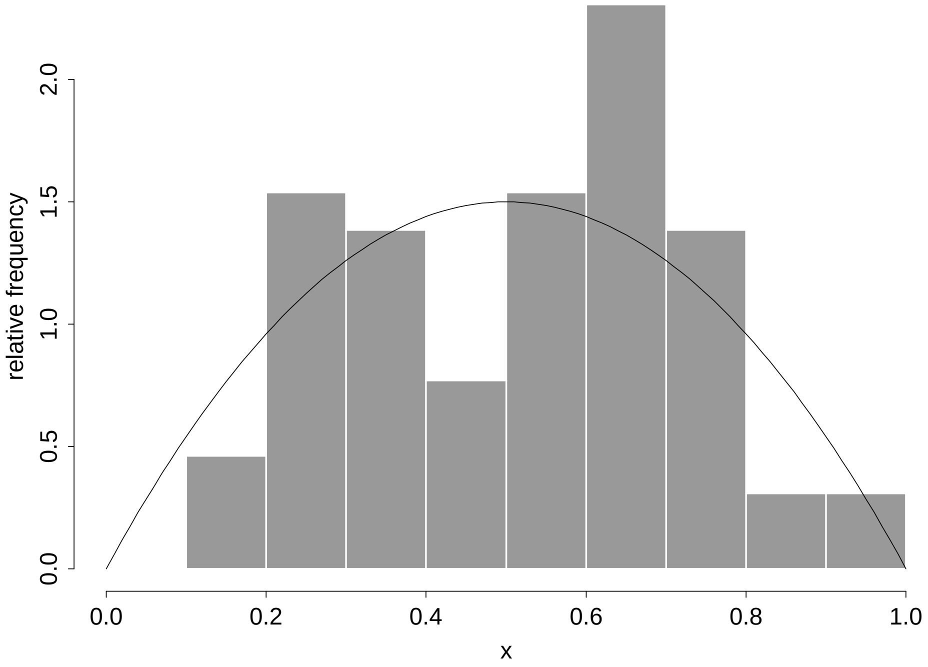

Figure 2.5: Histogram of simulated Betas

on how well $f(x)$ resembles the bounding rectangle. To improve the efficiency of the procedure, and to allow for situations where $f(x)$ may be unbounded, the technique can be modified to permit the bounding function to take any form $Kg(x)$, where $g(x)$ is the density of a distribution from which it is easy to simulate. Then the algorithm takes the form:

Algorithm 2.1

1. *Simulate x^* from $g(x)$.*
2. *Simulate y^* from $U(0, Kg(x^*))$.*
3. *Accept x^* if $y^* \leq f(x^*)$.*
4. *Continue.*

The reason this works is as follows: let X denote a random variable from any distribution $g(x)$, such that $f(x) \leq Kg(x)$ $\forall x$ for some value K. Let $h(x)$ be the probability that x is accepted: $h(x) = f(x)/Kg(x)$. Then

$$\Pr(X \leq x \text{ and } X \text{ accepted}) = \int_{-\infty}^{x} h(y)g(y)dy \tag{2.3}$$

and

$$\Pr(X \text{ accepted}) = \int_{-\infty}^{\infty} h(y)g(y)dy \tag{2.4}$$

so

$$\begin{aligned} \Pr(X \leq x | X \text{ accepted}) &= \frac{\int_{-\infty}^{x} h(y)g(y)dy}{\int_{-\infty}^{\infty} h(y)g(y)dy} \\ &= \frac{\int_{-\infty}^{x} f(y)dy}{\int_{-\infty}^{\infty} f(y)dy} \end{aligned}$$

so that the accepted values do indeed have pdf f. Furthermore, $\Pr(X \text{ accepted}) = \int gh = 1/K$.

Note, in particular, that the normalizing in the denominator means that f need only be known up to proportionality in order for this technique to work (though this is obvious geometrically). The efficiency of the procedure depends on the quality of the agreement between f and the bounding envelope Kg since if a large value of K is necessary, then the acceptance probability is low, so that large numbers of simulations are needed to achieve a required sample size.

As an example, consider the distribution with density

$$f(x) \propto x^2 e^{-x}; \quad 0 \leq x \leq 1 \tag{2.5}$$

a truncated gamma distribution. Then, since $f(x) \leq e^{-x}$ everywhere, we can set $g(x) = \exp(-x)$ and so simulate from an exponential distribution, rejecting according to the above algorithm. Figure 2.6 shows both $f(x)$ and $g(x)$. Clearly in this case the envelope is very poor so the routine is highly inefficient (though statistically correct).

Applying this to generate a sample of 100 data using the following code

```
 rej.sim_function(n)
{
        r <- NULL
```

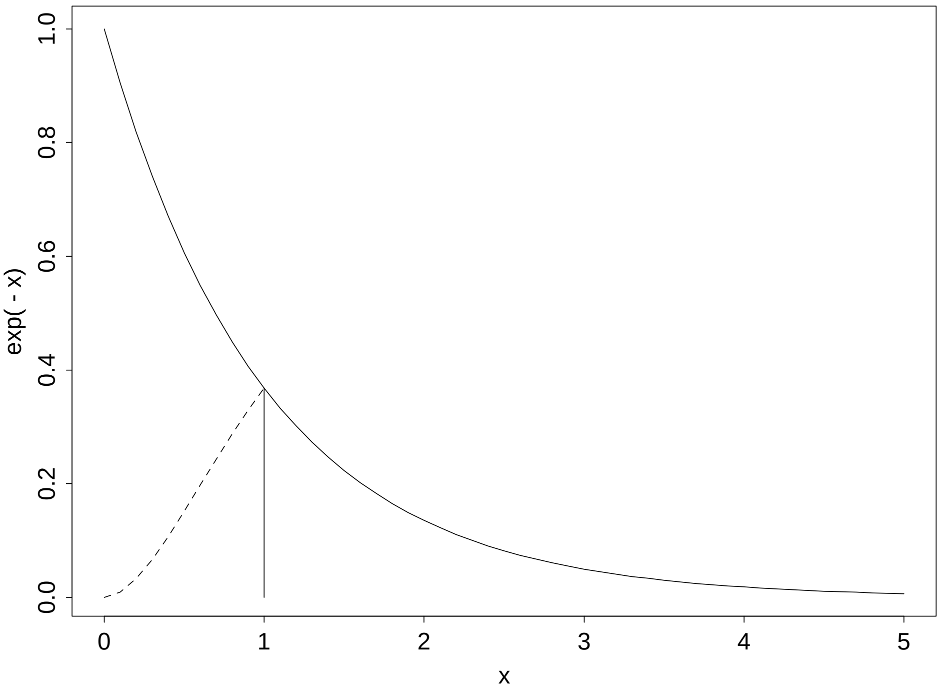

Figure 2.6: Scaled density and envelope

```
        for(i in 1:n) {
                t <- -1
                while(t < 0) {
                        x <- rexp(1, 1)
                        y <- runif(1, 0, exp( - x))
                        if(x > 1)
                                t <-  - y
                        else t <- x^2 * exp( - x) - y
                }
                r[i] <- x
        }
        r
}
```

gave the histogram in Figure 2.7.

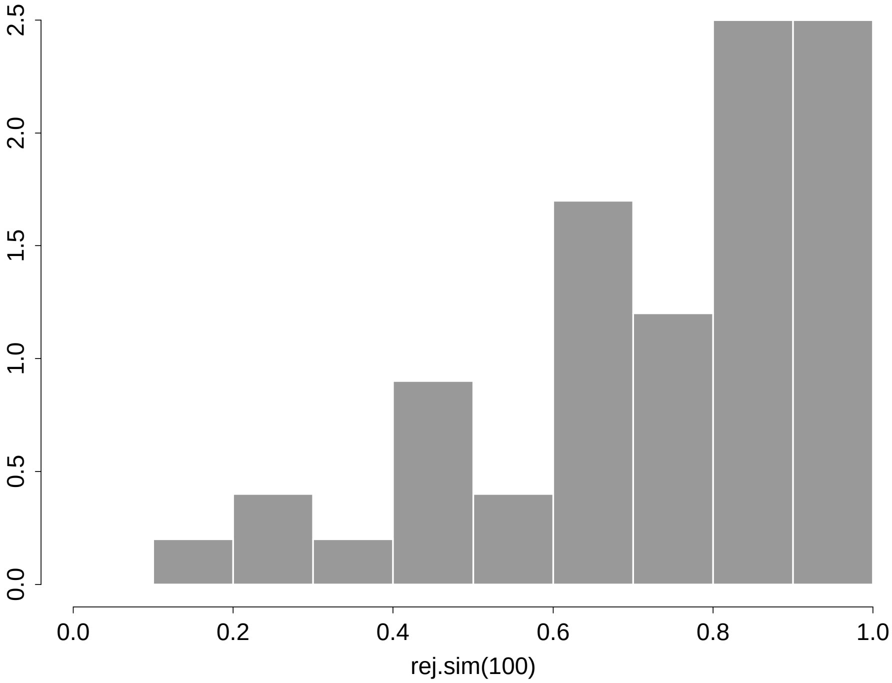

Figure 2.7: Histogram of simulated data

2.5.3 Ratio of uniforms

An adaptation of the rejection algorithm which works well for many distributions is the ratio of uniforms method. Here a pair of independent uniforms are simulated and the ratio accepted as a simulant from the required distribution according to a rejection scheme.

The basis of the technique is the following argument. Suppose h is a non–negative function such that $\int h < \infty$ and we let $C_h = \{(u, v) : 0 \leq u \leq \sqrt{h(v/u)}\}$. Then if (U, V) is uniformly distributed over C_h then $X = V/U$ has pdf $h/\int h$.

So, to simulate from a density proportional to h, we simulate uniformly over the region C_h, and take ratios of coordinates. In practice, C_h may be complicated in form, so the only practical solution is to bound it with a rectangle (if possible), simulate within the rectangle (by a pair of uniforms), and apply rejection: hence, *ratio of uniforms*.

The reason this works is as follows. Let Δ_h be the area of C_h. Then on changing variables $(u, v) \rightarrow (u, x)$, where $x = v/u$,

$$\Delta_h = \int\int_{C_h} dudv \quad = \int\int_0^{\sqrt{h(x)}} u \; dudx \quad = \int \frac{1}{2} h(x) dx \tag{2.6}$$

Because of the uniformity of (U, V) over C_h, (U, V) has pdf $1/\Delta_h$ so that on transformation, (U, X) has pdf u/Δ_h, and integrating out U gives the marginal pdf of X as:

$$\Delta_h^{-1} \int_0^{\sqrt{h(x)}} u \; du \quad = \quad h(x)/\{2\Delta_h\} \quad = \quad h(x)/\int h(x)dx \tag{2.7}$$

Thus V/U has pdf proportional to h. Again, a key property of this method is that h is only required to be specified up to proportionality.

As discussed above, this is only useful if we can generate uniformly over C_h, which is most likely to be achieved by simulating uniformly within a rectangle $[0, a] \times [b_-, b_+]$ which contains C_h (provided such a rectangle exists). If it does, we have the following algorithm.

Algorithm 2.2

```
1. Simulate independent $U \sim U[0, a]$, $V \sim U[b_-, b_+]$.
2. If $(U, V) \in C_h$, accept $X = V/U$, otherwise repeat.
3. Continue.
```

As an example, consider the Cauchy distribution with density

$$h(x) \propto \frac{1}{1 + x^2} \tag{2.8}$$

Then $C_h = \{(u, v) : 0 \leq u \leq \sqrt{h(v/u)}\} = \{(u, v) : 0 \leq u, u^2 + v^2 \leq 1\}$, a semicircle. Hence we can take $[0, a] \times [b_-, b_+] = [0, 1] \times [-1, 1]$ and get the algorithm

Algorithm 2.3

```
1. Simulate independent $U \sim U[0, 1]$, $V \sim U[-1, 1]$.
2. If $u^2 + v^2 \leq 1$ accept $x = v/u$, otherwise repeat.
3. Continue.
```

This can be implemented with the Splus code

```
ru.sim_function(n)
{
        r <- NULL
        for(i in 1:n) {
                t <- 1
                while(t > 0) {
                        u <- runif(1, 0, 1)
                        v <- runif(1, -1, 1)
                        t <- u^2 + v^2 - 1
                }
                r[i] <- v/u
        }
        r
}
```

and a histogram of 1000 simulated values is given in Figure 2.8 (note the unusual scale (!) because the heaviness of the Cauchy tail causes one or two simulated values to be extreme relative to the rest of the data).

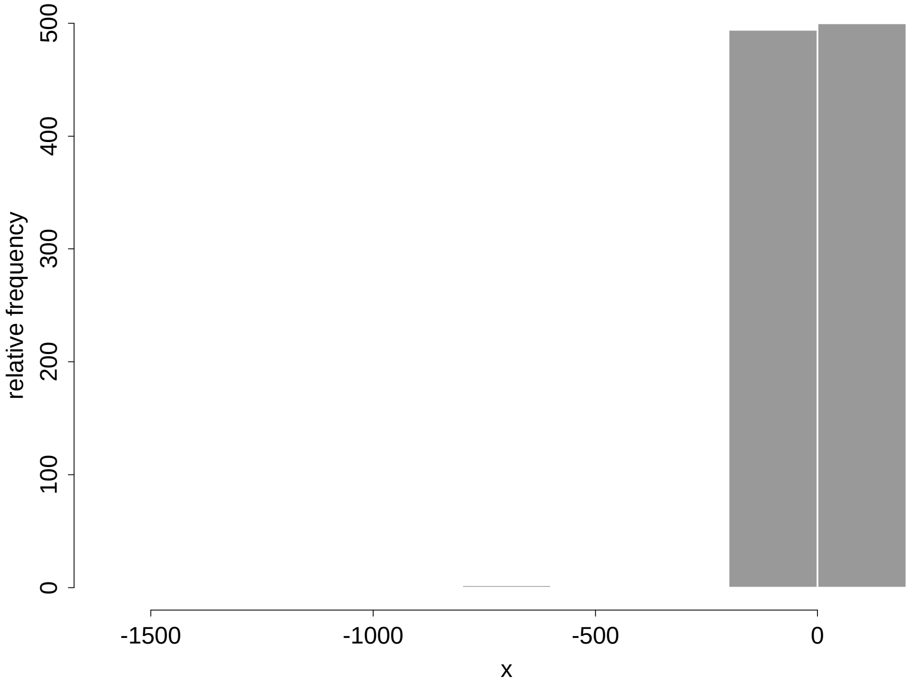

Figure 2.8: Histogram of simulated Cauchys

A number of modifications have been proposed to improve on the efficiency of this procedure, which amount to rescaling and locating distributions before applying the method.

Another method for improving the efficiency is by a process known as 'squeezing' or 'pre–testing'. This applies to both the rejection and ratio of uniform methods. The point is that, in the ratio of uniforms method for example, the slowest part of the algorithm can be the check of whether $(u, v) \in C_h$ or not. However, there may be simpler regions C_1 and C_2 such that $C_1 \subset C_h \subset C_2$, so that if (u, v) is found to lie inside C_1 or outside C_2 then we immediately know whether it lies inside C_h or not.

2.6 Monte Carlo integration

In one form or another, the reason for simulation can often be formulated as an integral. This is obviously the case for expectations: $E(X) = \int xf(x)dx$, so if we have a sample $x_1, x_2, \dots, x_n$ from the distribution of X, then we can approximate the theoretical mean by the sample mean to obtain the approximation:

$$E(X) \approx n^{-1} \sum_{i=1}^{n} x_i \tag{2.9}$$

But this argument can be generalized. Suppose we wish to calculate

$$\theta = \int \phi(x) f(x) dx \tag{2.10}$$

which is of course $E(\phi(X))$, where expectation is with respect to the distribution f. Then if $x_1, x_2, \dots, x_n$ is a sample from this distribution,

$$\hat{\theta} = n^{-1} \sum_{i=1}^{n} \phi(x_i) \tag{2.11}$$

is an unbiased estimate of θ. Again, this is simply a sample mean estimating a theoretical mean. This approach is remarkably easy to use, even in high dimensions. The cost for this simplicity is that the variance is high.

As an example of this, suppose we wish to calculate $P(X < 1, Y < 1)$ where (X, Y) are bivariate Standard Normal with correlation 0.5. This can be written as

$$\int I_A(x, y) f(x, y) dxdy \tag{2.12}$$

where f is the bivariate normal density, and I_A is the indicator function on $A = \{(x, y) : x < 1, y < 1\}$. Thus, provided we can simulate from the bivariate normal, we can estimate this probability as

$$n^{-1} \sum_{i=1}^{n} I_A(x_i, y_i) \tag{2.13}$$

which is simply the proportion of simulated points falling in A. There are various approaches to simulating bivariate Normals; conceptually easiest is to simulate each pair (x, y) by first simulating x, and then simulating from the conditional distribution of $Y|X$ which is also normal, but with modified mean and variance (essentially derived from standard regression). Splus code to achieve this is

```
bvsim_function(n, me, sd, ro)
{
        x <- rnorm(n, me[1], sd[1])
        y <- rnorm(n, me[2] + (ro * sd[2])/sd[1]
                * (x - me[1]), sd[2] * sqrt(1 -  ro * ro))
        cbind(x, y)
}
```

Hence, to obtain an estimate of the required probability on the basis of, say, 1000 simulations, we simply need

```
y_bvsim(1000,c(0,0),c(1,1),0.5)
```

and then

```
sum(y[,1]<1&y[,2]<1)/1000
```

I got 0.763 doing this. A scatterplot of the simulated values is given in Figure 2.9.

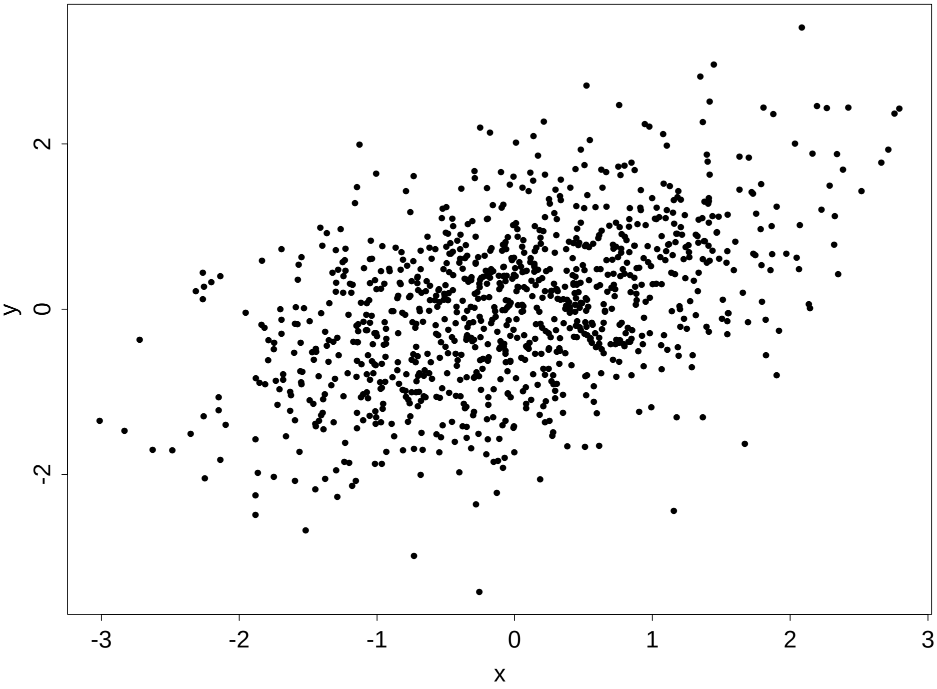

Figure 2.9: Simulated bivariate normals

2.6.1 Variance reduction

A number of techniques are available for improving the precision of Monte–Carlo integration. We'll look at one of these in detail, and describe the idea behind another two.

