Monte Carlo Methods

Lecture notes for MAP001169

Based on Script by Martin Sköld

adopted by Krzysztof Podgórski

Contents

Part I

Simulation and Monte-Carlo Integration

Chapter 1

Simulation and Monte-Carlo integration

1.1 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.

1.2 Buffon's Needle

We'll start with a simulation experiment which has 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 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 point, 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

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

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, $R \sim \text{Bin}(n, p)$, so $\text{Var}(\hat{p}) = p(1-p)/n$. Thus $\text{Var}(\hat{\phi}_0) = 2\rho\phi(1-2\rho\phi)/4\rho^2 n = \phi^2(1/2\rho\phi - 1)/n$ 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 $\text{Var}(\hat{\phi}_0) = (\phi/2 - \phi^2)/n$.

Figure 1.1 gives 2 realisations of Buffon's experiment, based on 5000 simulations each. The figures together with an estimate can be produced in R by

```
buf=function(n,d,l){
x=runif(n)*d/2
theta=runif(n)*pi/2
I=(l*cos(theta)/2>x)
R=cumsum(I)
phat=R/(1:n)
nn=1:n
  plot(nn[phat>0],2*l/d/phat[phat>0],xlab='proportion of hits',ylab='pi estimate',type='l')
}
```

Exercise 1.1. *Provide with the full details in the argument above which showed that the optimality is achieved for the estimator $\hat{\phi}$.*

Use the R-code given above and design a Monte Carlo study that confirms (or not) that optimality is also achieved for $\hat{\pi}$ when $\rho = 1$, i.e. $d = l$. First, explain why it is not obvious. When discussing this review the concepts of the bias, the variance and the mean-square error and relations between these three. Then explain or/and analyze numerically what is the bias, the variance and the mean-square error of $\hat{\phi}$ and $\hat{\pi}$. Hint: Note that the event that the needle does not crosses the line in any trial has a positive probability

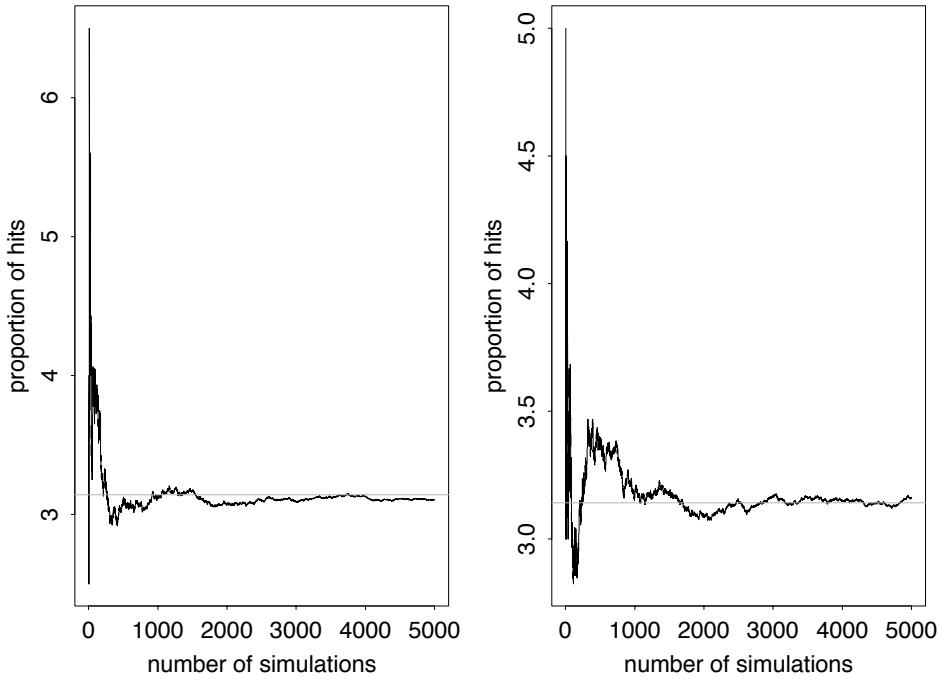

Figure 1.1: Two sequences of realisations of Buffon's experiment

and this affects existence of the mean and the variance of $\hat{\pi}$*. Modify the estimator to avoid the problem.*

The argument given above assumed that $l \leq d$*. Modify the algorithm to investigate also the case of* $d < l$*. Investigate the optimality in this case.*

Thus I've used the computer to simulate the physical simulations. You might like to check why this code works.

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.

1.3 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

$$x_i = ax_{i-1}(\text{mod } M) \tag{1.1}$$

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. Problems 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.

Chapter 2

Simulating from specified distributions

In this chapter we look at ways of simulating data from a specified distribution function F, on the basis of a simulated sample $u_1, u_2, \ldots, u_n$ from the distribution $U[0,1]$.

2.1 Transforming uniforms

We start with the case of constructing a draw x from a random variable $X \in \mathbf{R}$ with a continuous distribution F on the basis of a single u from $U[0,1]$. It is natural to try a simple transformation $x = h(u)$, but how should we choose h? Let's assume h is increasing with inverse $h^{-1} : \mathbf{R} \mapsto [0,1]$. The requirement is now that

$$\begin{aligned} F(v) &= P(X \leq v) = P(h(U) \leq v,) \\ &= P(h^{-1}(h(U)) \leq h^{-1}(v)) = P(U \leq h^{-1}(v)) \\ &= h^{-1}(v), \end{aligned}$$

for all $v \in \mathbf{R}$ and where in the last step we used that the distribution function of the $U[0,1]$ distribution equals $P(U \leq u) = u, u \in [0,1]$. The conclusion is clear, we should choose $h = F^{-1}$. If F is not one-to-one, as is the case for discrete random variables, the above argument remains valid if we define the inverse as

$$F^{-1}(u) = \inf\{x; F(x) \geq u\}. \tag{2.1}$$

The resulting algorithm for drawing from F is *the Inversion Method*:

Algorithm 2.1 (The Inversion Method)**.**

1. Draw u from $U[0,1]$.
2. $x = F^{-1}(u)$ can now be regarded a draw from F.

Figure 2.1 illustrates how this works. For example, to simulate from the

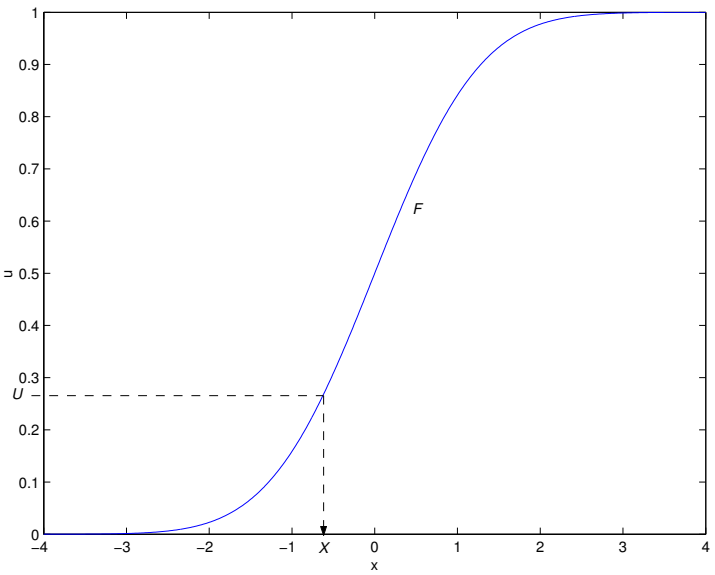

Figure 2.1: Simulation by inversion; the random variable $X = F^{-1}(U)$ will have distribution F if U is uniformly distributed on $[0, 1]$.

exponential distribution we have $F(x) = 1 - \exp(-\lambda x)$, so

$$F^{-1}(u) = -\lambda^{-1} \log(1 - u).$$

Thus with

```
u=runif(1,n);
x=-(log(1-u))/lambda;
```

we can simulate `n` independent values from the exponential distribution with parameter `lambda`. Figure 2.2 shows a histogram of 1000 standard ($\lambda = 1$) exponential variates simulated with this routine.

For discrete distributions, 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
```

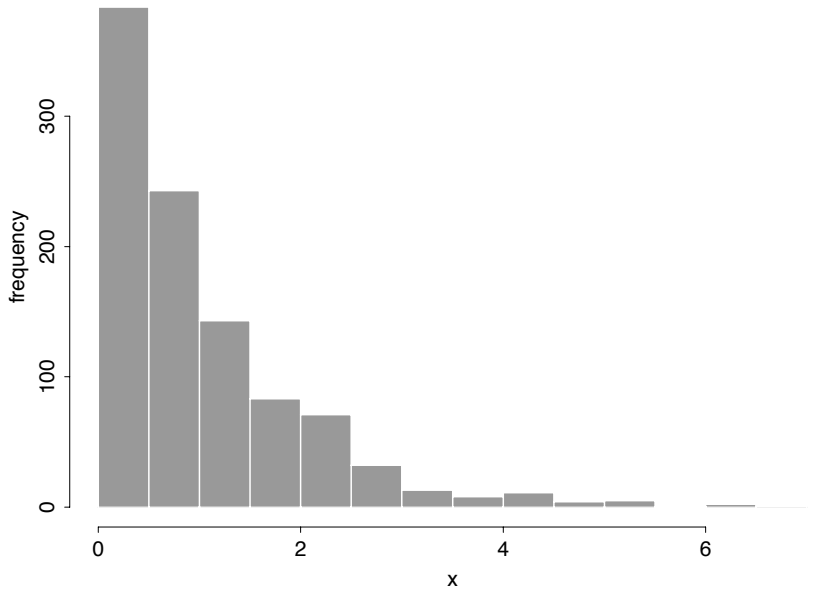

Figure 2.2: Histogram of 1000 simulated unit exponential variates

```
 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_i where $F(x_i - 1) < u_i \leq F(x_i)$. 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. An even more serious limitation is that the method only applies for generating draws from univariate random variables. To deal with such cases, we turn to a variety of alternative schemes.

2.2 Transformation methods

The inversion method is a special case of more general transformation methods. The following theorem can be used to derive the distribution of $Z = h(X)$ for a more general class of real-valued random variables X.

Theorem 2.1 (Transformation theorem)**.** *Let $X \in \mathcal{X} \subseteq \mathbf{R}$ have a continuous density f and h a function with differentiable inverse $g = h^{-1}$. Then the random variable $Z = h(X) \in \mathbf{R}$ has density*

$$f(g(z))|g'(z)|, \tag{2.2}$$

for $z \in h(\mathcal{X})$ and zero elsewhere.