Importance sampling

We want to calculate

$$\theta = \int \phi(x) f(x) dx \tag{2.14}$$

which can be re–written

$$\theta = \int \psi(x)g(x)dx \tag{2.15}$$

where $\psi(x) = \phi(x)f(x)/g(x)$. Hence, if we obtain a sample $x_1, x_2, \ldots, x_n$ from the distribution of g, then we can estimate the integral by the unbiased estimator

$$\hat{\theta}_g = n^{-1} \sum_{i=1}^{n} \psi(x_i) \tag{2.16}$$

for which the variance is

$$\text{Var}(\hat{\theta}_g) = n^{-1} \int \{\psi(x) - \theta\}^2 g(x)dx \tag{2.17}$$

This variance can be very low, much lower than the variance of $\hat{\theta}$, if g can be chosen so as to make ψ nearly constant. Essentially what is happening is that the simulations are being concentrated in the areas where there is greatest variation in the integrand, so that the informativeness of each simulated value is greatest.

This example taken from Ripley illustrates the idea. Suppose we want to estimate the probability $P(X > 2)$, where X follows a Cauchy distribution with density function

$$f(x) = \frac{1}{\pi(1 + x^2)} \tag{2.18}$$

so we require the integral

$$\int I_A(x)f(x)dx \tag{2.19}$$

where $A = \{x : x > 2\}$. We could simulate from the Cauchy directly and apply (2.11), but the variance of this estimator is substantial. (As with the bivariate Normal example, the estimator is the empirical proportion of exceedances; exceedances are rare, so the variance is large compared to its mean).

Alternatively, we observe that for large x, $f(x)$ is similar in behaviour to the density $g(x) = 2/x^2$ on $x > 2$. By inversion, we can simulate from g by letting $x_i = 2/u_i$ where $u_i \sim U[0, 1]$. Thus, our estimator becomes:

$$\hat{\theta}_g = n^{-1} \sum_{i=1}^{n} \frac{x_i^2}{2\pi(1 + x_i^2)} \tag{2.20}$$

where $x_i = 2/u_i$. Implementing this with the Splus function

```
i.s
function(n)
{
        x <- 2/runif(n)
        psi <- x^2/(2 * pi * (1 + x^2))
        mean(psi)
}
```

gave the estimate $\hat{\theta} = .1478$. The exact value is $.5 - \pi^{-1} \tan 2 = .1476$. In Figure 2.10 the convergence of this sample mean to the true value is demonstrated as a function of n.

For comparison, in Figure 2.11, we show how this compares with a sequence of estimators based on the sample mean when simulating directly from a Cauchy distribution. Clearly, the reduction in variability is substantial.

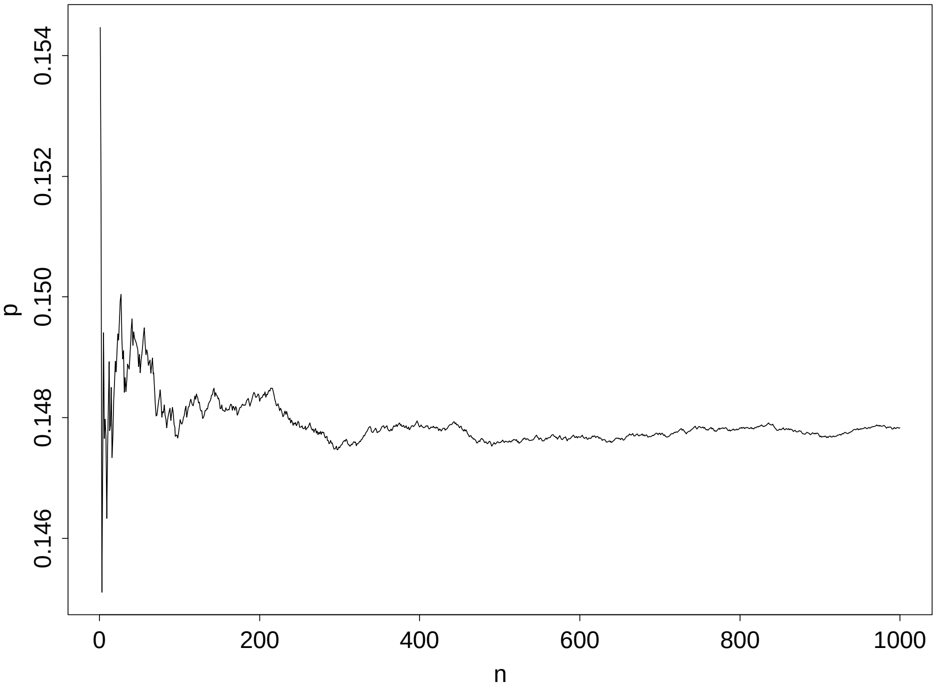

Figure 2.10: Convergence of importance sampled mean

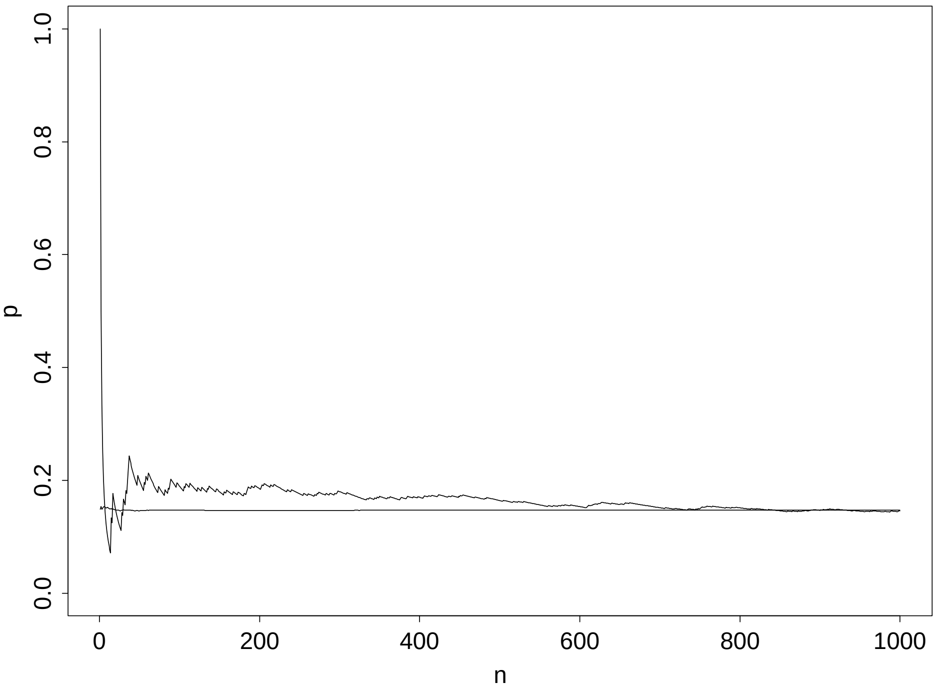

Figure 2.11: Comparison of importance sampled mean with standard estimator

Control and antithetic variates

In general, the idea of control variates is to modify an estimator according to a correlated variable whose mean is known. Thus, if we wish to estimate $\theta = E(Z)$ where $Z = \phi(X)$, we use the estimator

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n}\{Z_i - W_i\} + E(W) \tag{2.21}$$

which, of course, is unbiased for θ, but since

$$\mathrm{Var}(\hat{\theta}) = \frac{1}{n}(\mathrm{Var}(Z) - 2\mathrm{Cov}(W, Z) + \mathrm{Var}(W)), \tag{2.22}$$

the variance can be low if $\mathrm{Cov}(W, Z)$ is sufficiently large. In fact, if W is chosen to be a linear regression so that

$$Z = \beta_1 W_1 + \ldots \beta_p W_p \tag{2.23}$$

the variance of the estimator becomes approximately $n^{-2}RSS$ where RSS is the residual sum of squares of the regression fit.

This is easily applied in the context of Monte–Carlo integration. Again developing Ripley's Cauchy example, we require an estimate of

$$\theta = \frac{1}{2} - \int_0^2 f(x)dx \tag{2.24}$$

where $f(x) = \frac{1}{\pi(1+x^2)}$. We can estimate this as

$$\hat{\theta} = \frac{1}{2} - 2 \times \frac{1}{n}\sum_{i=1}^{n} f(x_i), \tag{2.25}$$

where the $x_i \sim U[0, 2]$.

To estimate the integral using a control variate we seek a function (or functions) with known mean which varies with $f(x)$ over $[0, 2]$. A Taylor series expansion of $f(x)$ suggests control variates of x^2 and x^4, whose means, with respect to the $U[0, 2]$ distribution, are easily evaluated over $[0, 2]$ as 8/6 and 32/10. Now, in principle, any function of the form $\beta_1 x^2 + \beta_2 x^4$ suffices as a control variate. To minimize variance however, we need to choose the β's so as to optimize agreement between $f(x)$ and $\beta_1 x^2 + \beta_2 x^4$. This is achieved through simple regression. I chose to simulate 10 observations uniformly over $[0, 2]$, leading to the regression equation

$$W(x) = .288 - .143x^2 + .024x^4 \tag{2.26}$$

and hence,

$$\hat{\theta} = \frac{1}{2} - 2\left[\frac{1}{n}\sum_{i=1}^{n}\{f(x_i) - W(x_i)\} + .288 - .143 \times 8/6 + .024 \times 32/10\right] \tag{2.27}$$

With $n = 1000$ I got an estimate of $\theta = .1474$.

Antithetic variates are almost the converse of control variates: we obtain a variate Z^* which has the same distribution as Z, but is negatively correlated with Z. Then,

$$\hat{\theta} = \frac{1}{2}(Z + Z^*) \tag{2.28}$$

is unbiased for θ, with variance:

$$\mathrm{Var}(\hat{\theta}) = \frac{1}{2}\mathrm{Var}(Z)\{1 + \mathrm{Corr}(Z, Z^*)] \tag{2.29}$$

which constitutes a reduction in variance provided the correlation is indeed negative. For simple problems, antithetic variates are easily achieved by inversion, since if $Z = F^{-1}(U)$ then $Z^* = F^{-1}(1-U)$ has the same distribution as Z and can be shown to be negatively correlated with Z for all choices of F. Applying this to the estimation of $\theta = \frac{1}{2} - \int_0^2 f(x)dx$ in the Cauchy example leads to the estimator

$$\frac{1}{2} - \frac{1}{n}\sum_{i=1}^{n}\left\{\frac{1}{\pi(1+u_i^2)} + \frac{1}{\pi(1+(2-u_i)^2)}\right\} \tag{2.30}$$

where $u_i \sim U[0, 2]$.

2.7 Stochastic processes

It is impossible to give general guidance on the simulation of stochastic processes, since the techniques required are often very specific. In most cases it is possible to exploit some characterization of a process as an aid to simulating it. The simplest example of this is the Poisson process, for which it is easiest to exploit the exponentiality of inter–arrival times as the basis for simulation. A similar procedure works for renewal processes.

Time series can often be simulated in the obvious recursive way. Spatial process often have a (spatial) Poisson process implicit in their characterization, which then leads naturally to a simulation technique.

Often iterative methods are needed to approximately simulate from stochastic processes. In particular for complex spatial processes, MCMC algorithms (see Chapter 3) need to be devised to simulate the processes.

Approximations to continuous time processes such as diffusion processes are more difficult again. Here it is not even possible to simulate exactly a representative sample path of the process. Here it is necessary to work with a fine discretization of the process.

Chapter 3

Markov Chain Monte Carlo

3.1 Bayesian statistics and conditional conjugacy

More than any other technique, Markov Chain Monte Carlo (MCMC) has been responsible for a current resurgence in Bayesian statistics, since its application allows a vast range of Bayesian models, previously thought to be completely intractible, to be estimated with ease. We will look at a number of applications later. Although MCMC is intrinsically just a device for simulating from a multivariate distribution, its application in statistics using the Gibbs sampler is greatly aided by the statistical concept known as *conditional conjugacy.*

Recall the notion of *conjugacy* used (normally) in Bayesian problems where there is a single unknown parameter θ, say. The concept is best illustrated using an example. Suppose that our data $X_1, X_2, \ldots$ are independent observations from an exponential distribution with parameter θ. Thus the likelihood is

$$L(\theta) = \theta^n e^{-\theta \sum_{i=1}^{n} x_i} . \tag{3.1}$$

Suppose our prior for θ is Gamma$(2, 1)$. Then the posterior distribution for θ given $\mathbf{x}$ is Gamma$(2 + n, 1 + \sum_{i=1}^{n} x_i)$. Thus the posterior distribution of θ remains in the two parameter gamma family whatever data is observed. The gamma family is called a *conjugate* family for this problem. The major advantage of the conjugate family is that it allows the effect of the data on our posterior beliefs to be summarised in terms of two parameters, which in turn describe a simple well known and easy to handle distribution.

It is usually possible to find useful conjugate priors in many problems with a single unknown parameter. However, even though in theory this is also possible in higher dimensional situations, in practice the conjugate families end up being highly complex and difficult to summarise easily.

On the other hand, many multi-parameter Bayesian problems exhibit conditional conjugacy. Again consider an example to illustrate the concept.

Example 3.1 *Suppose $X_i|\mu, \sigma^2 \sim N(\mu, \sigma^2)$ independently $1 \leq i \leq n$. Suppose also we have prior distributions for μ and σ^2:*

$$p(\mu) \sim N(\mu_0, \sigma_0^2) \tag{3.2}$$

$$p(\sigma^{-2}) \sim \text{Gamma}\,(\alpha_0, \beta_0) \tag{3.3}$$

where μ and σ are considered to be a priori independent and μ_0, σ_0^2, α_0 and β_0 are considered to be known hyperparameters. Writing $\tau = \sigma^{-2}, \tau_0 = \sigma_0^{-2}$ we can write the posterior distribution of (μ, τ):

$$\pi(\mu, \tau|\mathbf{x}) \propto \textit{likelihood} \times \textit{prior} \tag{3.4}$$

$$\propto \left[\prod_{i=1}^{n} e^{-\frac{\tau}{2}(x_i-\mu)^2}\right] e^{-\frac{\tau_0}{2}(\mu-\mu_0)^2}\tau^{\alpha_0+\frac{n}{2}-1}e^{-\beta_0\tau}. \tag{3.5}$$

Note that although the form of the prior distribution is highly tractable, the posterior distribution is a complex two-dimensional distribution. This is typical of Bayesian analyses of problems with multidimensional parameter sets.

However, we can describe the posterior distribution in terms of its conditional distributions:

$$\pi(\mu|\tau,\mathbf{x}) \propto \left[\prod_{i=1}^{n} e^{-\frac{\tau}{2}(x_i-\mu)^2}\right] e^{-\frac{\tau_0}{2}(\mu-\mu_0)^2} \tag{3.6}$$

$$\sim N\left(\frac{\tau\Sigma x_i+\mu_0\tau_0}{n\tau+\tau_0}, \frac{1}{n\tau+\tau_0}\right) \tag{3.7}$$

and

$$\pi(\tau|\mu,\mathbf{x}) \sim \text{Gamma}\left(\alpha_0+\frac{n}{2}, \beta_0+\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2}\right). \tag{3.8}$$

This phenomenon, where the conditionals have nice simple forms, is called conditional conjugacy, and is common, even in extremely complex high-dimensional problems. This will motivate the introduction of the Gibbs sampler in the next section.

In multi-dimensional Bayesian problems it is rarely possible to analytically compute summary statistics such as posterior means and variances, or even posterior probabilities. Therefore it is necessary to estimate the quantities of interest using a Monte Carlo approach. However simulating from an arbitrary high dimensional distribution is usually diffucult and often impossible to do directly. Markov chain Monte Carlo (MCMC) simulates instead a Markov chain, who's stationary or limiting distribution is the posterior distribution of interest.

The concept of conditional conjugacy is crucial in the construction of one of the basic forms of MCMC: the Gibbs sampler.

3.2 The Gibbs sampler

We wish to obtain a sample from the multivariate distribution $P(\theta_1,\ldots,\theta_d)$. The Gibbs sampler successively and repeatedly simulates from the conditional distributions of each component given the other components. Under conditional conjugacy, this simulation step is usually straightforward.

Algorithm 3.1

```
0. Initialize with $\theta = (\theta_1^{(0)},\ldots,\theta_d^{(0)})$.
1.1 Simulate $\theta_1^{(1)}$ from the conditional $\theta_1|(\theta_2^{(0)},\ldots,\theta_d^{(0)})$.
1.2 Simulate $\theta_2^{(1)}$ from the conditional $\theta_2|(\theta_1^{(1)},\theta_3^{(0)},\ldots,\theta_d^{(0)})$.
 ...
1.d Simulate $\theta_d^{(1)}$ from the conditional $\theta_d|(\theta_1^{(1)},\ldots,\theta_{d-1}^{(1)})$.
```

2. *Iterate this procedure.*

Under mild regularity conditions, convergence of the Markov chain to the stationary distribution $P(\theta_1, \dots, \theta_d)$ is guaranteed, so after a burn–in period, the observations $(\theta_1^{(k)}, \dots, \theta_d^{(k)}), \dots (\theta_1^{(n)}, \dots, \theta_d^{(n)})$ can be regarded as realizations from this distribution.

As stressed, this procedure is valid in any situation where the requirement is a sample from a multivariate distribution. Applications tend to have been concentrated in Bayesian statistics because the technique gives a sample based approach to posterior inference in situations where most other techniques are either difficult or impossible. Bayes' theorem has the form

$$P(\theta|x) = \frac{P(\theta)P(x|\theta)}{\int P(\theta)P(x|\theta)d\theta} \tag{3.9}$$

and it is often impossible to solve the normalizing integrals. With the Gibbs sampler, as long as we can simulate from each of the conditional distributions, then we can obtain a sample directly from the posterior distribution without having to worry about any integrals.

3.3 Example: a poisson count change point problem

To illustrate the use of Gibbs sampling in Bayesian statistics, we'll look at an example taken from Carlin *et al.*, concerning the change point in a Poisson process. In particular, they cite as example a series relating to the number of British coal mining disasters per year, over the period 1851 — 1962. A plot of these data is given in Figure 3.1, and the time series itself is stored in `coal.dat`.

From this plot it does seem to be the case that there has been a reduction in the rate of disasters over the period. We'll follow the model of Carlin *et al.* which has the form:

$$Y_i \sim \text{Poisson}(\theta); \quad i = 1, \dots k; \tag{3.10}$$

$$Y_i \sim \text{Poisson}(\lambda); \quad i = k+1, \dots n. \tag{3.11}$$

Thus, the model is for a Poisson number of disasters per year, with a mean rate of θ up to the kth year, and a rate of λ thereafter.

The Bayesian specification of the model is completed with a hierarchical framework: $\theta \sim \text{Gamma}(a_1, b_1)$, $\lambda \sim \text{Gamma}(a_2, b_2)$, k discrete uniform over $\{1, \dots, 112\}$, each independent of one another, and then $b_1 \sim \text{Gamma}(c_1, d_1)$ and $b_2 \sim \text{Gamma}(c_2, d_2)$.