Proof. The proof is a direct application of the change-of-variable theorem for integrals. Note that two random variables X and Z have the same distribution iff $P(X \in A) = P(Z \in A)$ for all sets A. □

This device is used extensively in simulation, for example when we want to generate a $N(\mu, \sigma^2)$ variate y, it is common to first draw x from $N(0, 1)$ and then set $y = \sigma x + \mu$. Use Theorem 2.1 to show that this works. Sums of random variables can also be useful in creating new variables. Recall that

Theorem 2.2. *Let $X \in \mathbf{R}$ and $Y \in \mathbf{R}$ be independent with densities f and g respectively, then the density of $Z = X + Y$ equals $f * g(z) = \int f(t - z)g(t)\, dt$.*

This can be used to generate Gamma random variables. A random variable X has a Gamma$(a, 1)$ distribution if its density is proportional to $x^{a-1} \exp(-x)$, $x > 0$. Using Theorem 2.2 we can show that if X and Y are independent Gamma$(a, 1)$ and Gamma$(a', 1)$ respectively, then $Z = X + Y$ has a Gamma$(a + a', 1)$ distribution. Since Gamma$(1, 1)$ (i.e. Exponential(1)) variables are easily generated by inversion, a Gamma$(k, 1)$ variable Z, for integer values k, is generated by

$$z = \sum_{i=1}^{k} -\log(u_k) \tag{2.3}$$

using independent draws of uniforms $u_1, \ldots, u_n$. As an alternative we can use a combination of Theorems 2.1 and 2.2 to show that

$$z = \sum_{i=1}^{2k} x_i^2/2 \tag{2.4}$$

is a draw from the same distribution if $x_1, \ldots, x_{2k}$ are independent standard Normal draws.

Example 2.1 (The Box-Muller transformation)**.** This is a special trick to simulate from the Normal distribution. In fact it produces two independent variates in one go. Let u_1, u_2 be two independently sampled U$[0, 1]$ variables, then it can be shown that

$$x_1 = \sqrt{-2\log(u_2)}\cos(2\pi u_1) \text{ and } x_2 = \sqrt{-2\log(u_2)}\sin(2\pi u_1)$$

are two independent N$(0, 1)$ variables.

Below we give the multivariate version of Theorem 2.1.

Theorem 2.3 (Multivariate transformation theorem)**.** *Let $X \in \mathcal{X} \subseteq \mathbf{R}^d$ have a continuous density f and $h : \mathcal{X} \mapsto \mathbf{R}^d$ a function with differentiable inverse $g = h^{-1}$. Further write $J(z)$ for the determinant of the Jacobian matrix of $g = (g_1, \ldots, g_d)$,*

$$J(x) = \begin{vmatrix} dg_1(z)/dz_1 & \ldots & dg_1(z)/dz_d \\ \vdots & \ddots & \vdots \\ dg_d(z)/dz_1 & \ldots & dg_d(z)/dz_d \end{vmatrix}. \tag{2.5}$$

Then the random variable $Z = h(X) \in \mathbf{R}^d$ has density

$$f(g(z))|J(z)|, \tag{2.6}$$

for $z \in h(\mathcal{X})$ and zero elsewhere.

Example 2.2 (Choleski method for multivariate Normals)**.** The Choleski method is a convenient way to draw a vector z from the multivariate Normal distribution $N_n(0, \Sigma)$ based on a vector of n independent $N(0, 1)$ draws $(x_1, x_2, \ldots, x_n)$. Choleski decomposition is a method for computing a matrix C such that $CC^T = \Sigma$, in R the command is `chol`. We will show that $z = Cx$ has the desired distribution. The density of X is $f(x) = (2\pi)^{-d/2} \exp(-x^T x/2)$ and the Jacobian of the inverse transformation, $x = C^{-1}z$, equals $J(z) = |C^{-1}| = |C|^{-1} = |\Sigma|^{-1/2}$. Hence, according to Theorem 2.3, the density of Z equals

$$\begin{aligned} f(C^{-1}z)|\Sigma|^{-1/2} &= (2\pi)^{-d/2} \exp(-(C^{-1}z)^T (C^{-1}z)/2)|\Sigma|^{-1/2} \\ &= (2\pi)^{-d/2} \exp(-z^T \Sigma^{-1} z/2)|\Sigma|^{-1/2}, \end{aligned}$$

which we recognise as the density of a $N_n(0, \Sigma)$ distribution. Of course, $z + \mu$, $\mu \in \mathbf{R}^d$ is a draw from $N_n(\mu, \Sigma)$.

2.3 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 method is based on the following theorem:

Theorem 2.4. *Let f be the density function of a random variable on $\mathbf{R}^d$ and let $Z \in \mathbf{R}^{d+1}$ be a random variable that is uniformly distributed on the set $A = \{z; 0 \leq z_{d+1} \leq Mf(z_1, \ldots, z_d)\}$ for an arbitrary constant $M > 0$. Then the vector $(Z_1, \ldots, Z_d)$ has density f.*

Proof. First note that

$$\begin{aligned}\int_A dz &= \int_{\mathbf{R}^d} \left(\int_0^{Mf(z_1,\ldots,z_d)} dz_{d+1} \right) dz_1 \cdots dz_d \\ &= M \int f(z_1, \ldots, z_d)\, dz_1 \cdots dz_d = M.\end{aligned}$$

Hence, Z has density $1/M$ on A. Similarily, with $B \subseteq \mathbf{R}^d$, we have

$$\begin{aligned}P((Z_1, \ldots, Z_d) \in B) &= \int_{\{z; z\in A\}\cap\{z;(z_1,\ldots,z_d)\in B\}} M^{-1}\, dz \\ &= M^{-1} \int_B Mf(z_1, \ldots, z_d)\, dz_1 \cdots dz_d \\ &= \int_B f(z_1, \ldots, z_d)\, dz_1 \cdots dz_d,\end{aligned}$$

and this is exactly what we needed to show. □

The conclusion of the above theorem is that we can construct a draw from f by drawing uniformly on an appropriate set and then drop the last coordinate of the drawn vector. Note that the converse of the above theorem is also true, i.e. if we draw $(z_1, \ldots, z_d)$ from f and then z_{d+1} from $U(0, Mf(z_1, \ldots, z_d))$, $(z_1, \ldots, z_{d+1})$ will be a draw from the uniform distribution on $A = \{z; 0 \leq z_{d+1} \leq Mf(z_1, \ldots, z_d)\}$. The question is how to draw uniformly on A without having to draw from f (since this was our problem in the first place); the rejection method solves this by drawing uniformly on a larger set $B \supset A$ and rejecting the draws that end up in $B\backslash A$. A natural choice of B is $B = \{z; 0 \leq z_{d+1} \leq Kg(z_1, \ldots, z_d)\}$, where g is another density, the *proposal density*, that is easy to draw from and satisfies $Mf \leq Kg$.

Algorithm 2.2 (The Rejection Method)**.**

1. Draw $(z_1, \ldots, z_d)$ from a density g that satisfies $Mf \leq Kg$.
2. Draw z_{d+1} from $U(0, Kg(z_1, \ldots, z_d))$.
3. Repeat steps 1-2 until $z_{d+1} \leq Mf(z_1, \ldots, z_d)$.
4. $x = (z_1, \ldots, z_d)$ can now be regarded as a draw from f.

It might seem superflous to have two constants M and G in the algorithm. Indeed, the rejection method is usually presented with $M = 1$. We include M here to illustrate the fact that you only need to know the density up to a constant of proportionality (i.e. you know Mf but not M or f). This situation is very common, especially in applications to Bayesian statistics.

The efficiency of the rejection method depends on how many points are rejected, which in turn depends on how close Kg is to Mf. The probability of accepting a particular draw $(z_1, \ldots, z_d)$ from g equals

$$\begin{aligned} &P(Z_{d+1} \leq Mf(Z_1, \ldots, Z_d)) \\ &\quad = \int \left(\int_0^{Mf(z_1,\ldots,z_d)} (Kg(z_1, \ldots, z_d))^{-1} \, dz_{d+1} \right) g(z_1, \ldots, z_d \Big) \, dz_1 \cdots dz_d \\ &\quad = \frac{M}{K} \int f(z_1, \ldots, z_d) \, dz_1 \cdots dz_d = \frac{M}{K}. \end{aligned}$$

For large d it becomes increasingly difficult to find g and K such that M/K is large enough for the algorithm to be useful. Hence, while the rejection method is not strictly univariate as the inversion method, it tends to be practically useful only for small d.

The technique is illustrated in Figure 2.3.

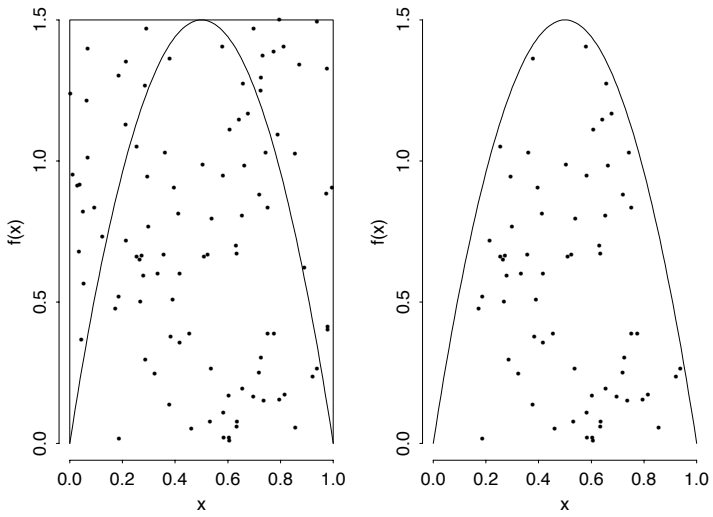

Figure 2.3: Simulation by rejection sampling from an $U[0, 1]$ distribution (here $M = 1$ and $G = 1.5$); the x–coordinates of the points in the right panel constitute a sample with density f

As an example, consider the distribution with density

$$f(x) \propto x^2 e^{-x}; \quad 0 \leq x \leq 1, \tag{2.7}$$

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 2.4 shows both $f(x)$ and $g(x)$. Clearly in this case the envelope is very poor so the routine is highly