It is not too difficult to check that this choice of specification leads to the following conditionals:

$$\theta|Y, \lambda, b_1, b_2, k \sim \text{Gamma}\left(a_1 + \sum_{i=1}^{k} Y_i, k + b_1\right) \tag{3.12}$$

$$\lambda|Y, \theta, b_1, b_2, k \sim \text{Gamma}\left(a_2 + \sum_{i=k+1}^{n} Y_i, n - k + b_2\right) \tag{3.13}$$

$$b_1|Y, \theta, \lambda, b_2, k \sim \text{Gamma}(a_1 + c_1, \theta + d_1) \tag{3.14}$$

$$b_2|Y, \theta, \lambda, b_1, k \sim \text{Gamma}(a_2 + c_2, \lambda + d_2) \tag{3.15}$$

and

$$p(k|Y, \theta, \lambda, b_1, b2) = \frac{L(Y; k, \theta, \lambda)}{\sum_{j=1}^{n} L(Y; j, \theta, \lambda)} \tag{3.16}$$

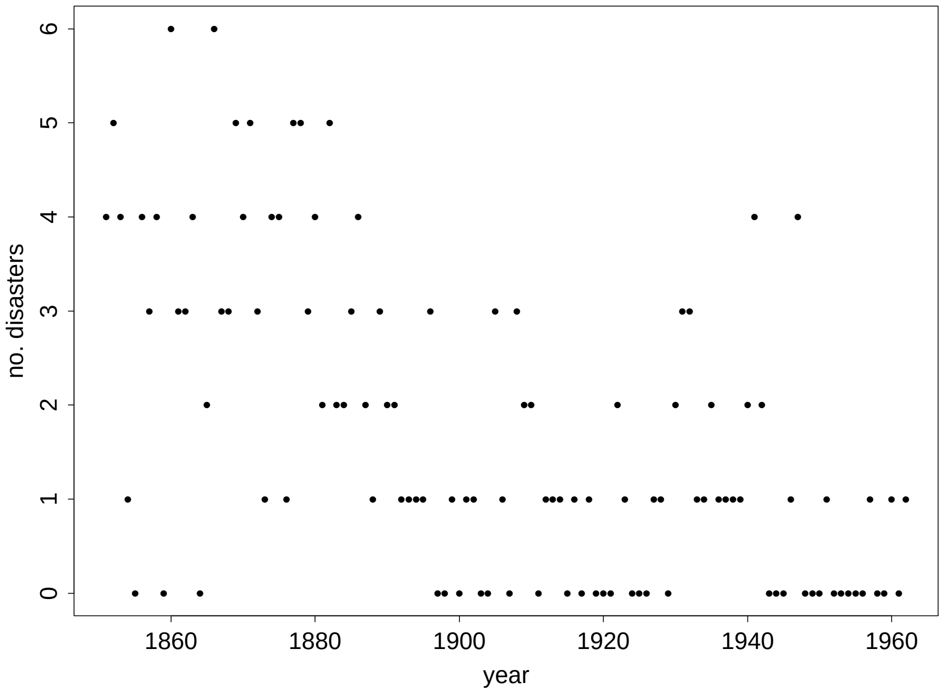

Figure 3.1: Time series of counts of coal mine disasters

where

$$L(Y;k,\theta,\lambda) = \exp\left\{k(\lambda-\theta)\right\}(\theta/\lambda)^{\sum_{i=1}^{k} Y_i} \tag{3.17}$$

This is implemented in Splus with the following code:

```
gibbs2_function(n, dat, init, const)
{
        l <- NULL
        th <- init[1]
        la <- init[2]
        b1 <- init[3]
        b2 <- init[4]
        k <- init[5]
        a1 <- const[1]
        a2 <- const[2]
        c1 <- const[3]
        c2 <- const[4]
        d1 <- const[5]
        d2 <- const[6]
        nn <- length(dat)
        th.o <- NULL
        la.o <- NULL
        k.o <- NULL
        for(i in 1:n) {
                th <- rgamma(1, a1 + cumsum(dat)[k])/(k + b1)
                la <- rgamma(1, a2 + sum(dat) - cumsum(dat)[k])/(nn - k + b2)
                b1 <- rgamma(1, a1 + c1)/(th + d1)
                b2 <- rgamma(1, a2 + c2)/(la + d2)
                for(j in 1:nn) {
                        l[j] <- exp((la - th) * j) * (th/la)^(cumsum(dat)[j])
                }
                k <- sample(1:nn, size = 1, prob = l)
                th.o <- c(th.o, th)
                la.o <- c(la.o, la)
                k.o <- c(k.o, k)
        }
        cbind(th.o, la.o, k.o)
}
```

The sequence of output for 1100 iterations of θ, λ and k with the prior specification $a_1 = a_2 = 0.5$, $c_1 = c_2 = 0$, $d_1 = d_2 = 1$ is shown in Figure 3.2.

Convergence of the algorithm seems rapid (after compensation for poor choice of starting values). So, I've deleted the first 100 values, and based the subsequent analysis on the remaining 1000 points. A frequency plot of the posterior distribution of k is given in Figure 3.3.

On the basis of this estimate, it is almost certain that a changepoint has occurred, with the posterior mode estimate being $k = 41$, corresponding to a changepont at the year 1891.

Kernel density estimates of the posterior distributions of θ and λ are given in Figure 3.4.

It is clear from Figure 3.4 that λ is (almost certainly) less than θ. An important feature of the Gibbs sampler approach is that with almost no extra effort, this feature can be examined directly. That is, by looking at the sequence of realisations of $(\theta - \lambda)_i$, we obtain a sample from the posterior marginal distribution of $P(\theta - \lambda)$. No complicated transformation theory is needed

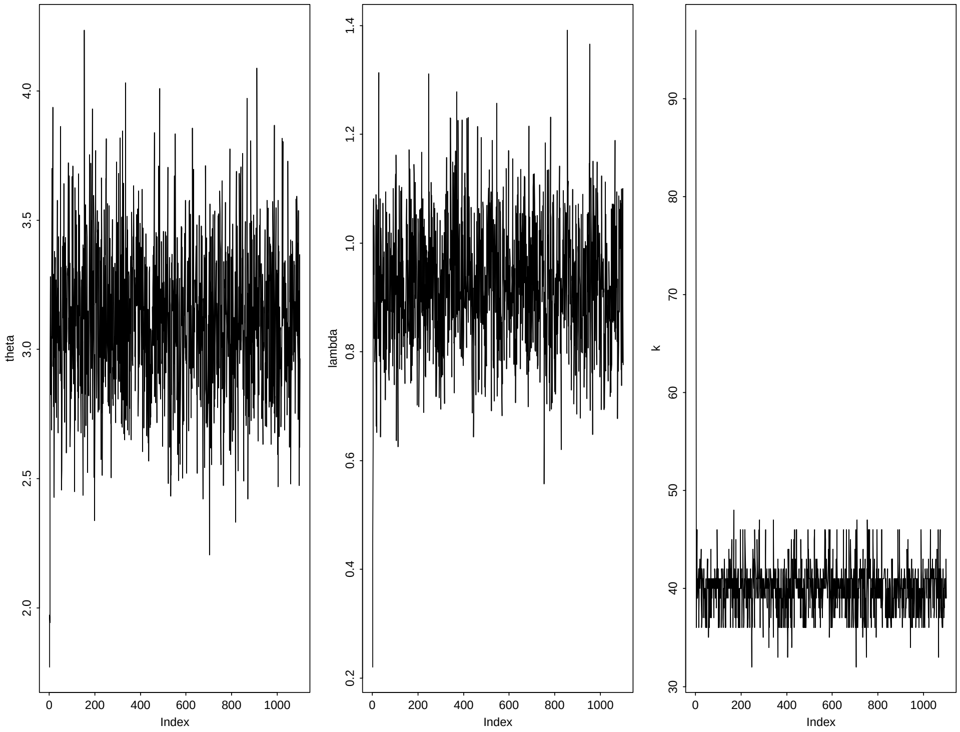

Figure 3.2: Gibbs sample of Poisson changepoint model parameters

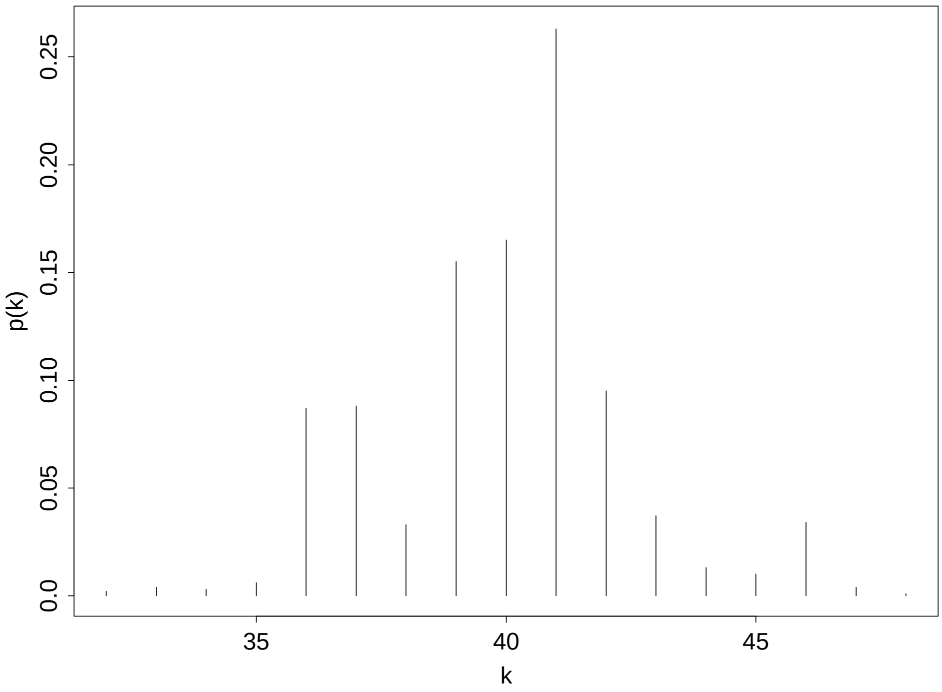

Figure 3.3: Frequency plot of changepoint parameter k

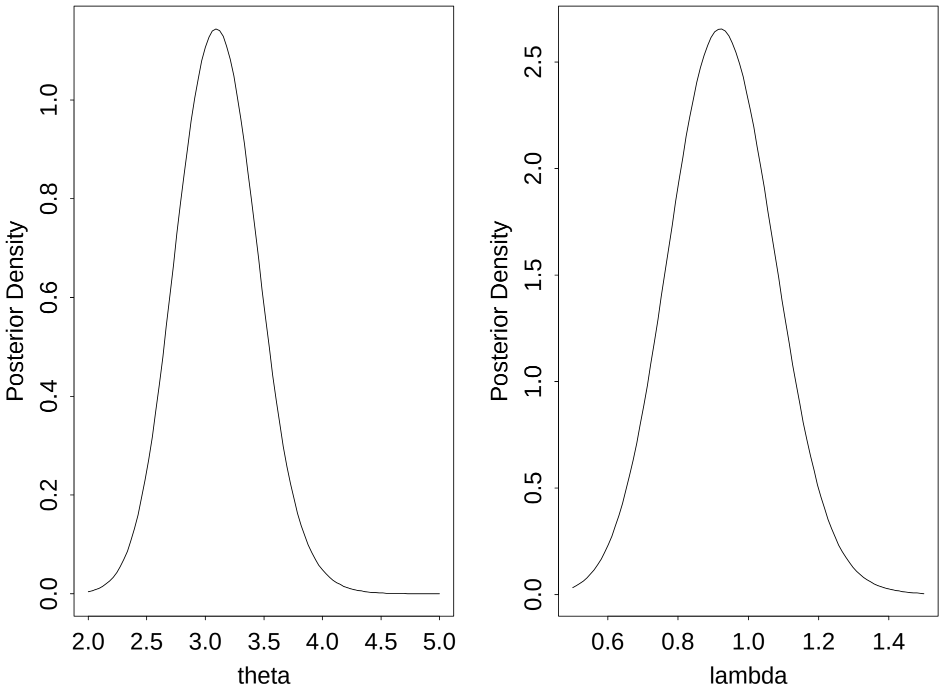

Figure 3.4: Kernel density estimates of parameters θ and λ

because we are working directly on the sample. A kernel density estimate of $P(\theta - \lambda)$ produced this way is shown in Figure 3.5.

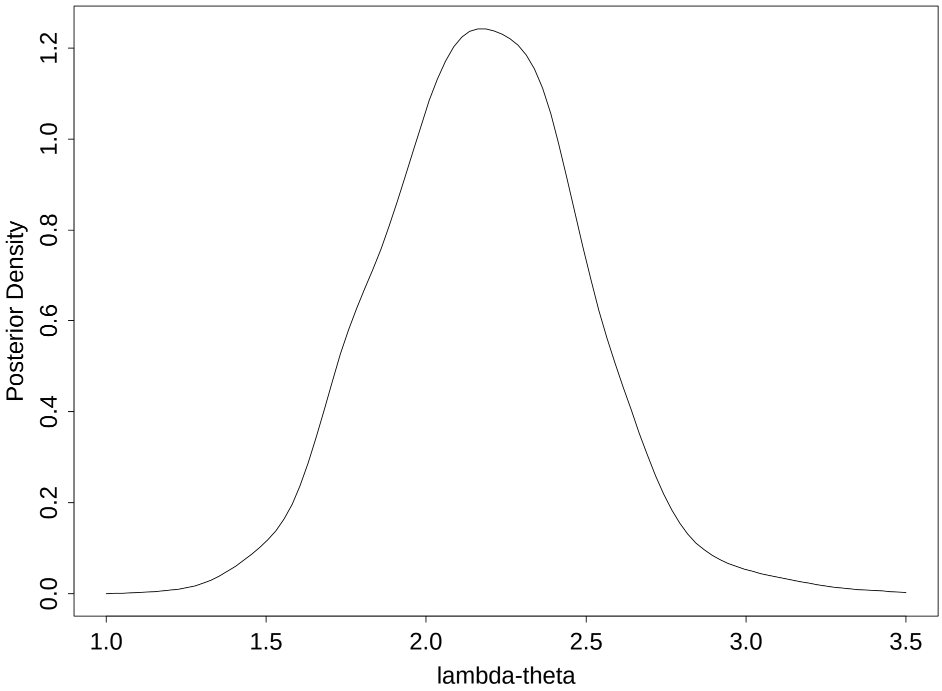

Figure 3.5: Kernel density estimate of $\theta - \lambda$

3.4 Data augmentation

Data augmentation is a technique which can be thought of as a special case of the Gibbs sampler. It can be applied equally well to the following two situations.

1. One of the main problems that afflicts statistical computing is that in the real world, some of our data could be missing. In theory the distribution of the observed data can be obtained by integrating out the missing data. However this is frequently difficult if not impossible to do.
2. The likelihood if the data is not tractable for one reason or another (for instance there is no simple conjugate prior) but that conditional on a collection of unobserved data, the likelihood becomes easy to handle.

These two situations are best illustrated using simple examples.

1. Suppose $Y_0, Y_1, \ldots, Y_n$ is a time series of observations defined by $Y_0 = 0$ and for each $1 \leq i \leq n$
$$Y_i | Y_0, Y_1, \ldots Y_{i-1} \sim Y_{i-1} + \text{Beta}(\theta, \theta) \ , \tag{3.18}$$

for $\theta > 0$. Then given the full dataset $Y_0, Y_1, \ldots, Y_n$, the likelihood of θ is easy to write down (as a product of beta terms for the successive increments). However suppose that Y_{i^*} is missing, then the likelihood is now not available in closed form.

2. Suppose that $Y_1, \ldots Y_n$ are IID data from the *mixture* density

$$\frac{1}{2\sqrt{2\pi}}\left(e^{-x^2/2} + e^{-(x-\theta)^2/2}\right) . \tag{3.19}$$

This mixture density can be thought of as the density of the random variable obtained by the following procedure. Toss a fair coin. Given a head sample from a $N(0,1)$ distribution. Given a tail, sample from a $N(\theta,1)$ distribution. Now the likelihood of θ can be written explicitly:

$$\prod_{i=1}^{n} \frac{1}{2}\left(e^{-y_i^2/2} + e^{-(y_i-\theta)^2/2}\right) . \tag{3.20}$$

However this is a difficult function to maximize (for classical inference) or compute a posterior distribution from. However, suppose I were to know about the sequence of heads and tails, that is the information about which observations came from $N(0,1)$ (and therefore irrelevant to inference about θ) and which came from $N(\theta,1)$. Then the inference problem is straightforward (both in a classical and a Bayesian framework).

In both of these problems, the type of inference (whether classical or Bayesian) is not relevant. So it turns out that the numerical techniques we use to exploit the fact that *more data makes the inference easier* are highly related. We'll deal with the EM algorithm (for maximum likelihood inference) later.

Now introduce some general notation. Let y_{obs} denote the observed data and y_{mis} the missing data. Let θ be the unknown parameters with prior $p(\theta)$. Then the situation that the above examples demonstrate is where $f(y_{obs}, y_{mis}|\theta)$ is available. As a result of this, the conditional distribution of θ and Y_{mis} is proportional to

$$\pi(\theta, y_{mis}) = f(y_{obs}, y_{mis}|\theta)p(\theta). \tag{3.21}$$

Data augmentation proceeds by carrying out Gibbs sampling to successively sample from θ and Y_{mis} to produce a sample from this joint distribution. The marginal distribution of θ is therefore the posterior distribution of interest.

How does this work in the two examples above?

1. Given a uniform prior for θ on $(0,1)$, and given complete data $Y_0, Y_1, \ldots, Y_n$, the posterior for θ is explicit and can be sampled by inversion. The distribution of $Y_{i^*}|$everything else, can be seen to have a density proportional to

$$\left((y_{i^*} - y_{i^*-1})(y_{i^*-1} + 1 - y_{i^*})(y_{i^*} - y_{i^*+1} - 1)(y_{i^*+1} - y_{i^*})\right)^{\theta-1} \tag{3.22}$$

on the region $(y_{i^*-1}, y_{i^*-1}+1) \cap (y_{i^*+1}-1, y_{i^*+1})$. This sampling can be carried out (among other ways) by rejection sampling.

2. Here Y_{mis} is a sequence of n heads or tails. Suppose that the prior for θ is $N(0,1)$. Then its posterior given that Y_{mis} allocates Y_i to the $N(\theta,1)$ component for all $i \in A$ is

$$N\left(\frac{\sum_{i\in A} Y_i}{1+|A|}, \frac{1}{1+|A|}\right) , \tag{3.23}$$

where $|A|$ denotes the number of elements in A. On the other hand if C_i is the result of the coin toss for the ith observation, then it can be seen that

$$P(C_i = T|\theta, \text{ everything else}) = \frac{e^{-(y_i-\theta)^2/2}}{e^{-(y_i-\theta)^2/2} + e^{-y_i^2/2}} . \tag{3.24}$$

3.5 A worked example in genetics

We look at a genetic linkage example. This example is well–worn, but is useful because it illustrates very simply all the ideas necessary for this (and subsequent) algorithms. The example concerns genetic linkage of 197 animals, in which the animals are distributed into 4 categories:

$$Y = (y_1, y_2, y_3, y_4) = (125, 18, 20, 34) \tag{3.25}$$

with cell probabilities

$$(\frac{1}{2} + \frac{\theta}{4}, \frac{1}{4}(1-\theta), \frac{1}{4}(1-\theta), \frac{\theta}{4}) \tag{3.26}$$

Though it is just possible to sample the posterior distribution of θ directly by (say) rejection sampling), data augmentation brings about a substantial simplification. Specifically, we augment the observed data Y by dividing the first cell into two, with respective cell probabilities $\frac{1}{2}$ and $\frac{\theta}{4}$, giving an augmented data set $X = (x_1, x_2, x_3, x_4, x_5)$, where $x_1 + x_2 = y_1$, and $x_3 = y_2, x_4 = y_3, x_5 = y_4$. Now using a flat prior for θ, we have

$$P(\theta|Y) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4} \tag{3.27}$$

whereas

$$P(\theta|X) \propto \theta^{x_2+x_5}(1-\theta)^{x_3+x_4} \tag{3.28}$$

In this case, and writing $X = (y, Z)$ for 'missing' data $Z = X_2$ we have

$$\theta|Z, Y \sim \text{Beta}(Z + X_5 + 1, X_3 + X_4 + 1) \tag{3.29}$$

and

$$Z|\theta, Y \sim \text{Bin}(125, \frac{\theta}{\theta+2}) \tag{3.30}$$

This is easily coded in Splus as

```
gibbs1_function(n, xa)
{
        z <- 20
        th <- 0.5
        z.o <- z
        th.o <- th
        for(j in 1:n) {
                th <- rbeta(1, z + xa[4] + 1, xa[2] + xa[3] + 1)
                p <- th/(th + 2)
                z <- rbinom(1, xa[1], p)
                z.o <- c(z.o, z)
                th.o <- c(th.o, th)
        }
        cbind(th.o, z.o)
}
```

Note that the initial values `z=20` and `th=0.5` are chosen arbitrarily. Applying this algorithm with `n=100` iterations gave the output in Figure 3.6.