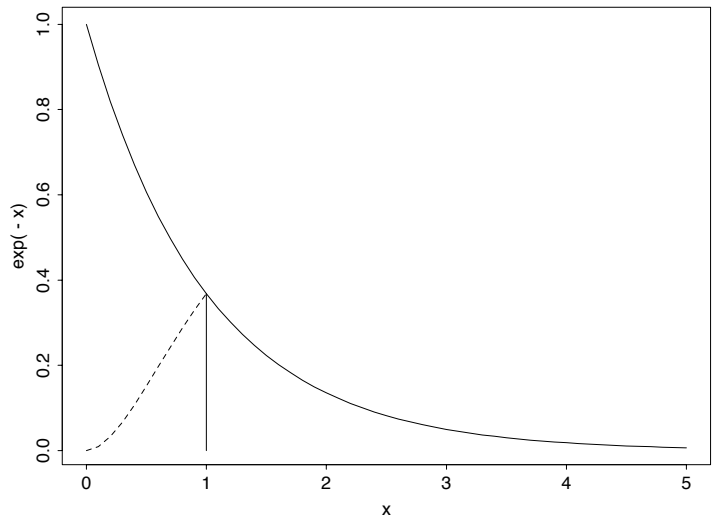

Figure 2.4: Scaled density and envelope

inefficient (though statistically correct). Applying this to generate a sample of 100 data by

```
RS=rejsim(100)
hist(RS$sample)
RS$count
```

using the following code

```
rejsim=function(n){
   x=vector("numeric",n)
   m=0
   for(i in 1:n)
   {
     acc=0
     while(acc<1){
       m=m+1
       z1=-log(runif(1))
       z2=runif(1)*exp(-z1)
       if(z2<z1^2*exp(-z1)*(z1<1)){
       acc=1
       x[i]=z1
       }
     }
   }
   rejsim=list(sample=x,count=m)
}
```

gave the histogram in Figure 2.5. The variable `m` contains the number of random variate pairs (Z_1, Z_2) needed to accept 100 variables from the correct distribution, in our simulation `m=618` suggesting that the algorithm is rather poor. What values of M and G did we use in this example?

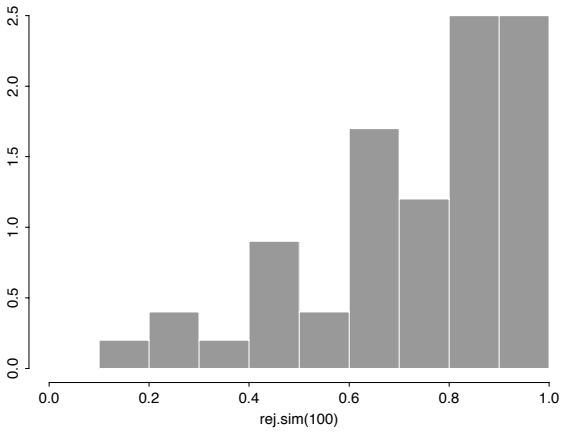

Figure 2.5: Histogram of simulated data

Exercise 2.1. *Suggest and implement a more efficient method of rejection sampling for the above truncated distribution. Compare numerical efficiency of both the methods through a Monte Carlo study.*

2.4 Conditional methods

The inversion method is strictly univariate, since the inverse F^{-1} is not well-defined for functions $F : \mathbf{R}^d \mapsto [0, 1]$ when $d > 1$. The rejection method is not limited to $d = 1$, but for large d it becomes increasingly difficult to find a bounding function $Kg(x)$ that preserves a reasonably high acceptance rate. A general technique to simulate from a multivariate distribution, using steps of univariate draws, is suggested by a factorization argument. Any d-variate density function f can be factorised recursively as

$$f(x_1, \ldots, x_d) = f_1(x_1) f_2(x_2|x_1) f_3(x_3|x_2, x_1) \cdots f_d(x_d|x_{d-1}, \ldots, x_1). \quad (2.8)$$

Given the above factorisation, a draw from f can now be produced recursively by

Algorithm 2.3.

1. Draw x_1 from the distribution with density $f_1(\cdot)$.
2. Draw x_2 from the distribution with density $f_2(\cdot|x_1)$.
3. Draw x_3 from the distribution with density $f_3(\cdot|x_1, x_2)$.

$\vdots$

d. Draw x_d from the distribution with density $f_d(\cdot|x_1, x_2, \ldots, x_{d-1})$.

$(x_1, \ldots, x_d)$, is now a draw from $f(x_1, \ldots, x_d)$ in (2.8).

In the above algorithm, each step could be performed with an univariate method. The problem is that, commonly, the factorisation in (2.8) is not explicitly available. For example, deriving f_1 involves the integration

$$f_1(x_1) = \int f(x_1, \ldots, x_d)\, dx_2 \cdots dx_d,$$

which we might not be able to perform analytically.

Example 2.3 (Simulating a Markov Chain)**.** Recall that a Markov Chain is a stochastic process $(X_0, X_1, \ldots, X_n)$ such that, conditionally on $X_{i-1} = x_{i-1}$, X_i is independent of the past $(X_0, \ldots, X_{i-2})$. Assuming x_0 is fixed, the factorisation (2.8) simplifies to

$$f(x_1, \ldots, x_n) = f_1(x_1|x_0) f(x_2|x_1) \cdots f(x_n|x_{n-1}),$$

for a common transition density $f(x_i|x_{i-1})$ of $X_i|X_{i-1} = x_{i-1}$. Thus, to simulate a chain starting from x_0, we proceed recursively as follows

1. Draw x_1 from the distribution with density $f(\cdot|x_0)$.
2. Draw x_2 from the distribution with density $f(\cdot|x_1)$.

$\vdots$

n. Draw x_n from the distribution with density $f(\cdot|x_{n-1})$.

Example 2.4 (Bivariate Normals)**.** Another application of the factorisation argument is useful when generating draws from the bivariate Normal distribution. If

$$(X_1, X_2) \sim N_2\left((\mu_1, \mu_2), \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}\right),$$

we obviously have that $X_1 \sim N(\mu_1, \sigma_1^2)$ and it is a straightforward exercise to show that $X_2|X_1 = x_1 \sim N(\mu_2 + \rho\sigma_2(x_1 - \mu_1)/\sigma_1, \sigma_2^2(1 - \rho^2))$.

Exercise 2.2. *Propose and implement the conditional method of simulation for a normal vector $(X_1, \ldots, X_2) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Perform numerical comparison of the efficiency of your method with the one based on Cholesky's decompostion.*

Chapter 3

Monte-Carlo integration

3.1 Generic Monte Carlo integration

Monte-Carlo integration is a numerical method for integration based on the *Law of Large Numbers* (LLN). The algorithm goes as follows:

Algorithm 3.1 (Basic Monte-Carlo Integration)**.**

1. Draw N values $x_1, \ldots, x_N$ independently from f.
2. Approximate $\tau = E(\phi(X))$ by

$$t_N = t(x_1, \ldots, x_N) = \frac{1}{N} \sum_{i=1}^{N} \phi(x_i).$$

As an example of this, suppose we wish to calculate $P(X < 1, Y < 1)$ where (X, Y) are bivariate normal distribution with correlation 0.5 and having standard normal distribution for marginals. This can be written as

$$\int \mathbf{1}\{x < 1, y < 1\} f(x, y)\, dx\, dy \tag{3.1}$$

where f is the bivariate normal density. Thus, provided we can simulate from the bivariate normal, we can estimate this probability as

$$n^{-1} \sum_{i=1}^{n} \mathbf{1}\{x_i < 1, y_i < 1\} \tag{3.2}$$

which is simply the proportion of simulated points falling in the set defined by $\{(x, y); x < 1, y < 1\}$. Here we use the approach from Example 2.4 for simulating bivariate normals. R code to achieve this is

```
 bvnsim=function(n,m,s,r){
  x=rnorm(n)*s[1]+m[1]
  y=rnorm(n)*s[2]*sqrt(1-r^2)+m[2]+(r*s[2])/s[1]*(x-m[1])
  bvnsim=matrix(0,ncol=2,nrow=n)
  bvnsim[,1]=x
  bvnsim[,2]=y
  bvnsim
}
```

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

```
X=bvnsim(1000,c(0,0),c(1,1),.5);
mean((X[,1]<1)&(X[,2]<1))
```

I got the estimate 0.763 doing this. A scatterplot of the simulated values is given in Figure 3.1.

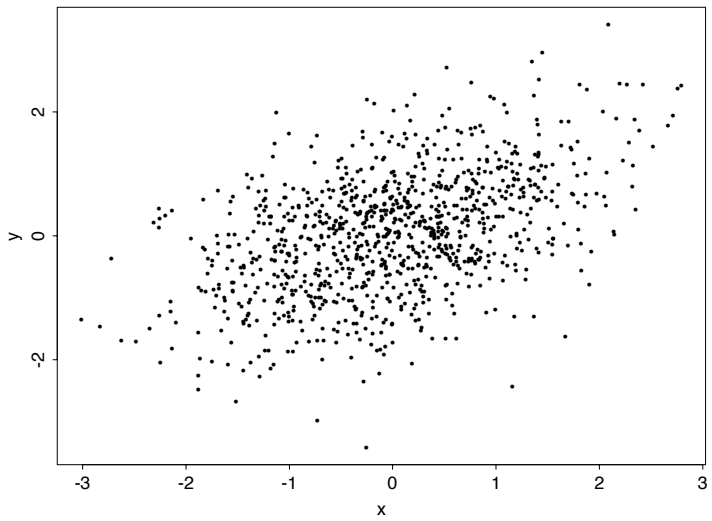

Figure 3.1: Simulated bivariate normals

Example 3.1. For a non-statistical example, say we want to estimate the integral

$$\begin{aligned}\tau &= \int_0^{2\pi} x \sin[1/\cos(\log(x+1))]^2\, dx \\ &= \int (2\pi x \sin[1/\cos(\log(x+1))]^2)(\mathbf{1}\{0 \le x \le 2\pi\}/(2\pi))\, dx,\end{aligned}$$

where, of course, the second term of the integrand is the $U[0, 2\pi]$ density function. The integrand is plotted in Figure 3.2, and looks to be a challenge for many numerical methods.

Monte-Carlo integration in R proceeds as follows:

```
x=runif(10000)*2*pi
tn=mean(2*pi*x*sin(1/cos(log(x+1)))^2)
tn
[1] 8.820808
```

Maple, using `evalf` on the integral, gave 8.776170832. A larger run of the Monte-Carlo algorithm shows that this might be an overestimate and that the true value is close to 8.756.

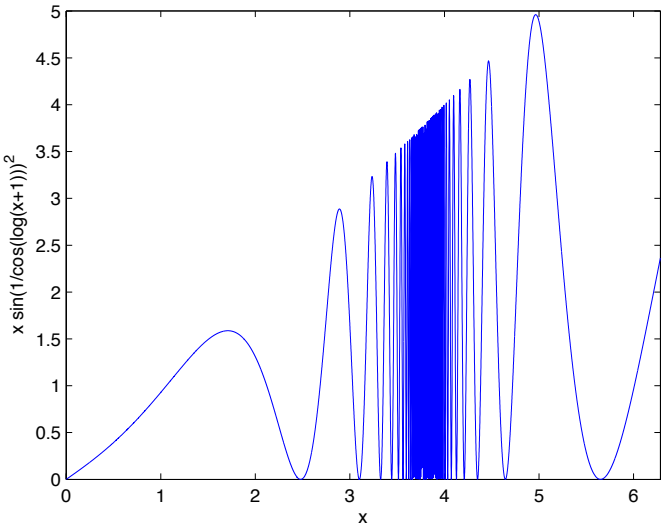

Figure 3.2: An attempt at plotting $x \sin(1/\cos(\log(x+1)))^2$.