It seems as though convergence of the algorithm is immediate, though there is perhaps some evidence that the chain for θ behaves slightly differently at the start. Consequently, I've elected to disregard the first 20 realisations (this is the burn–in). The remaining sample is regarded as a sample from the required joint distribution $P(\theta, Z|Y)$. In particular, histograms of the

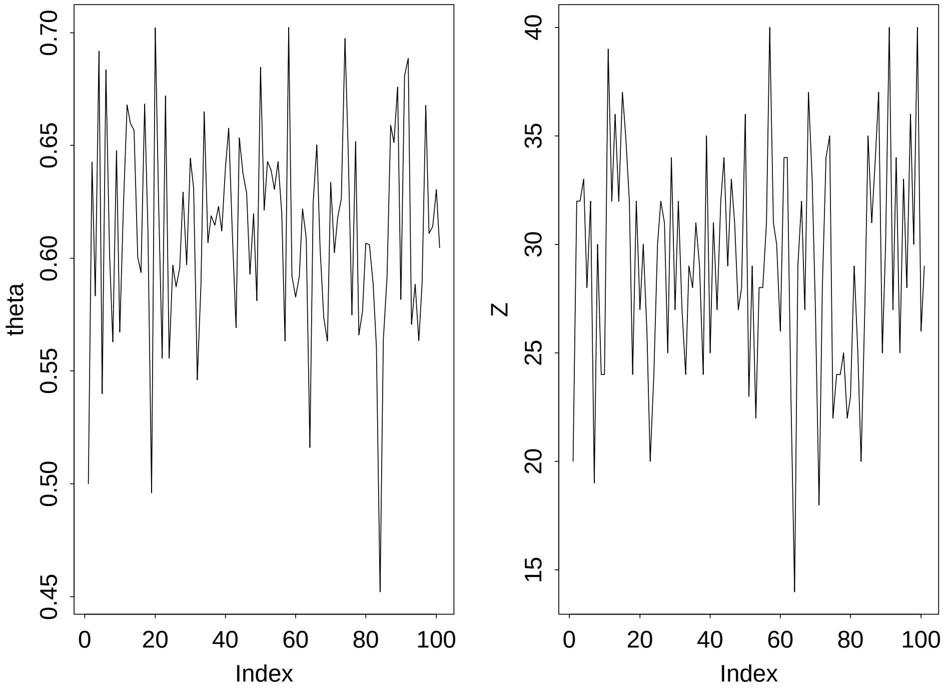

Figure 3.6: Realizations from data augmentation in genetic example

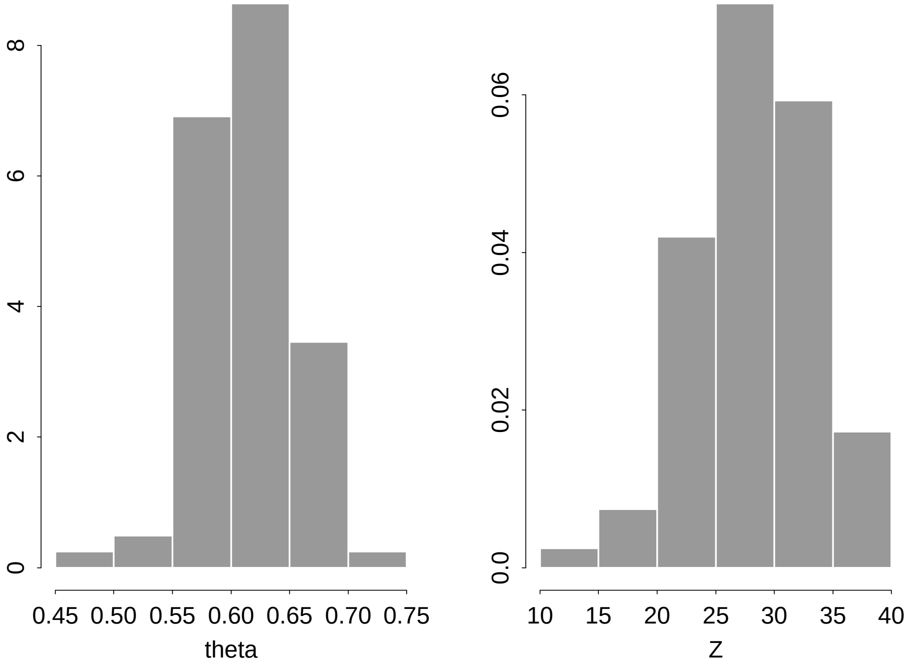

Figure 3.7: Marginal posteriors in genetic example

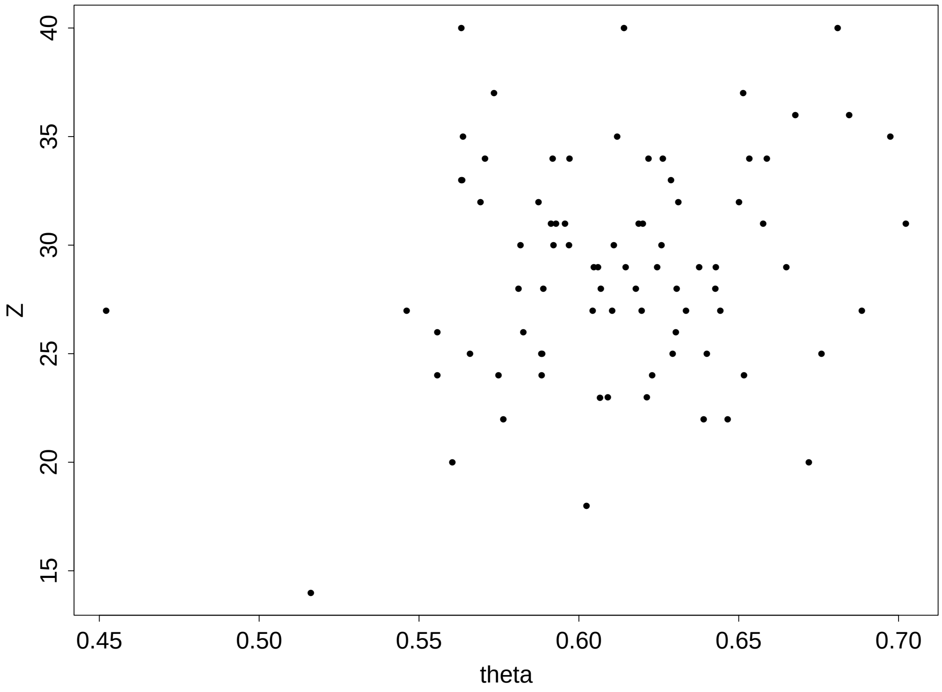

Figure 3.8: Scatterplot of (θ, Z) pairs in genetic example

separate θ and Z components (Figure 3.7) are estimates of the posterior distributions $P(\theta|Y)$ and $P(Z|Y)$. Similarly, a plot of the (θ, Z) pairs (Figure 3.8) gives an impression of the joint posterior $P(\theta, Z|Y)$.

Interpretation of these results does need care, since the sequence of realisations are not (by definition) independent. There are two alternative strategies:

1. Run a single chain for sufficiently long, so that dependence between successive realisations does not diminish the precision of the sample information; and

2. Run several chains, and average across them.

I'm going to stick with the first of these approaches, though it should also be recognised that approaches to assessing convergence of such chains are often based on comparisons of within–chain and across–chain variability.

3.6 Prediction

Another example of the application of the Gibbs sampler is for prediction. The predictive distribution of a future observation y given past observations x is defined as

$$p(y|x) = \int p(y|\theta)p(\theta|x)d\theta. \tag{3.31}$$

so that the likelihood, $p(y|\theta)$ is averaged across the uncertainty in θ contained in the posterior distribution $p(\theta|x)$. Hence, given a sampled sequence of realizations $\theta^{(1)}, \ldots, \theta^{(n)}$ from this posterior, we can estimate

$$p(y|x) \approx \frac{1}{n}\sum_{i=1}^{n} p(y|\theta_i) \tag{3.32}$$

For the coal mining example, we can use this estimator to estimate the predictive distribution of the number of disasters in a future year (for which the Poisson rate is λ). A graph of the predictive distribution is given in Figure 3.9, together with an estimate simply based on the posterior mean of λ. In this case, the lack of variation in the posterior distribution for λ has meant that the two distributions are almost identical, but in general the predictive distribution is likely to have much greater variation on account of the uncertainty in λ.

3.7 The Metropolis–Hastings algorithm

The Gibbs sampler is the best publicised MCMC algorithm, but there are many others. The most general algorithm in this context is the Metropolis–Hastings algorithm. To simplify notation this time, suppose that we wish to simulate from a (multivariate) distribution $\pi(x)$. We now let $q(x, y)$ be any arbitrary transistion probability (that is $q(x, y)$ is the probability density of moving to y from x), but from which simulation is straightforward. Then the Metropolis–Hastings algorithm is:

Algorithm 3.2

1. *Given the current position $X_n = x$, generate a 'candidate value', y^* from $q(x, y)$.*

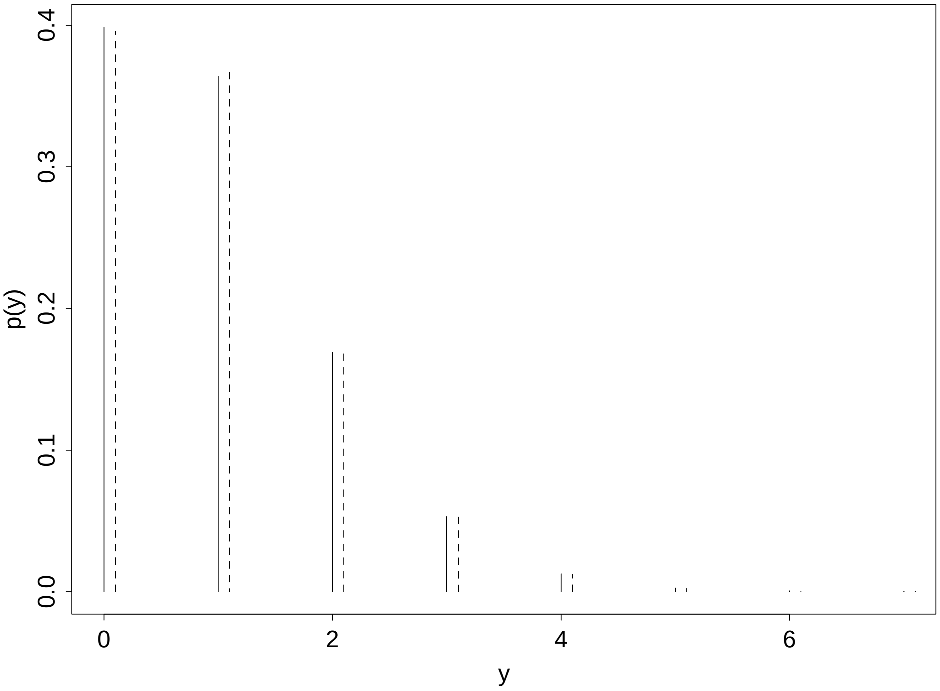

Figure 3.9: Predictive and estimative distributions of number of coal mining disasters in a year (solid line is predictive)

2. *Calculate*

$$\alpha(x,y) = \begin{cases} \min\{\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1\} & \textit{if } \pi(x)q(x,y) > 0 \\ 1 & \textit{if } \pi(x)q(x,y) = 0 \end{cases}$$

   with $y = y^*$.

3. *With probability* $\alpha(x, y^*)$ *accept the candidate value and set* $X_{n+1} = y^*$*; otherwise reject and set* $X_{n+1} = x$.

4. *Repeat.*

It's not difficult to show that $\pi(x)$ is a stationary distribution of this chain. Under mild conditions, the chain will converge to this equilibrium distribution.

The Metropolis–Hastings algorithm has the major advantage over the Gibbs sampler that it is not necessary to know all the conditional distributions — we only need to simulate from q which we can choose arbitrarily. Moreover, and this can be of crucial importance, we only need to know π up to proportionality, since any constants of proportionality cancel in the numerator and denominator of the calculation of α. The price for the simplicity is that if q is poorly chosen, then the number of rejections can be high, so that the efficiency of the procedure can be very low.

We can illustrate the procedure with the genetic linkage example for which we had

$$\pi(\theta) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4} \tag{3.33}$$

Taking, for simplicity, $q(x,y) = 1$ on $[0,1]$, Splus code is

```
m.hast_function(n, xa)
{
        pi.q <- function(th, xa){
        (2 + th)^xa[1] * (1 - th)^(xa[2] + xa[3]) * th^xa[4]}
        th <- 0.2
        th.o <- th
        for(j in 1:n) {
                y <- runif(1)
                al <- min(pi.q(y, xa)/pi.q(th, xa), 1)
                u <- runif(1)
                if(u < al)
                        th <- y
                th.o <- c(th.o, th)
        }
        th.o
}
```

As with the Gibbs sampler, it's necessary to monitor output to ensure convergence. Figure 3.10 shows simulated values of θ for 1100 iterations. Convergence seems ok after the first few iterations, so again I'll just delete the first 100 observations. Note that the chain is highly dependent as a consequence of the high rejection rate because of the choice of q.

The corresponding posterior density estimate of θ is given in Figure 3.11.

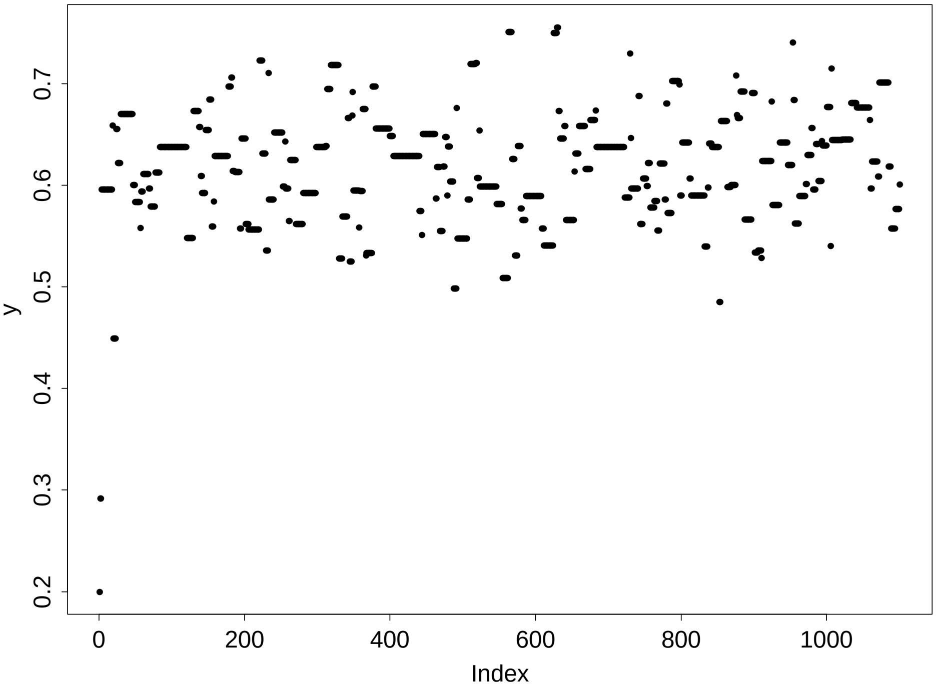

Figure 3.10: Series of θ in Metropolis–Hastings algorithm

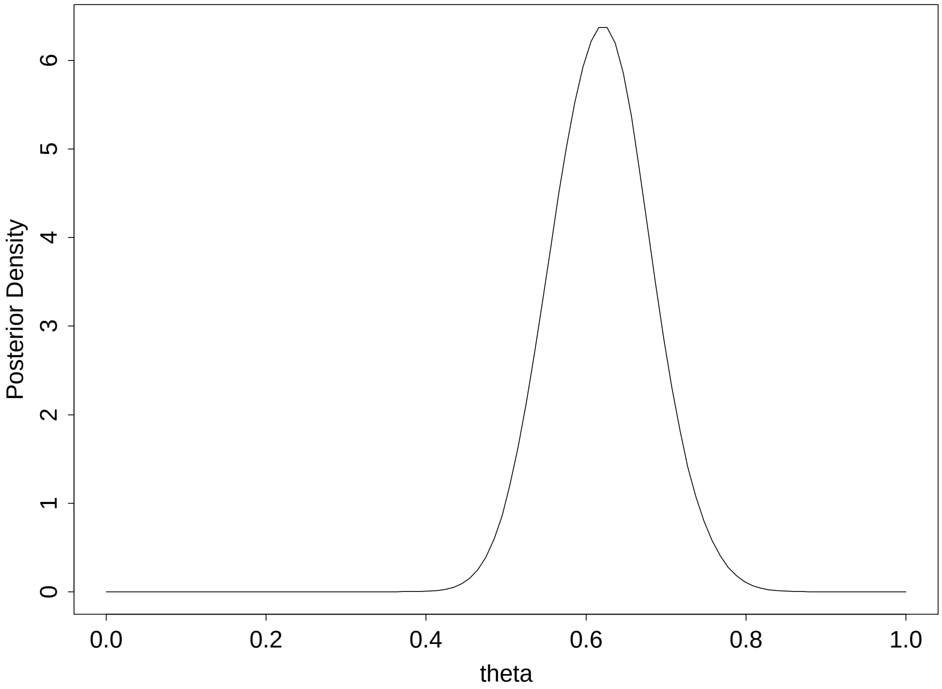

Figure 3.11: Posterior density of θ based on Metropolis–Hastings algorithm

3.8 Hybrid chains

It is possible, and often necessary, to create hybrid chains whereby Monte Carlo chains are imbeded within Monte Carlo chains of a different type. A particularly useful application of this is when using the Gibbs sampler, if it is not possible to obtain the conditional distributions required in exact form. A natural approach is to use a one–dimensional Metropolis–Hastings step to simulate the one–dimensional conditionals. This is straightforward since, up to proportionality, the conditionals are obtained as the joint posterior evaluated with the conditional ordinates fixed.

3.9 Uses of MCMC in classical statistics and beyond...

MCMC has certainly had a fundamental effect on statistics in the last 10 years, influencing not only the way quantities for inference are computed, but also the type of models the statistician tries to fit. This is because the flexibility of the technique expands immeasurably the classes of statistical models for which computation is considered possible. This effect is most pronounced in Bayesian statistics. However MCMC also has many important applications in classical statistics.

We'll briefly mention three examples.

1. Firstly note that given a flat prior, the likelihood and the posterior density are equal. Therefore it is feasible to estimate the maximum of the likelihood as the mode of a density estimate of the posterior distribution. This involves the additional problem of obtaining an estimate of the density from a sample of points from that density. This is usually done in terms of a *kenel density estimate.* For example, given a sample $x_i, 1 \leq i \leq n$ we could estimate the density as

$$\frac{1}{n}\sum_{i=1}^{n} \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\{(x-x_i)^2/(2\sigma^2)\} \tag{3.34}$$

   The 'smootheness' parameter σ^2 needs to be chosen. As a general rule, the larger n, the smaller we would normally like to choose σ^2.