We suggested the motivation comes from the LLN. There are many versions of this celebrated theorem, we will provide a simple mean-square version. First note that if $X_1, \ldots, X_n$ is a sequence of random variables and $T_n = t(X_1, \ldots, X_n)$ for a function t, we say that T_n *converges in the mean square sense* to a fixed value τ if

$$E(T_n - \tau)^2 \to 0 \text{ as } n \to \infty.$$

Theorem 3.1 (A Law of Large Numbers)**.** *Assume $Z_1, \ldots, Z_n$ is a sequence of independent random variables with common means $E(Z_i) = \tau$ and variances $Var(Z_i) = \sigma^2$. If $T_n = n^{-1} \sum_{i=1}^{n} Z_i$, we have*

$$E(T_n - \tau)^2 = \frac{\sigma^2}{n} \to 0 \text{ as } n \to \infty. \tag{3.3}$$

Proof. Simple and straightforward; exercise. □

The above theorem tells us that with $Z_i = \phi(X_i)$ where X_i are independent with density f, the arithmetic mean of $Z_1, \ldots, Z_n$ converges in mean square error to $\tau = E(g(X))$. Moreover, it gives the precise rate of the error: $(E(T_n - \tau)^2)^{1/2} = O(n^{-1/2})$ and this rate is *independent of dimension d*. This is in contrast to deterministic methods for numerical integration, like the trapezoidal rule and Simpson's rule, that have errors of $O(n^{-2/d})$ and $O(n^{-4/d})$ respectively. Monte-Carlo integration is to be preferred in high dimensions (greater than 4 and 8 respectively). Another advantage is that we can reuse the drawn values $x_1, \ldots, x_N$ to estimate other expectations with respect to f without much extra effort.

More precise information on the Monte-Carlo error $(T_n - \tau)$ is given by celebrated result no. 2: the *Central Limit Theorem* (CLT).

Theorem 3.2 (Central Limit Theorem)**.** *Assume $Z_1, \ldots, Z_n$ is a sequence of i.i.d. random variables with common means $E(Z_i) = \tau$ and variances $Var(Z_i) = \sigma^2$. If $T_n = n^{-1} \sum_{i=1}^n Z_i$, we have*

$$P\Big(\frac{\sqrt{n}(T_n - \tau)}{\sigma} \leq x\Big) \to \Phi(x) \text{ as } n \to \infty, \tag{3.4}$$

where Φ is the distribution function of the $N(0,1)$ distribution.

Proof. Almost as simple, but somewhat less straightforward than LLN. Look it up in a book. □

Slightly less formally, the CLT tells us that the difference $T_n - \tau$ has, at least for large n, approximately an $N(0, \sigma^2/n)$ distribution. With this information we can approximate probabilities like $P(|T_n - \tau| > \epsilon)$, and perhaps more importantly find ϵ such that $P(|T_n - \tau| > \epsilon) = 1 - \alpha$ for some specified confidence level α. To cut this discussion short, the random interval

$$[T_n - 1.96\hat{\sigma}/\sqrt{n}, T_n + 1.96\hat{\sigma}/\sqrt{n}] \tag{3.5}$$

will cover the true value τ with approximately 95% probability. Here $\hat{\sigma}$ is your favourite estimate of standard deviation, e.g. based on

$$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^n (z_i - \bar{z})^2, \tag{3.6}$$

and 1.96 is roughly $\Phi^{-1}(0.95)$, the standard Normal 95% quantile.

A similar result to the central limit theorem also holds for the median and general sample quantiles:

Theorem 3.3. *Assume $Z_1, \ldots, Z_n$ is a sequence of i.i.d. random variables with distribution function $F(z - \tau)$ such that $F(0) = \alpha$ and that at zero F has density $f(0) > 0$. Then*

$$P(\sqrt{C_\alpha n}(Z_{(\lceil n\alpha \rceil)} - \tau) \leq x) \to \Phi(x) \text{ as } n \to \infty, \tag{3.7}$$

where $C_\alpha = \alpha(1-\alpha)f^2(0)$ and Φ is the distribution function of the $N(0,1)$ distribution.

Exercise 3.1. *Let (X_1, X_2, X_3) have the trivariate exponential distribution with density proportional to*

$$\exp(-x_1 - 2x_2 - 3x_3 - \max(x_1, x_2, x_3)), \quad x_i > 0, \quad i = 1, \ldots, 3.$$

Construct an algorithm that draws from (X_1, X_2, X_3) using the rejection method, proposing a suitable vector of independent exponentials.

Use basic Monte-Carlo integration to produce an approximate 95% accuracy interval for the probability $P(X_1^2 + X_2^2 \leq 2)$.