2. A rather different application is to the (common) situation where the density of the data given the unknown parameter is known only up to a normalisation constant. The following idea is due to Charlie Geyer.

$$l(\theta|x) = \frac{f_\theta(x)}{c(\theta)}, \tag{3.35}$$

   where $f_\theta(x)$ is an unnormalised density and $c(\theta)$ is its normalisation constant

$$c(\theta) = \int f_\theta(x)dx.$$

   The problem here is that the normalisation constant, $c(\theta)$, can easily be a function of θ and moreover will affect the maximisation problem for the likelihood. In this case, it is necessary to estimate c. Now suppose θ_0 is a 'first guess' of the MLE. We can write

$$\frac{l(\theta|x)}{l(\theta_0|x)} = \frac{f_\theta(x)}{f_{\theta_0}(x)}\frac{c(\theta_0)}{c(\theta)}\ , \tag{3.36}$$

   and we can estimate $c(\theta)/c(\theta_0)$ for an arbitrary θ from a sample $(x_1, \dots x_n)$ from f_{θ_0} using the estimator

$$\frac{1}{n}\sum_{i=1}^{n} \frac{f_\theta(x_i)}{f_{\theta_0}(x_i)}\ . \tag{3.37}$$

From this simulation therefore, we can estimate $l(\theta|x)/l(\theta_0|x)$ from (3.36), and improve our estimate of the MLE. This procedure is usually iterated a number of times until the estimated terms stabilize.

3. *Simulated annealing* is a stochastic algorithm for maximising functions. It is very closely related to MCMC. The only difference is that the target density changes as the algorithm proceeds. First notice that if we want to maximise a non-negative function $h(y), y \in A$, the maximising value also maximises $h(y)^{1/T}$ for arbitrary $T > 0$ known as the temperature. However for large T, assuming that $h^{1/T}$ is an integrable function, the distribution with density proportional to $h(y)^{1/T}$ concentrates most of its probability mass in the vicinity of the mode of the function. The idea behind simulated annealing is, at each iteration n, to carry out MCMC on a density proportional to h^{1/T_n}, where T_n is a sequence of temperatures that converges to zero. The algorithm converges to the maximum value of h under suitable regularity conditions. It is important that T_n does not converge to zero too rapidly, since it turns out that this could then lead to the algorithm getting stuck in a minor mode of the function h.

3.10 The dangers of MCMC

A real danger of MCMC is that it compute pretty much any model you can throw at it - whether or not the model is statistically sensible. Algorithms can produce very reasonable results, even when the number of parameters in the model far exceeds the size of the dataset.

In high dimensional situations, the assessment of convergence of MCMC is extremely problematic. Therefore it is rarely possible to be absolutely certain that our algorithm has converged sufficiently well. Moreover, even if we could start the algorithm from the target density, the dependent values in the subsequent chain can lead to extremely poor estimates of quantities of interest.

3.11 Appendix: Markov chains

A Markov chain $\{X_n, n \geq 0\}$ is a random process with the following 'memoryless' property:

$$\begin{aligned} P[X_n \in A \mid X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \ldots, X_0 &= x_0] \\ &= P[X_n \in A \mid X_{n-1} = x_{n-1}]. \end{aligned}$$

For simplicity and illustration of the basic ideas, assume here that the state space for X is countable. Let $P_{ij} = P[X_n = j \mid X_{n-1} = i]$, and let P denote the matrix with elements $\{P_{ij}\}$. Then $(P^n)_{ij} = P[X_n = j \mid X_0 = i]$. A Markov chain is

irreducible if, for all i and j, there exists some n such that $P_{ij}^n > 0$;

aperiodic if the greatest common divider of $\{n; P_{ij}^n > 0\} = 1$;

recurrent if P[Markov chain returns to i | it started at i] $= 1$ for all i; and

positive recurrent if it is irreducible aperiodic and recurrent, and $\{\pi_j\}$ is a collection of probabilities such that

$$\lim_{n\to\infty} P_{ij}^n = \pi_j \qquad (3.38)$$

and

$$\sum_i \pi_i P_{ij} = \pi_j \ . \tag{3.39}$$

For MCMC, the invariant probability distribution π is given a priori – this is the distribution from which we wish to sample. Therefore it remains to demonstrate irreducibility and aperiodicty. They can usually be easily checked in individual situations, although there do occur examples where the use of MCMC is thwarted by *reducibility*.

For an excellent review of the properties of Markov chains, see Grimmett and Stirzaker (Oxford University Press, 1992).

Chapter 4

The Bootstrap

4.1 Bootstrap variance

The bootstrap is a computer–based technique of assessing the variance, or some other property, of a statistical estimator. In its simplest setting, the idea is as follows. Suppose we have a sample $\mathbf{x} = x_1, x_2, \ldots, x_n$ from an unknown distribution F and we have an estimator $\hat{\theta}(\mathbf{x})$ of some population parameter θ.

If we could obtain many other samples, we could evaluate the estimator on each of these samples as well, and use the sample variance of the estimators to estimate the true variance of the estimator.

In actual fact, we only have the one sample to work with, so the idea of bootstrapping is to simulate not from the population, but from the single sample which we have available. For example, if our sample consisted of the values 3,5 the possible bootstrap samples are (3,3), (3,5), (5,5) with probabilities 1/4, 1/2, 1/4 respectively. (Note that the bootstrap samples are made with replacement). In this way the bootstrap samples mimic the ideal of resampling from the whole population. In theory, the true population variance of the estimator can be calculated with respect to the bootstrap distribution. In practice this is rarely possible, so it is usual to simulate samples from the bootstrap distribution and estimate the true bootstrap variance by the sample bootstrap variance.

Slightly more formally, the basic idea of bootstrapping is to replace the unknown distribution function F with a sample–based estimate, usually $\hat{F}$, the empirical distribution function defined with ordered sample values $x_{(1)} \leq x_{(2)} \leq \ldots x_{(n)}$ as

$$\hat{F}(x) = \frac{i}{n}; \qquad x_{(i)} \leq x < x_{(i+1)} \tag{4.1}$$

An example of the empirical distribution function for 20 values simulated from a standard normal is shown in Figure 4.1 together with the true standard normal distribution function. The point is that the empirical distribution function converges to the true distribution function, so that statistics based on the EDF will converge to those based on F itself.

Thus, for example, an estimate of $\mathrm{Var}_F(\hat{\theta}(\mathbf{x}))$, the variance of the estimator $\hat{\theta}$ with respect to F is given by the bootstrap variance , $\mathrm{Var}_{\hat{F}}(\hat{\theta}(\mathbf{x}))$, the corresponding variance but with respect to the bootstrap distribution.

Usually this is impossible to evaluate analytically, so an estimate is formed by simulating directly from $\hat{F}$. This gives rise to what is usually regarded as the bootstrap algorithm:

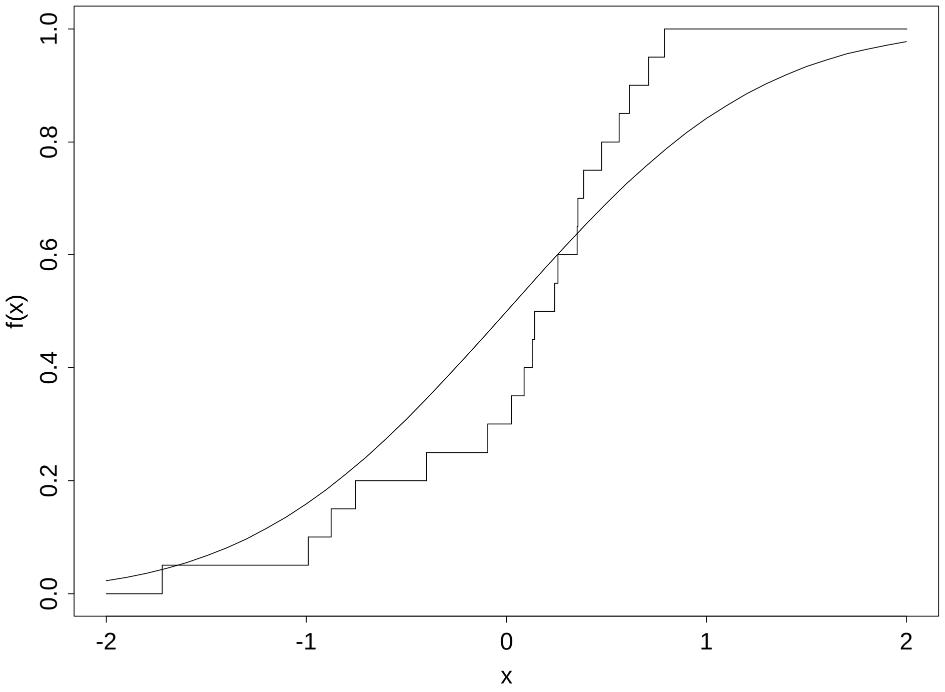

Figure 4.1: Comparison of true and empirical distribution functions

Algorithm 4.1

1. *Evaluate the empirical distribution function* $\hat{F}$.
2. *Simulate* B *samples* $\mathbf{x}^{*\mathbf{1}}, \mathbf{x}^{*\mathbf{2}}, \ldots, \mathbf{x}^{*\mathbf{B}}$ *each of size* n *from* $\hat{F}$.
3. *Estimate the population bootstrap variance as the sample variance*

$$Var_B(\hat{\theta}) = \frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}(\mathbf{x}^{*\mathbf{b}}) - \theta^*)^2 \tag{4.2}$$

where

$$\theta^* = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}(\mathbf{x}^{*\mathbf{b}}) \tag{4.3}$$

So, there are two aspects to the procedure: replacement of the distribution function by its empirical counterpart; and simulation from the empirical distribution to estimate statistic characteristics.

This is all extremely easy to implement in Splus using the following simple function:

```
bootstrap1_function(x, nboot, theta, ...)
{
        z <- list()
        data <- matrix(sample(x, size = length(x) * nboot, replace = T), nrow
                 = nboot)
        bd <- apply(data, 1, theta, ...)
        est <- theta(x, ...)
        z$est <- est
        z$distn <- bd
        z$bias <- mean(bd) - est
        z$se <- sqrt(var(bd))
        z
}
```

where `theta` is the function we wish to bootstrap. This returns a list giving the estimate, the bootstrap sample, the bootstrap estimate of bias (see below) and the bootstrap estimate of standard error.

Example 4.1 *MINITAB gives a data set representing the observed daily activity time, in hours, of a group of salamanders. These data are stored in the vector* `sal.dat`.

A histogram of the data is shown in Figure 4.2.

Suppose then that we are interested in the median activity time of the population, assuming that the observed sample is a random sample from this population. The sample median is $\hat{\theta} = 0.8$ hours. We now bootstrap to assess the accuracy of this estimate, using the Splus command

```
bootstrap1(sal.dat,200,median)
```

which gives 200 realisations from the bootstrap distribution of the sample median. A histogram of the bootstrap realisations of the sample median is given in Figure 4.3. The standard deviation of these simulations is 0.255, which is therefore our bootstrap standard error for $\hat{\theta}$.

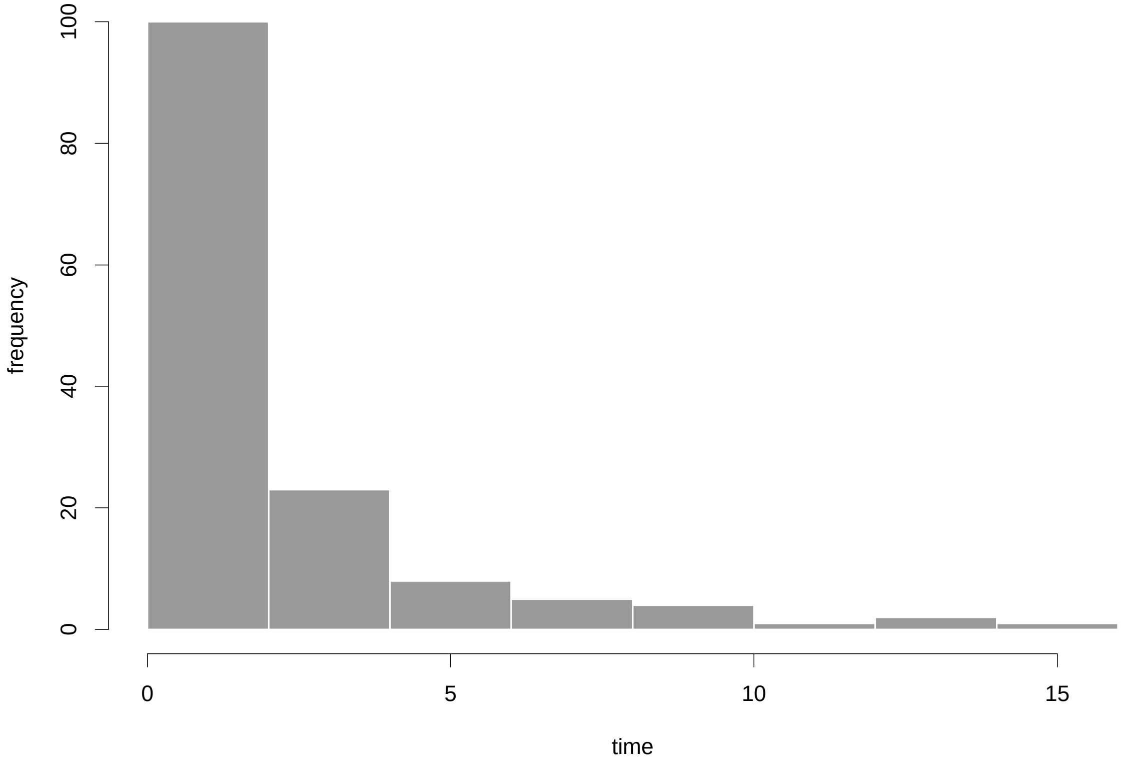

Figure 4.2: Histogram of salamander data

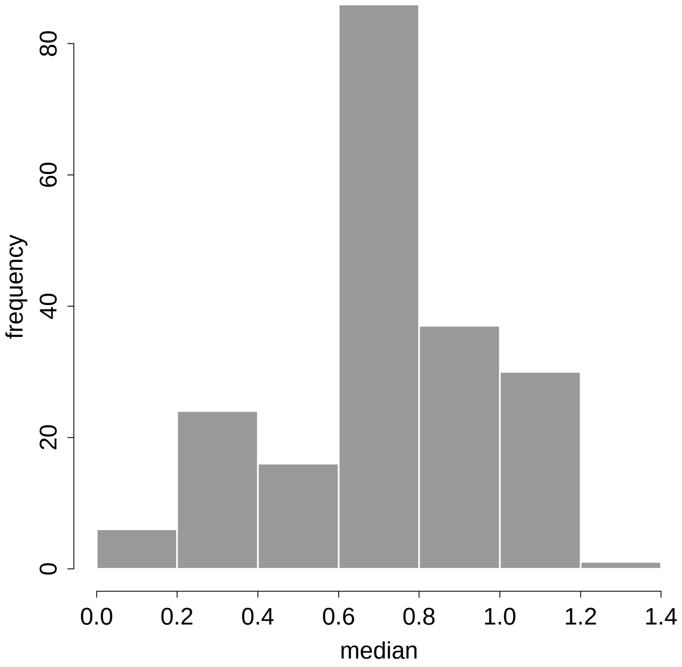

Figure 4.3: Histogram of bootstrap sample medians

Example 4.2 *The above method applies almost as easily to slightly more complicated data sets. We consider here a set of data (given by Efron) relating two score tests, LSAT and GPA, at a sample of 15 American law schools. Of interest is the correlation between these measurements.*

The data are given as

LSAT	GPA
576	3.39
635	3.30
558	2.81
578	3.03
666	3.44
580	3.07
555	3.00
661	3.43
651	3.36
605	3.13
653	3.12
575	2.74
545	2.76
572	2.88
594	2.96

and are plotted in Figure 4.4. They are stored in Splus as the matrix `law.dat`. Estimating the population correlation by the sample correlation gives $\hat{\theta} = 0.776$, but how accurate is this estimate? We can assess this using the bootstrap algorithm. In this case we wish to sample from the rows of the data matrix; this requires care in the use of the above Splus function, since there it was anticipated that `x` would be a vector. We get round this problem by setting `x` to be the vector `1:n`, in this case, `1:15`. Then the data matrix is passed to the function `theta`, which in this case evaluates the correlation coefficient, as an additional argument. Thus, we need

```
bootstrap1(1:15,nboot,theta1,law.dat)
```

where

```
theta1_function(x,xdata){
cor(xdata[x,1],xdata[x,2])
}
```

A histogram of 200 bootstrap simulations of θ is given in Figure 4.5 and the sample standard error of these values is found to be 0.135. Note in particular the skewness of the bootstrap distribution.

4.2 Other bootstrap estimates

We have focused so far on the bootstrap estimate of variance, or standard error. However, since we have shown in the previous examples how the entire bootstrap sample is generated, it is clear how other properties of the sampling distribution of $\hat{\theta}$ can be examined.

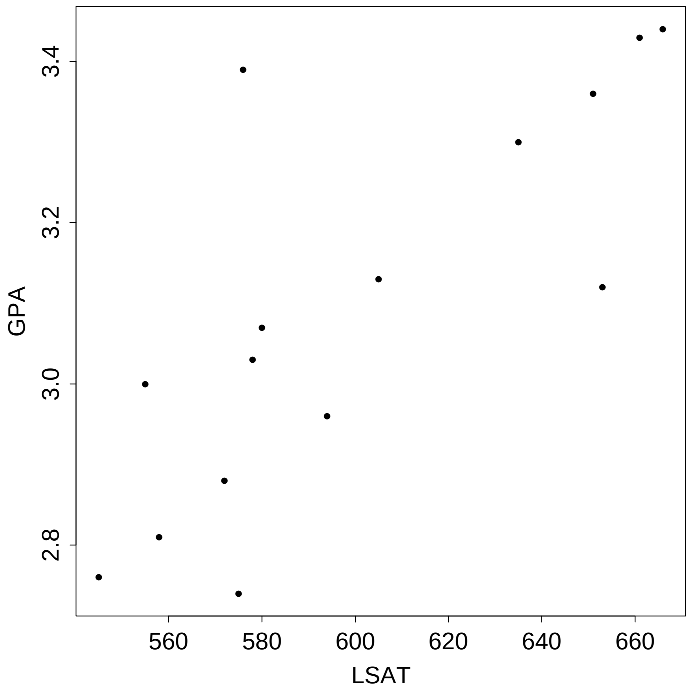

Figure 4.4: Scatterplot of law school data

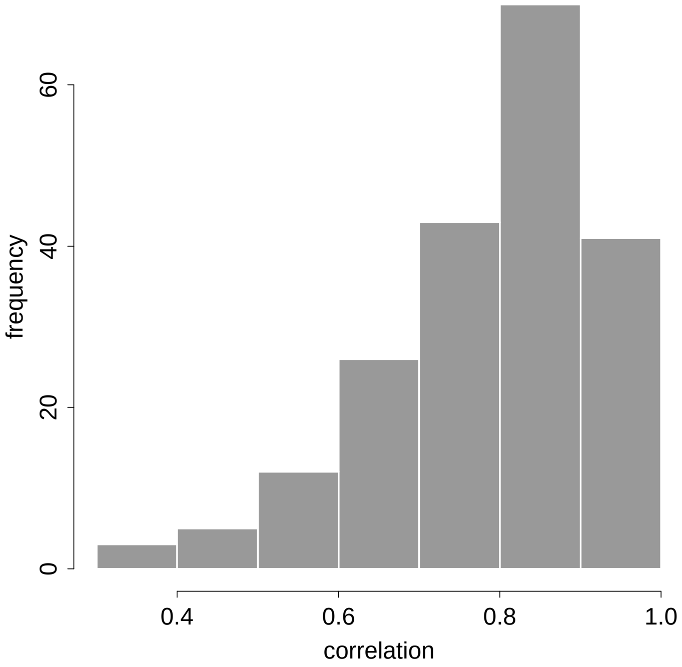

Figure 4.5: Histogram of bootstrap correlations

The bias of an estimator $\hat{\theta}$ of θ is defined as

$$bias = E(\hat{\theta}) - \theta \tag{4.4}$$

We define the bootstrap estimate of bias to be

$$bias = E_{\hat{F}}(\hat{\theta}) - \hat{\theta} \tag{4.5}$$

That is, the difference between the expected value of the bootstrap distribution, and the estimated value. Of course, we must again estimate $E_{\hat{F}}(\hat{\theta})$ from the bootstrap sample as

$$E_{\hat{F}}(\hat{\theta}) \approx \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}(\mathbf{x}^{*\mathbf{b}}) \tag{4.6}$$

For the salamander data we have $\frac{1}{B}\sum_{b=1}^{B}\hat{\theta}(\mathbf{x}^{*\mathbf{b}}) = 0.800$ and so the bootstrap estimate of bias is 0. For the law school data, the mean of the bootstrap simulations is 0.786, giving a bootstrap estimate of bias of $0.786 - 0.776 = 0.010$, a small enough value not to give cause for concern.

This is not the most efficient way to estimate bias: a more efficient procedure related to the idea of control variates in simulation is described by Efron and Tibshirani (Chapter 23).

4.3 Structured data

The bootstrap procedure is easily extended to models of more detailed structure. Here we look at a few examples to illustrate the general principle.

4.3.1 Comparing two populations

Example 4.3 *Figure 4.6 gives histograms of historical and current measurements of Ph levels at each of 149 lakes in Wisconsin. The data are stored respectively in* `ph1.dat` *and* `ph2.dat`.

Historical data from 25 of the lakes are missing, so paired sample comparisons are not possible. We will compare the difference in medians of the historical and current populations based on $\hat{\theta}$, the difference in sample medians, which turns out to be 0.422. Each bootstrap simulation consists of simulating from the empirical distribution function of the two separate samples, obtaining the median of each and differencing. Repeating this B times gives the full bootstrap distribution.

The following Splus function `bootstrap2` gives the necessary modification to the original function to achieve this procedure.

```
bootstrap2_function(x, y, nboot, theta, ...)
{
        z <- list()
        data.x <- matrix(sample(x, size = length(x) * nboot, replace = T), nrow
                 = nboot)
        data.y <- matrix(sample(y, size = length(y) * nboot, replace = T), nrow
                 = nboot)
        data <- cbind(data.x, data.y)
        bd <- apply(data, 1, theta, ...)
```

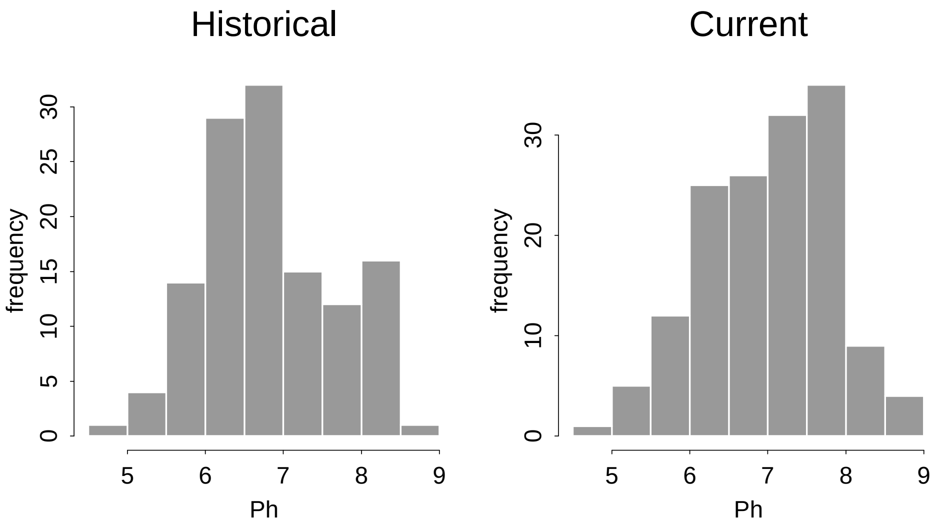

Figure 4.6: Histograms of historical and current Ph levels in Wisconsin lakes

```
        est <- theta(c(x, y), ...)
        z$est <- est
        z$distn <- bd
        z$bias <- mean(bd) - est
        z$se <- sqrt(var(bd))
        z
}
```

This is simply applied with the data from the two series in x and y respectively, and with n as the length of x. Thus, in this case, we apply the function with

```
theta2_function(z, n)
{
        median(z[(n + 1):length(z)]) - median(z[1:n])
}
```

Applying this with nboot = 200 gives the histogram of bootstrap values in Figure 4.7.

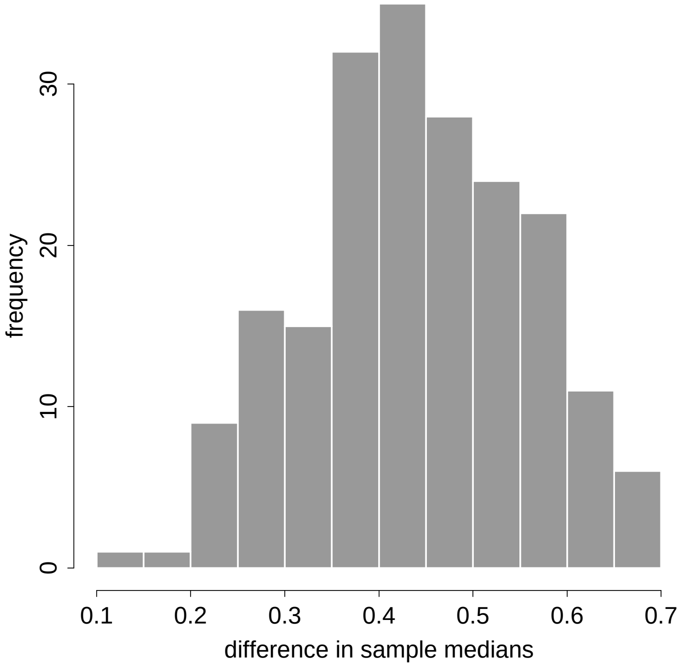

Figure 4.7: Histogram of simulated median differences

The mean of the simulated values is 0.440, suggesting low bias, and the standard error is 0.112.

4.3.2 Time series

Example 4.4 *The time series, z_t, in Figure 4.8 is taken from Diggle (1990, Time Series: a biostatistical introduction) and relates to a time series of lutenizing hormone measurements after* *relocation by subtraction of sample mean. The data are stored in* `hormone.dat`.

The objective is to model this time series and use bootstrapping to assess the accuracy of the fitted model. The challenge is to find a way of bootstrapping from the sample series to create new series which have the same stochastic structure as the original series. We will illustrate the procedure using a simple AR(1) model, though this isn't necessarily the best model for these data. The residuals from an AR(1) fit are shown in Figure 4.9; the evident skewness suggests that estimates of variance based on the usual Gaussian error model are likely to be invalid. Thus, bootstrapping provides an alternative data–based estimate of parameter precision. One approach is given by the following algorithm:

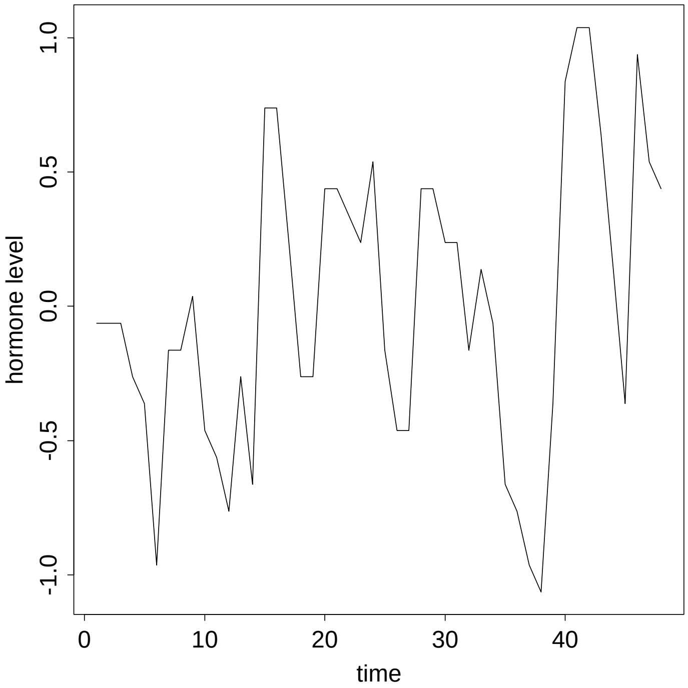

Figure 4.8: Time series of lutenizing hormone level

Algorithm 4.2

1. *Fit the AR(1) model to the data:* $z_t = \hat{\beta} z_{t-1} + \epsilon_t$
2. *Obtain the residuals* e_t *from the fitted model.*
3. *Obtain the empirical distribution function* $\hat{F}$ *of the residuals.*
4. *Simulate B bootstrap replications of the series by:*
   (a) *Set* $y_1 = z_1$.
   (b) *Evaluate recursively* $y_t = \hat{\beta} y_{t-1} + \epsilon_t^*$, *where* ϵ_t^* *is a realization from* $\hat{F}$.

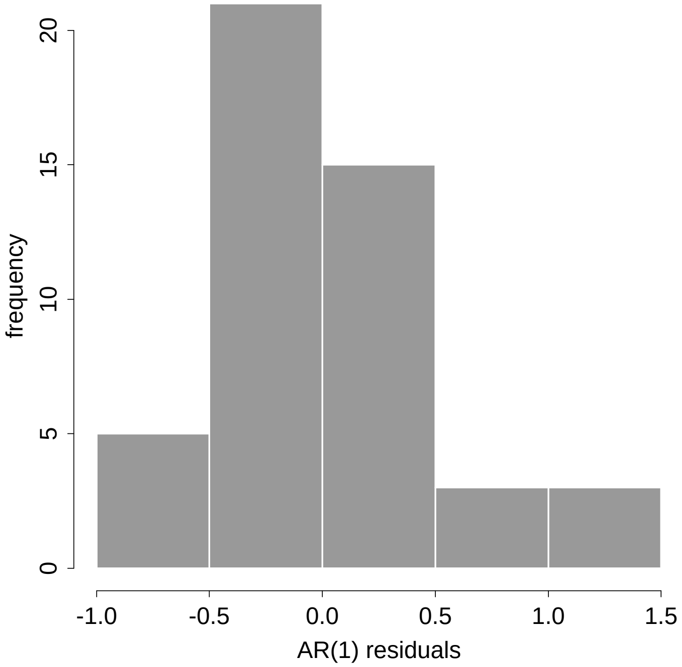

Figure 4.9: Histogram of residuals of AR(1) fit

5. *Re–fit the AR(1) model to each bootstrapped time series to obtain B bootstrap realisations of $\hat{\beta}$.*

6. *Summarize the required characteristics of the bootstrap distribution.*

In this way, the bootstrap realizations of the time series are forced to have the correct model structure. Within Splus, this is all easily achieved using the `bootstrap1` function called as

```
bootstrap1(1:47,200,theta3,resid,beta,z1)
```

where `beta` and `z1` are the estimated AR coefficient and the first value of the time series respectively, and `theta3` is defined as

```
theta3_function(x, resid, beta, z1)
{
        y <- NULL
        y[1] <- z1
        for(i in 1:length(resid)) {
                y[i + 1] <- beta * y[i] + resid[x[i]]
        }
        ar.fit <- ar(y, aic=F, ord = 1)
        plot(y, type = "l")
        ar.fit$ar
}
```

For this example, Figure 4.10 shows 9 bootstrap realisations of the series.

Figure 4.11 gives the histogram of 200 bootstrap realizations of $\hat{\beta}$, the AR(1) coefficient. The mean and standard deviation of the bootstrap distribution are 0.504 and 0.133 respectively, suggesting that $\hat{\beta}$ is a somewhat biased estimator of β (bootstrap bias $= .504 - .58 = -.076$) and a bootstrap standard error of 0.133.

This is not the only method which has been proposed for bootstrapping a time series. Another method based on blocking data and bootstrapping the blocks has also been proposed in the literature.

4.3.3 Non–parametric regression

Here we consider a well–studied data set of Silverman (1985), giving measurements of acceleration against time for a simulated motorcycle accident. The data are shown in Figure 4.12. Clearly the relationship is non–linear, and has structure that will not easily be modelled parametrically. Splus provides a number of different routines for non–parametric regression — here we will use just one: `loess`. The details don't matter too much, but in outline the fit is a locally weighted least–squares regression. The smoothing is determined by an argument `span`, which determines the proportion of data to be included in the moving window which selects the points to be regressed on. With `span = 1/3`, the `loess` fit to the motorcycle data is included in Figure 4.12.

Because of the non–parametric structure of the model, classical approaches to assessment of parameter precision are not available. The challenge again in bootstrapping is to create bootstrap realisations of the data which have similar stochastic structure to the original series. This is achieved simply by bootstrapping the pairs (x, y) in the original plot and fitting `loess` curves to each simulated bootstrap series. In Splus we simply use

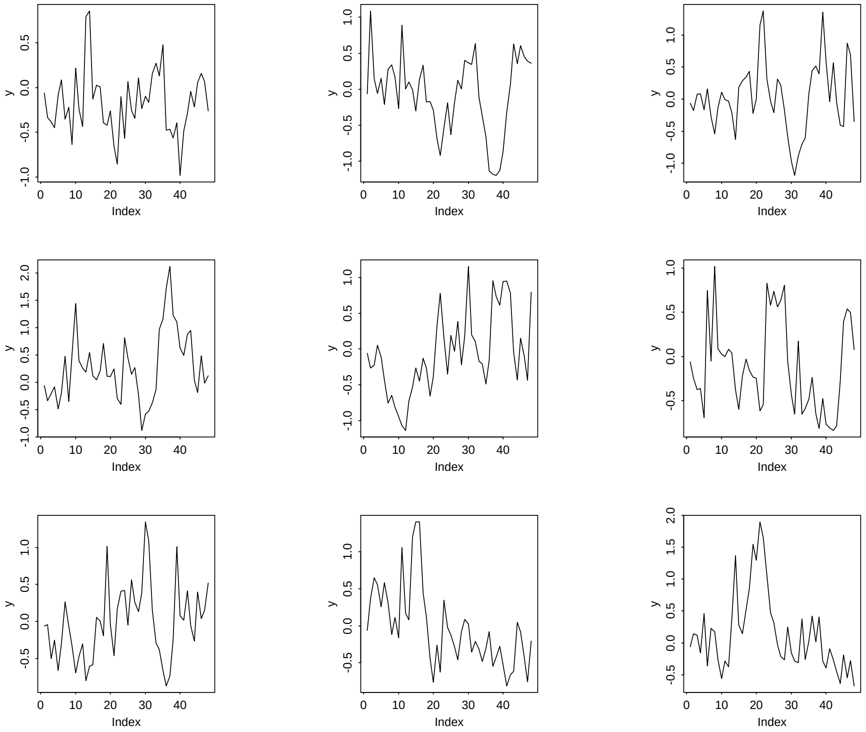

Figure 4.10: Bootstrap time series realisations of the lutenizing hormone level data

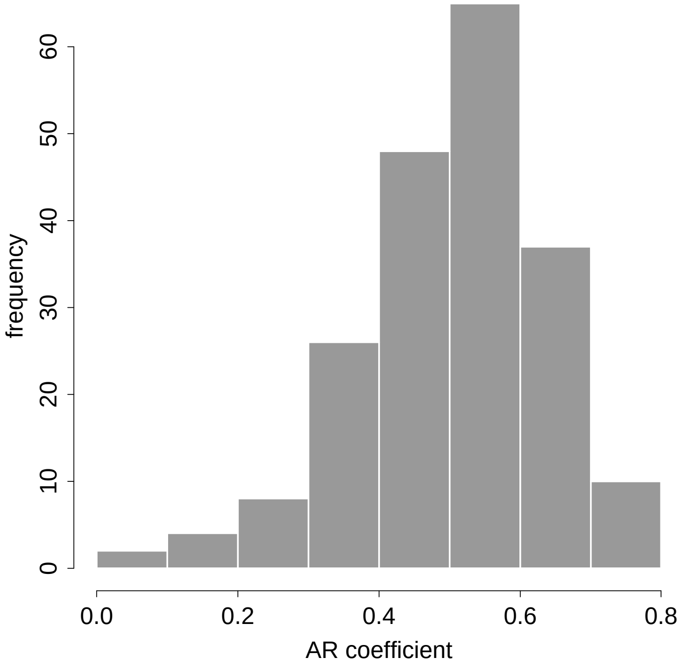

Figure 4.11: Histogram of bootstrap distribution of AR(1) coefficient

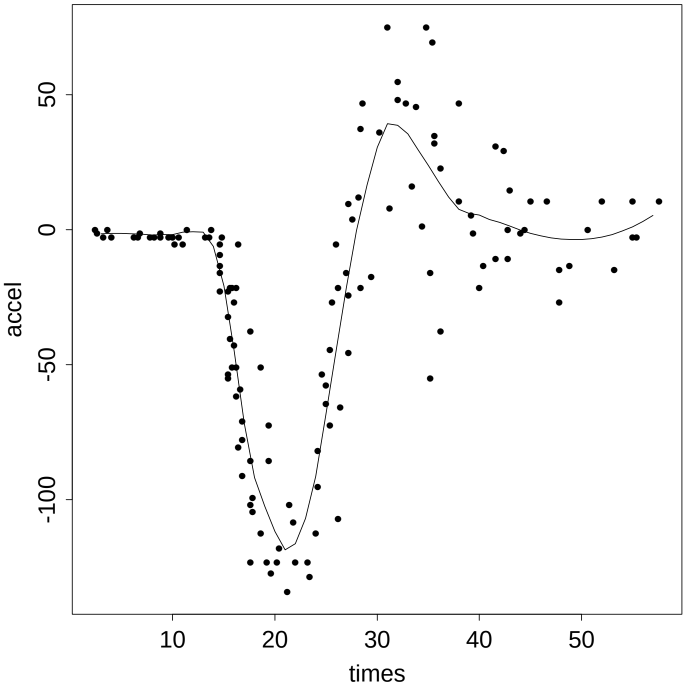

Figure 4.12: Scatterplot of motorcycle data

```
bootstrap1(1:133,20,theta4,mc.dat)
```

where `mc.data` contains the data, and

```
theta4_function(x, xdata)
{
        mc.lo <- loess(xdata[x, 2] ~ xdata[x, 1], span = 1/3)
        y <- predict.loess(mc.lo, 1:60)
        lines(1:60, y)
}
```

Applying this to the motorcycle data gave the `loess` curves in Figure 4.13. In this way we can assess both pointwise and globally the variability in the original `loess` fit.

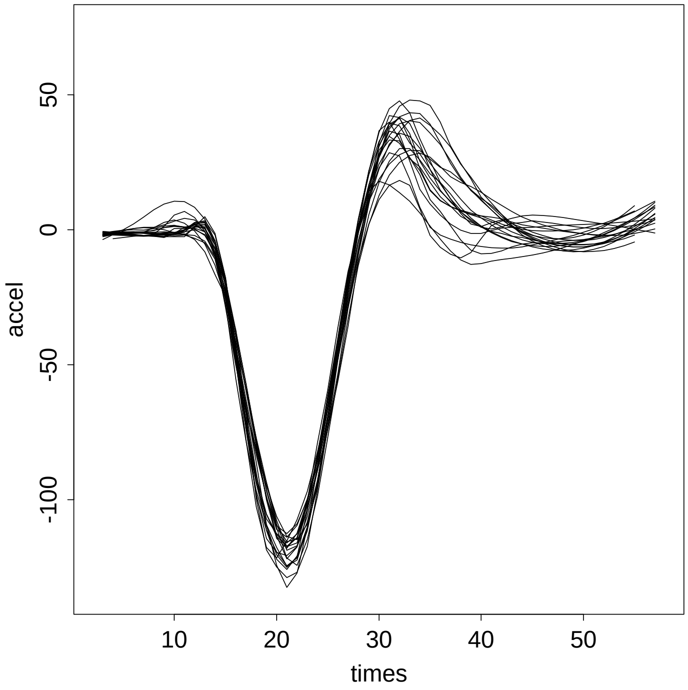

Figure 4.13: Bootstrap realisations of loess fits to motorcycle data

4.4 Regression

A more formal regression–type model has the structure

$$y_i = f(x_i; \beta) + \epsilon_i \tag{4.7}$$

where f is a specified function acting on the covariates, x_i, with parameters β, and ϵ_i is a realisation from a specified error structure. With this model framework there are two alternative ways to bootstrap the model:

1. Bootstrap from the pairs (x_i, y_i), re–fit the model to each realisation, form the bootstrap distribution of β (this is the approach we used in the non–parametric regression example). Or;

2. Fit the regression model, form the empirical distribution function of the residuals, generate bootstrap replications of the data by substitution into (4.7), re–fit the model to each realisation to obtain bootstrap distribution of β.

To illustrate these techniques on a simple data set, we'll use the data shown in Figure 4.14, for which we assume a simple model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \tag{4.8}$$

without making any distributional assumptions about the ϵ_i. A least squares fit to these data gives $\hat{\beta}_0 = -1160.5$ and $\hat{\beta}_1 = 57.51$.

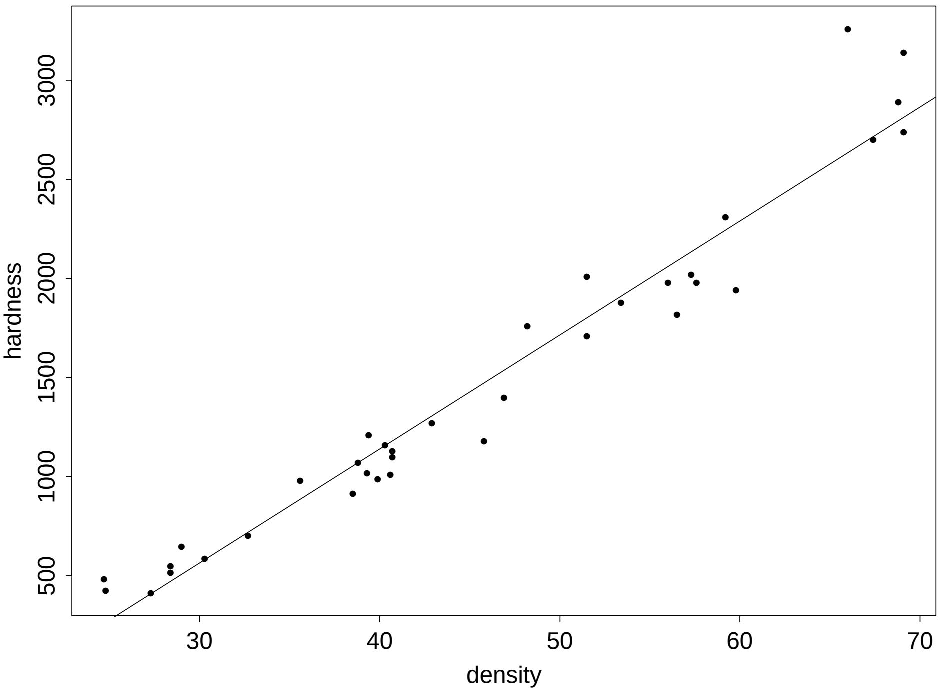

Figure 4.14: Hardness versus density of Australian timbers

To apply the first approach, we use the `bootstrap1` function with `theta5` defined as:

```
theta5_function(x, xdata)
{
        ls <- lsfit(xdata[x, 1], xdata[x, 2])
        abline(ls)
        ls$coef
}
```

A graph of 10 bootstrap realisations of the regression line is given in Figure 4.15.

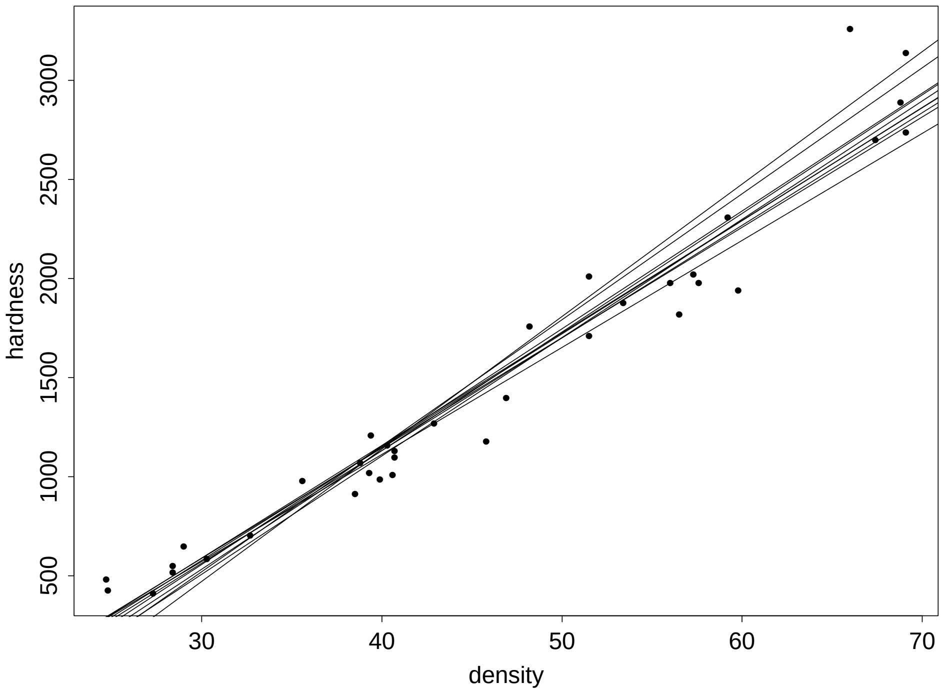

Figure 4.15: Bootstrap regression lines

The bootstrap distribution of the two regression parameters is given in Figure 4.16. The means and standard deviations of these empirical distributions are $(-1157.5, 57.3)$ and $(115.7, 2.91)$ respectively.

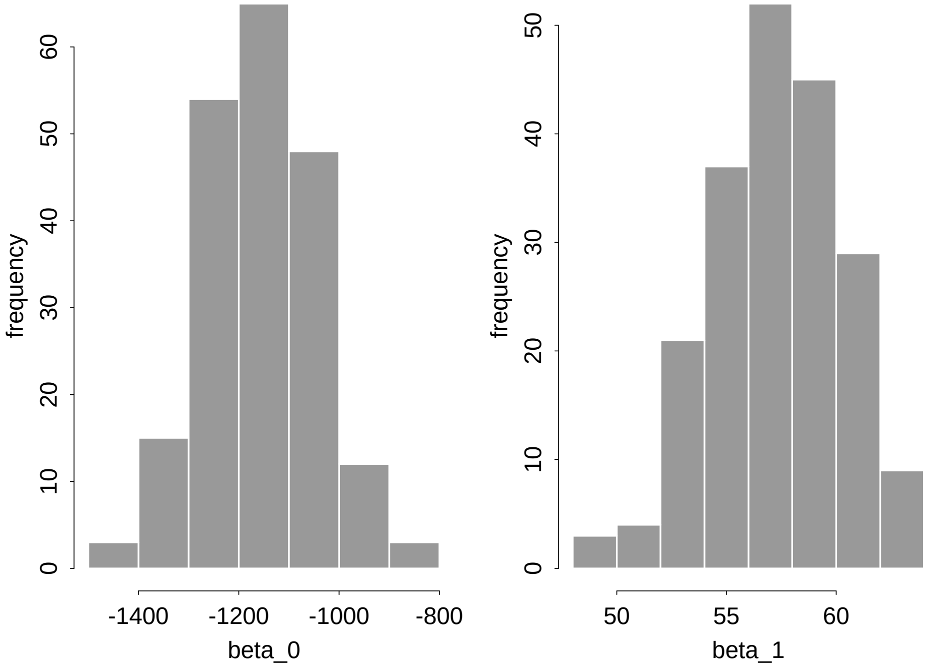

Figure 4.16: Bootstrap distributions of regression parameters

To apply the second method, we call the **bootstrap1** function as

```
bootstrap1(resids,10,theta6,xdata=wood.dat)
```

where **resids** contains the residuals from the original fit, with **theta6** defined as

```
theta6_function(x, xdata)
{
        y <- x + xdata[, 2]
        ls <- lsfit(xdata[, 1], y)
        abline(ls)
        ls$coef
}
```

Plots corresponding to Figures 4.15 and 4.16 are given in Figures 4.17 and 4.18. In this case the respective means and standard deviations are $(-1158.8, 57.5)$ and $(96.1, 2.04)$.

The point about this is that there is less variability using the second approach, but the price for this is that the accuracy of the bootstrap distributions is more dependent on the accuracy of the linear fit. Thus if there is uncertainty about the adequacy of the specified model structure, then bootstrapping the pairs is more robust than bootstrapping the residuals.

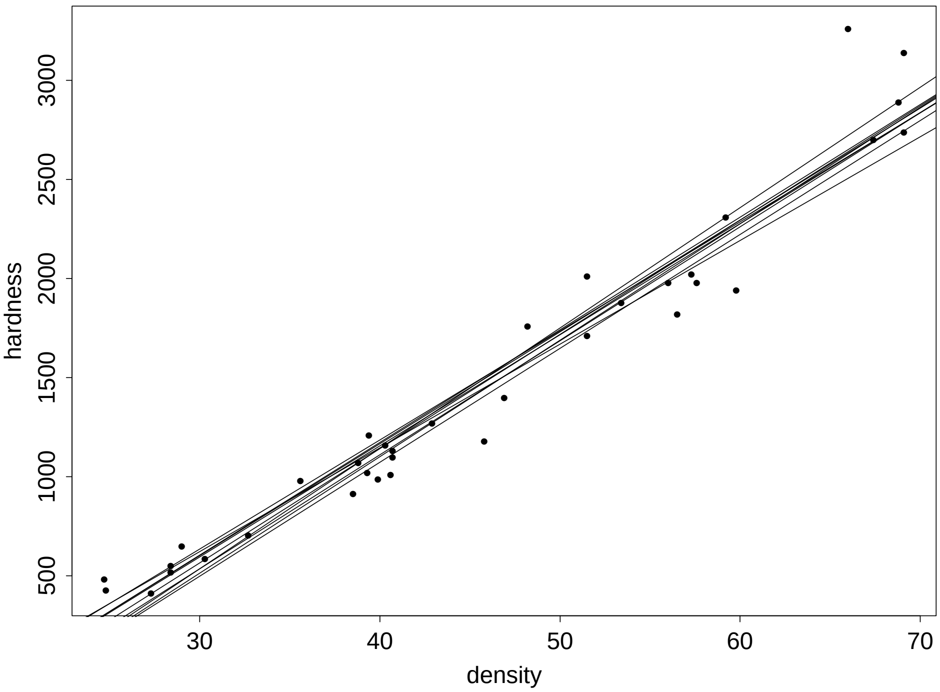

Figure 4.17: Bootstrap regression lines using second approach

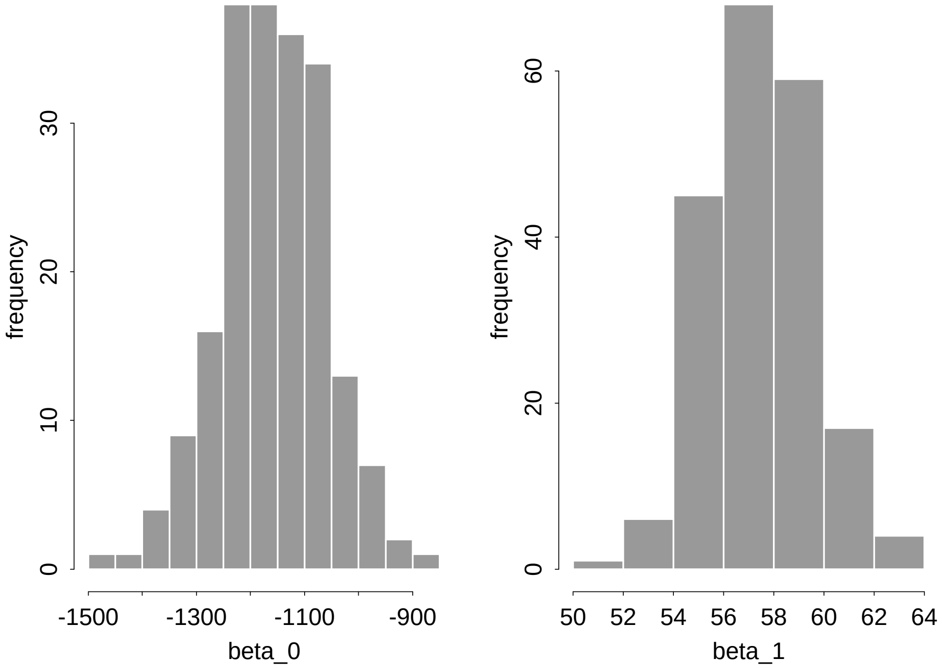

Figure 4.18: Bootstrap distributions of regression parameters using second approach

4.5 Bootstrap confidence intervals and tests

Since the technique of bootstrapping gives a bootstrap sample of the parameter(s) of interest, it is obvious how to use this sample to produce either confidence intervals of the parameter (using quantiles of the sample output), or hypothesis tests. It should be noted however, that to achieve reasonable accuracy, it may be necessary to produce a bootstrap sample of substantially greater size than was necessary in just producing bootstrap estimates of variability. There are a number of modifications which can be made to improve bootstrap confidence intervals, which either correct for bias or speed up the procedure. Efron and Tibshirani sketch the details for this, and also supply Splus code to enact the procedures.

4.6 The jackknife

We mention briefly that bootstrapping isn't the only scheme for estimating the statistical properties of estimators by resampling. A slightly older idea is that of jackknifing. The idea is similar to bootstrapping, but instead of resampling from the original sample, we apply the estimator systematically to all samples obtained from the original sample with just one sample value deleted.

Thus, for example, if our original sample were $x_1, x_2, \dots, x_n$ then the jackknife samples would be $(x_2, \dots, x_n), (x_1, x_3, \dots, x_n), \dots (x_1, x_2, \dots, x_{n-1})$. The corresponding jackknife estimate values are $\hat{\theta}_{(1)}, \hat{\theta}_{(2)}, \dots \hat{\theta}_{(n)}$. We then define the jackknife estimates of bias and variance respectively by

$$\hat{bias}_{jack} = (n-1)(\hat{\theta}_{(.)} - \hat{\theta}) \tag{4.9}$$

and

$$\hat{\mathrm{Var}}_{jack} = \left(\frac{n-1}{n}\right) \sum (\hat{\theta}_{(i)} - \hat{\theta}_{(.)})^2 \tag{4.10}$$

where

$$\hat{\theta}_{(.)} = \frac{1}{n} \sum \hat{\theta}_{(i)} \tag{4.11}$$

The multipliers $(n-1)$ and $(n-1)/n$ in these expressions might seem surprising, but are necessary because the operation of jackknifing (unlike bootstraping) generates samples which are more homogeneous than samples from the original population would be. Thus it is necessary to multiply by these inflation factors to compensate for this. The actual values are chosen to give accurate jackknife estimates in simple examples (such as the sample mean) for which analytical results are available.

A simple Splus function for jackknifing, which has much the same structure as the corresponding `bootstrap1` function, is given as follows:

```
jackknife_function(x, theta, ...) {
        z <- list()
        n_length(x)
        jd <- NULL
        for(i in 1:n) jd[i] <- theta(x[ - i], ...)
        est_theta(x,...)
        z$est_est
        z$dist <- jd
        z$bias <- (mean(jd) - est)*(n-1)
        z$se_sqrt(var(jd)*(n-1)^2/n)
        z
}
```

Chapter 5

The EM algorithm

5.1 Introduction

The EM algorithm is a technique highly related to the data augmentation algorithm in MCMC. However the EM algorithm (in its basic form at least) is a deterministic algorithm, and whereas the data augmentation algorithm simulated alternatively from the missing data and the parameters according to the appropriate conditional distributions, the EM algorithm replaces the simulation steps by an expectation and a maximisation step. The algorithm is applied to the likelihood function and converges to a local maximum of the likelihood of the observed data. Therefore where the likelihood is unimodal it converges to the MLE. Thus it is usually used as an algorithm to estimate the MLE of a statistical problem with missing data. However, it can also be applied in a Bayesian context. For instance by assuming a 'flat' prior on the unknown parameters, the EM algorithm can be used to try and approximate the mode of the posterior distribution.

Typically though, the EM algorithm is used to maximize a likelihood when, with respect to the likelihood specification, some of the data give incomplete information. This might be the case, for example, in a regression model where some of the data is censored. If the data were all complete, maximum likelihood would be a straightforward application of least squares, but the censoring complicates this. The EM algorithm solves this problem iteratively by first using a current estimate of the regression model to estimate the values of the censored data; then re–estimating the regression line on the basis of the original and augmented data. This procedure is then iterated until convergence. The two steps are respectively called **the E–step** (estimation) and **the M–step** (maximization).

5.2 The algorithm

We suppose that our aim is to find the maximum likelihood estimator of $L(\theta|Y)$, where Y represents our observed data. Suppose that this is hard to do directly but that it is possible to augment the observed data Y with additional data Z, such that the augmented likelihood $L(\theta|Y,Z)$ is easier to maximize. In essence, we then estimate Z from a current estimate of θ and then maximize $L(\theta|Y,Z)$ with respect to θ. Then iterate to convergence. More formally, given a current estimate θ^i of θ, we define the function

$$Q(\theta,\theta^i) = \int_Z \log(L(\theta|Y,Z))P(Z|\theta^i,Y)dZ \tag{5.1}$$

that is

$$Q(\theta, \theta^i) = E(\log(L(\theta|Y, Z)) \tag{5.2}$$

where expectation is with respect to the distribution of Z given the current estimate of θ and Y, i.e., $P(Z|\theta^i, Y)$. The EM algorithm is defined formally as:

Algorithm 5.1

1. **The E–Step**: *Calculation of* Q;
2. **The M–Step**: *Maximization of* Q *with respect to* θ.

Implicit within the Q–step is the estimation of the augmented data with respect to the current model estimate and complete data Y. This estimation simply involves taking expectations with respect to the current model (conditional on the observed data). In comparison with the data augmentation algorithm, both of the sampling steps are replaced by deterministic steps (an expectation and a maximization).

The EM always increases the likelihood $L(\theta|Y)$. To show this we first note the following inequality that holds for all densities f and g by Jensen's inequality (see for example Grimmett and Stirzaker, Probability and Random Processes).

$$E_g\left[\log\frac{f(X)}{g(X)}\right] \leq 0\ . \tag{5.3}$$

A consequence of this, letting f be $P(Z|\theta^{i+1}, Y)$ and g be $P(Z|\theta^i, Y)$ is

$$\int \log\left(\frac{P(Z|\theta^{i+1}, Y)}{P(Z|\theta^i, Y)}\right) P(Z|\theta^i, Y)dZ \leq 0\ . \tag{5.4}$$

Since the second step of the EM algorithm maximizes $Q(\theta, \theta^i)$ as a function of its first argument, $Q(\theta^{i+1}, \theta^i) \geq Q(\theta^i, \theta^i)$, so that

$$\int \log\left(\frac{P(Y, Z|\theta^{i+1})}{P(Y, Z|\theta^i)}\right) P(Z|\theta^i, Y)dZ \geq 0\ . \tag{5.5}$$

Now subtracting (5.4) from (5.5), we get

$$\int \log\left(\frac{P(Y, Z|\theta^{i+1})}{P(Z|\theta^{i+1}, Y)}\frac{P(Z|\theta^i, Y)}{P(Y, Z|\theta^i)}\right) P(Z|\theta^i, Y)dZ \geq 0\ . \tag{5.6}$$

But the two terms multiplied together are just $P(Y|\theta^{i+1})$ and $1/P(Y|\theta^i)$ respectively and these do not depend on Z, so we get

$$\log P(Y|\theta^{i+1}) \geq \log P(Y|\theta^i) \tag{5.7}$$

so that the likelihood is always increased.

5.3 A genetic example

We return to the genetic example of Chapter 2. Recall the data set contains information concerning the genetic linkage of 197 animals, in which the animals are distributed into 4 categories:

$$Y = (y_1, y_2, y_3, y_4) = (125, 18, 20, 34) \tag{5.8}$$

with cell probabilities

$$(\frac{1}{2} + \frac{\theta}{4}, \frac{1}{4}(1-\theta), \frac{1}{4}(1-\theta), \frac{\theta}{4}) \tag{5.9}$$

Though it is by no means impossible to maximize this multinomial likelihood directly, we illustrate how the EM agorithm brings about a substantial simplification, by using the same augmentation method that was used in the data augmentation case. Specifically, we augment the observed data Y by dividing the first cell into two, with respective cell probabilities $\frac{1}{2}$ and $\frac{\theta}{4}$, giving an augmented data set $X = (x_1, x_2, x_3, x_4, x_5)$, where $x_1 + x_2 = y_1$, and $x_3 = y_2, x_4 = y_3, x_5 = y_4$. Now we have

$$L(\theta|Y) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4} \tag{5.10}$$

whereas

$$L(\theta|X) \propto \theta^{x_2+x_5}(1-\theta)^{x_3+x_4} \tag{5.11}$$

Thus we obtain,

$$\begin{aligned} Q(\theta, \theta^i) &= E((x_2+x_5)\log(\theta) + (x_3+x_4)\log(1-\theta)|\theta^i, Y) \\ &= [E(x_2|\theta^i, Y) + x_5]\log(\theta) + (x_3+x_4)\log(1-\theta) \end{aligned}$$

where $x_2|\theta^i, Y \sim \text{Bin}(125, \frac{\theta^i}{\theta^i+2})$. Thus

$$Q(\theta, \theta^i) = \left[\frac{125\theta^i}{\theta^i+2} + 34\right]\log(\theta) + 38\log(1-\theta) \tag{5.12}$$

This is easily maximized to give:

$$\theta^{i+1} = \frac{E(X_2|\theta^i, Y) + x_5}{E(X_2|\theta^i, Y) + x_3 + x_4 + x_5} \tag{5.13}$$

where $E(X_2|\theta^i, Y) = \frac{125\theta^i}{\theta^i+2}$.

Thus, the alternation between estimation and maximization is clearly seen. This can all be easily carried out with a pocket calculator. Starting with $\theta^1 = 0.5$ we obtain the sequence in Table 5.1. Hence the maximum likelihood estimate (posterior mode) is $\theta = 0.6268$.

i	θ^i
1	0.5
2	0.6082
3	0.6243
4	0.6265
5	0.6268
6	0.6268

Table 5.1: Sequence of EM iterates

150	170	190	220
8064*	1764	408	408
8064*	2772	408	408
8064*	3444	1344	504
8064*	3542	1344	504
8064*	3780	1440	504
8064*	4860	1680*	528*
8064*	5196	1680*	528*
8064*	5448*	1680*	528*
8064*	5448*	1680*	528*
8064*	5448*	1680*	528*

Table 5.2: Censored regression data

5.4 Censored regression example

Here we consider a regression problem involving censored data. Consider the data in Table 5.2. These data represent failure times (hours) of motorettes at four different temperatures (Celsius). The asterisks denote a censored observation, so that for example, 8064* means the item lasted *at least* 8064 hours.

Physical considerations suggest the following model relating the logarithm (base 10) of lifetime (t_i) and $v_i = 1000/(\text{temperature } + 273.2)$:

$$t_i = \beta_0 + \beta_1 v_i + \epsilon_i \tag{5.14}$$

where $\epsilon_i \sim N(0, \sigma^2)$ On this scale a plot of t_i against v_i is given in Figure 5.1 in which censored data are plotted as open circles. The raw data are contained in `motor.dat`, and the transformed data are in `motor2.dat`. In each case there is an additional column representing an indicator variable, which takes the value 1 if the observation has been censored and 0 otherwise.

Now, in this situation, if the data were uncensored we would have a simple regression problem. Because of the censoring we use the EM algorithm to first estimate where the censored values actually are (the E–step) and then fit the model to the augmented data set (the M–step). Re–ordering the data so that the first m values are uncensored, and denoting by Z_i the augmented data values, we have the augmented log–likelihood (or log–posterior with a flat prior):

$$\log(L(\theta|V,Z)) = -n\log\sigma - \sum_{i=1}^{m}(t_i-\beta_0-\beta_1 v_i)^2/2\sigma^2 - \sum_{i=m+1}^{n}(Z_i-\beta_0-\beta_1 v_i)^2/2\sigma^2 \tag{5.15}$$

The E–step requires us to find expectations of $P(Z_i|\beta_0,\beta_1,\sigma,c_i)$ where c_i denotes the censored time. Unconditionally on the censoring, the distribution of each Z_i is normal, so the conditioning is a normal probability, conditioned on $Z_i > c_i$. Thus,

$$\begin{aligned} Q = -n\log\sigma &- \frac{1}{2\sigma^2}\sum_{i=1}^{m}(t_i-\beta_0-\beta_1 v_i)^2 - \frac{1}{2\sigma^2}\sum_{i=m+1}^{n}[E(Z_i^2|\beta_0,\beta_1,\sigma,Z_i>c_i) \\ &- 2(\beta_0+\beta_1 v_i)E(Z_i|\beta_0,\beta_1,\sigma,Z_i>c_i) + (\beta_0+\beta_1 v_i)^2]. \end{aligned}$$

Furthermore, it is fairly straightforward to show that

$$E(Z_i|\beta_0,\beta_1,\sigma,Z_i>c_i) = \mu_i + \sigma H\left(\frac{c_i-\mu_i}{\sigma}\right) \tag{5.16}$$

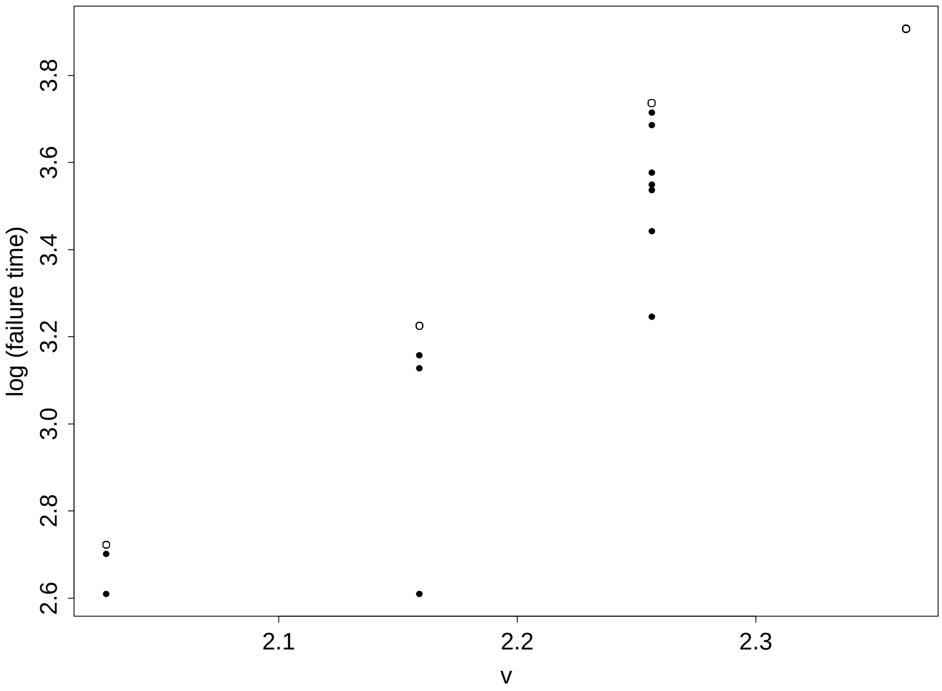

Figure 5.1: Censored regression data

and

$$E(Z_i^2|\beta_0, \beta_1, \sigma, Z_i > c_i) = \mu_i^2 + \sigma^2 + \sigma(c_i + \mu_i)H\left(\frac{c_i - \mu_i}{\sigma}\right) \tag{5.17}$$

where $\mu_i = \beta_0 + \beta_1 v_i$ and $H(x) = \phi(x)/\{1 - \Phi(x)\}$.

It follows that maximizing Q amounts to solving the usual normal equations for β_0 and β_1, but that maximization with respect to σ requires solution of the equation:

$$\begin{aligned} &\frac{\sum_{i=1}^m (t_i - \beta_0 - \beta_1 v_i)^2}{\sigma^4} + \\ &\frac{\sum_{i=m+1}^n [E(Z_i^2|\beta_0, \beta_1, \sigma, Z_i > c_i) - 2(\beta_0 + \beta_1 v_i)E(Z_i|\beta_0, \beta_1, \sigma, Z_i > c_i) + (\beta_0 + \beta_1 v_i)^2]}{\sigma^4} \\ - \quad &\frac{n}{\sigma^2} = 0 \end{aligned}$$

and so

$$\sigma_{i+1} = \sqrt{\frac{\sum_{j=1}^m (t_j - \mu_j^i)^2}{n} + \frac{\sigma_i^2 \sum_{j=m+1}^n \left[1 + \left(\frac{c_j - \mu_j^i}{\sigma_i}\right) H\left(\frac{c_j - \mu_j^i}{\sigma_i}\right)\right]}{n}} \tag{5.18}$$

where $\mu_j^i = \beta_0^i + \beta_1^i v_j$.

This might seem complicated, but all that is going on is:

1. Fitting of regression line by least squares;
2. Maximum likelihood estimate of error variance (allowing for uncertainty in censored observations);
3. Updating of estimates of censored data on basis of fitted model;
4. Iteration of these steps.

This is easily implemented in Splus with the following code:

```
em.reg_function(x, n)
{
        a <- lm(x[, 2] ~ x[, 1])
        b <- coef(a)
        sig <- sqrt(deviance(a)/38)
        cat(b, sig, fill = T)
        for(i in 1:n) {
                mu <- b[1] + b[2] * x[, 1]
                ez <- mu + sig * em.h((x[, 2] - mu)/sig)
                ez[1:17] <- x[1:17, 2]
                a <- lm(ez ~ x[, 1])
                b <- coef(a)
                ss <- sum((ez[1:17] - mu[1:17])^2)/40
                ss <- ss + (sig^2 * sum(((x[18:40, 2] - mu[18:40])/sig)
                     * em.h((x[18:40, 2] - mu[18:40])/sig) + 1))/40
                sig <- sqrt(ss)
                cat(b, sig, fill = T)
                abline(a, col = i)
        }
}
```

and the function

```
em.h_function(x)
{
        dnorm(x)/(1 - pnorm(x))
}
```

This produced the output:

```
-4.9305 3.7470 0.1572
-5.2601 3.9262 0.1998
-5.5112 4.0558 0.2173
-5.6784 4.1398 0.2288
-5.7854 4.1933 0.2371
-5.8552 4.2284 0.2431
-5.9023 4.2521 0.2474
-5.9350 4.2686 0.2506
-5.9581 4.2803 0.2529
-5.9747 4.2887 0.2545
-5.9867 4.2947 0.2558
```

and the sequence of linear fits as shown in Figure 5.2. Note that the initial fit was made as if the data were not censored, so the difference between the first and final fits gives an indication of the necessity of taking the censoring into account.

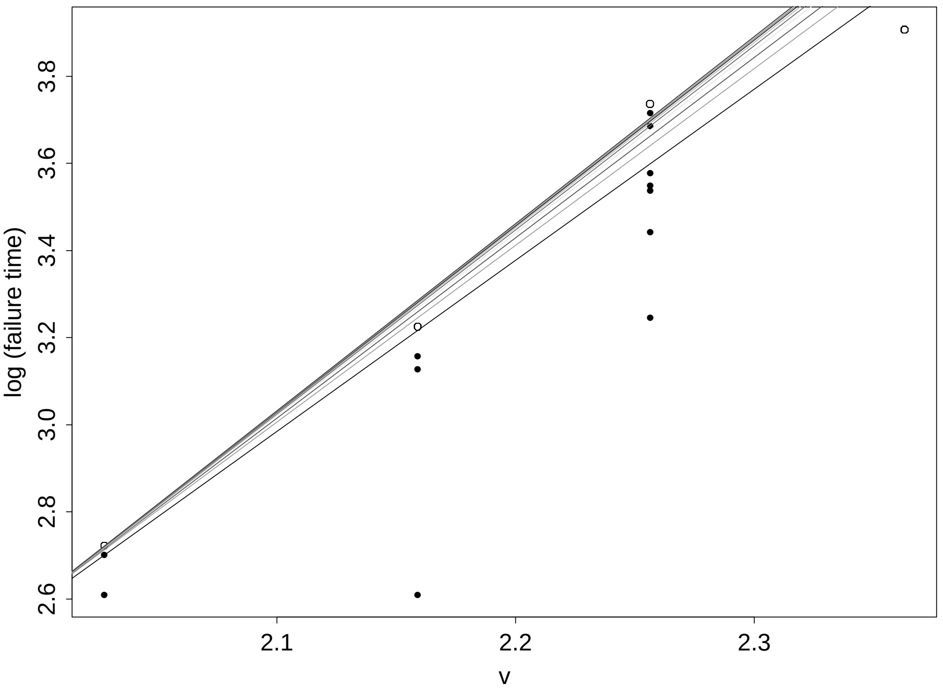

Figure 5.2: EM iterates to regression fit of censored data

5.5 Convergence

Every iteration of an EM algorithm results in a value of θ with higher likelihood. Furthermore, it can be shown that if these iterates converge, then they converge to a stationary point of the likelihood function, though this is not necessarily a global maximum. Furthermore, the rate of convergence can be very slow, suggesting that alternative procedures, or techniques which accelerate the convergence, may often be appropriate.

5.6 Standard errors

As usual, calculation of standard errors requires calculation of the inverse Hessian matrix:

$$\left[-\frac{d^2 \log p(\theta|Y)}{d\theta_i d\theta_j}\right]^{-1} \tag{5.19}$$

evaluated at the mode θ^*. In missing data problems this may be difficult to evaluate directly. It is possible however to exploit the structure of the EM algorithm in simplifying this calculation. This uses what is known as the 'missing information principle':

Observed information = Complete information — Missing Information

Explicitly, this takes the form:

$$-\frac{d^2 \log p(\theta|Y)}{d\theta_i d\theta_j} = \left[-\frac{d^2 Q(\theta,\phi)}{d\theta_i d\theta_j}\right]_{\phi=\theta} - \left[-\frac{d^2 H(\theta,\phi)}{d\theta_i d\theta_j}\right]_{\phi=\theta} \tag{5.20}$$

where

$$H(\theta,\phi) = \int_Z \log(p(Z|\theta,Y))p(Z|\phi,Y)dZ \tag{5.21}$$

Without the missing data, the first term on the right hand–side of (5.20) would be the Hessian; the second term compensates for the missing information. The proof of (5.20) is almost immediate from the representation

$$\log p(\theta|Y) = \log p(\theta|Y,Z) - \log p(Z|Y,\theta) + C \tag{5.22}$$

which is essentially a statement of Bayes' theorem. Application of the missing information principle is simplified by using the result:

$$-\frac{d^2 H(\theta,\phi)}{d\theta_i d\theta_j} = \text{Var}\left[\frac{d \log p(\theta|Y,Z)}{d\theta}\right] \tag{5.23}$$

a result which mirrors the corresponding result in the classical (full data) theory. We can illustrate this with the genetic linkage example. In that case

$$\begin{aligned}
-\left[\frac{d^2 Q(\theta,\phi)}{d\theta^2}\right]_{\theta^*} &= \frac{E(X_2|\theta^*,Y)+x_5}{\theta^{*2}} + \frac{x_3+x_4}{(1-\theta^*)^2} \\
&= \frac{29.83}{0.6268^2} + \frac{38}{(1-0.6268)^2} \\
&= 435.3
\end{aligned}$$

This is the information had the augmented data been genuine data. Now, we also have:

$$\frac{d \log p(\theta|Y,Z)}{d\theta} = \frac{x_2 + x_5}{\theta} - \frac{x_3 + x_4}{1-\theta} \tag{5.24}$$

so that

$$\begin{aligned}\mathrm{Var}\left[\frac{d \log p(\theta|Y,Z)}{d\theta}\right] &= \frac{\mathrm{Var}(X_2|\theta^*)}{\theta^{*2}} \\ &= 125\left(\frac{\theta^*}{2+\theta^*}\right)\left(\frac{2}{2+\theta^*}\right)\frac{1}{\theta^{*2}} \\ &= 22.71/0.6268^2 = 57.8\end{aligned}$$

Thus

$$-\frac{d^2 \log p(\theta|Y)}{d\theta^2} = 435.3 - 57.8 = 377.5 \tag{5.25}$$

and the standard error of θ^* is $\sqrt{(1/377.5)} = 0.05$.

5.7 Monte–Carlo EM algorithm

In some situations, direct evaluation of the Q function, or equivalently, estimation of the augmented data as expected values given the current model estimate and complete data, is difficult. In such situations we can employ a Monte–Carlo step to approximate Q. Thus, the E–Step of the EM algorithm is replaced with

1. Simulate $z_1, z_2, \ldots, z_m$ from $P(Z|Y, \theta^i)$
2. Let $\hat{Q}_{i+1} = \frac{1}{m}\sum_{j=1}^{m} \log L(\theta|z_j, Y)$

Thus, the exact expectation required for Q is replaced by a Monte–Carlo estimate of the integral.

The choice of m here will affect the accuracy of the final result. One approach therefore is to initiate the algorithm with a small value of m, but then to increase m later to minimize variability due to the error in the Monte–Carlo integration.

We can apply this again to the genetic linkage example. There we had

$$Q(\theta, \theta^i) = [E(X_2|\theta^i, Y) + x_5]\log(\theta) + (x_3 + x_4)\log(1-\theta) \tag{5.26}$$

where $x_2|\theta^i, Y \sim \mathrm{Bin}(125, \frac{\theta^i}{\theta^i+2})$. So, we simply generate $z_1, z_2, \ldots, z_m$ from $\mathrm{Bin}(125, \frac{\theta^i}{\theta^i+2})$, and take the mean of these realisations, $\bar{z}$, as the approximation to the expectation in (5.26). Then the M–Step continues as before with

$$\theta^{i+1} = \frac{\bar{z} + x_5}{\bar{z} + x_3 + x_4 + x_5} \tag{5.27}$$

This is implemented in Splus with the following code:

```
em.mc_function(x, m, th.init)
{
        th <- th.init
        for(i in 1:length(m)) {
                p <- th/(2 + th)
                z <- rbinom(m[i], 125, p)
                me <- mean(z)
                th <- (me + x[4])/(me + x[2] + x[3] + x[4])
                cat(i, round(th, 5), fill = T)
        }
}
```

where x is the data, m is a vector of the values of m we choose for each step in the iteration, and th.init is an initial value for θ. Choosing $m = 10$ for the first 8 iterations, and $m = 1000$ for the next 8, and with an initial value of $\theta = 0.5$ we obtained:

```
1 0.61224
2 0.62227
3 0.62745
4 0.62488
5 0.62927
6 0.62076
7 0.62854
8 0.62672
9 0.62734
10 0.627
11 0.62747
12 0.62706
13 0.62685
14 0.62799
15 0.62736
16 0.62703
```

Note that monitoring convergence is more difficult in this case due to the inherent variation in the Monte–Carlo integrals.