Exercise 3.2. *Let $\pi(k)$ be the number of primes less than k. How can you approximate $\pi(10^9)$ without having to check all integers less than 10^9? You* could *use the famous* prime-number theorem*, which says that $\pi(k) \approx k/\log(k)$ for large k. See the following Wikipedia link for more details on historical and mathematical aspects of this result: Prime Number Theorem. We will not this "deterministic result". Instead, let X be uniformly distributed on the odd numbers $\{1, 3, \ldots, 10^9 - 1\}$ (but remember that 2 is also a prime). Let ψ be an indicator of a prime number, i.e. it is a function that takes value one if its argument is prime and zero otherwise.*

Find the (simple) relation between the expected value $E(\psi(X))$ and $\pi(10^9)$. Then use Monte-Carlo method to approximate $\pi(10^9)$ by sampling $X_1, \ldots, X_n$ from X averaging $\psi(X_i)$, $i = 1, \ldots, n$. By what a result in probability theory averaging approximates the expected value of $E(\psi(X))$. You might find R package 'primes' with its `is_prime` *function useful here. Provide with the error assessment. Compare your result with the prime-number theorem.*

3.2 Bias and the Delta method

It is not always the case that we can find a random variable T_n such that $E(T_n) = \tau$. For example we might be interested in $\tau = h(E(X))$ for some specified smooth function h. If $\bar{X}$ again is the arithmetic mean, then a natural choice is $T_n = h(\bar{X})$. However, unless h is linear, $E(T_n)$ is not guaranteed to equal τ. This calls for a definition: the *bias* of t (when viewed as an estimator of τ), $T_n = t(X_1, \ldots, X_n)$ is

$$\text{Bias}(t) = E(T_n) - \tau. \tag{3.8}$$

The concept of bias allows us to more fully appreciate the concept of mean square error, since

$$E(T_n - \tau)^2 = \text{Var}(T_n) + \text{Bias}^2(t), \tag{3.9}$$

(show this as an exercise). The mean square error equals variance plus squared bias. In the above mentioned example, a Taylor expansion gives an impression of the size of the bias. Roughly we have with $\mu = E(X)$

$$\begin{aligned} E(T_n - \tau) &= E[h(\bar{X}) - h(\mu)] \\ &\approx E(\bar{X} - \mu)h'(\mu) + \frac{E(\bar{X} - \mu)^2}{2}h''(\mu) \\ &= \frac{\text{Var}(X)}{2n}h''(\mu). \end{aligned} \tag{3.10}$$

And it is reassuring that (3.10) suggests a small bias when sample size n is large. Moreover, since variance of T_n generally is of order $O(n^{-1})$ it will dominate the $O(n^{-2})$ squared bias in (3.9) suggesting that bias is a small problem here (though it can be a serious problem if the above Taylor expansions are not valid).

We now turn to the variance of T_n. First note that while $\mathrm{Var}(\bar{X})$ is easily estimated by e.g. (3.6), estimating $\mathrm{Var}(h(\bar{X}))$ is not so straightforward. An useful result along this line is the *Delta Method*

Theorem 3.4 (The Delta method)**.** *Let r_n be an increasing sequence and S_n a sequence of random variables. If there is μ such that h is differentiable at μ and*

$$P(r_n(S_n - \mu) \leq x) \to F(x), \text{ as } n \to \infty$$

for a distribution function F, then

$$P(r_n(h(S_n) - h(\mu)) \leq x) \to F(x/|h'(\mu)|).$$

Proof. Similar to the Taylor expansion argument in (3.10). □

This theorem suggests that if $S_n = \bar{X}$ has variance σ^2/r_n, then the variance of $T_n = h(S_n)$ will be approximately $\sigma^2 h'(\mu)^2/r_n$ for large n. Moreover, if S_n is asymptotically normal, so is T_n.

Exercise 3.3. *Implement Monte Carlo evaluation of integral*

$$I = \int_0^{2\pi} x^{2|sinx|} e^{x\cos^{3/2} x} dx.$$

Analyze the error of your evaluation. Suppose that one is interested in the accuracy of I^{-2} from the obtained approximation of I. Apply the delta method to assess this accuracy.

3.3 Variance reduction by rejection sampling

The method based on uniform sampling is simple but also not very inteligent. After all uniform distribution contains no information about the function at integral of which we aim. Through uniform samples we sometimes sample over regions were values of the function that do contribute much to the value of the integral but equally often (uniformly) we sample over regions where this is not true. Due to this we have large variability in the approximations – large variance. One can try to reduce this variability by being smarter, i.e. by utilizing some information about the function. In fact one method of achieving it can utilize rejection sampling algorithm that was discussed as a method of sampling from a distribution. There we were approximating a shape of the density (up to the normalizing constant) by the shape of a proposal density from which we could sample. The fact that the method worked without necessity of knowing the normalizing constants can be utilized here.

Consider a known density $f(x)$ on (a,b) from which one can simulate samples. Let us assume that $\phi(x) > 0$ is a function on (a,b) from which an integral is supposed to be approximated. Let a constant K be such that $\phi(x) \leq Kf(x)$. Consider 0-1 random variables X_i indicating if the ith attempt in rejection sampling is rejected or accepted. Then

$$P(X_i = 1) = \frac{\int_a^b \phi(x)\ dx}{K}.$$

Since X_i are iid thus by the LLN

$$\hat{I}_n = K\bar{X} \approx \int_a^b \phi(x)\ dx.$$

One can easily see that the variance of the method is given by

$$Var(\hat{I}_n) = K^2 \frac{\int_a^b \phi(x)\ dx}{K}\left(1 - \frac{\int_a^b \phi(x)\ dx}{K}\right)/n = \int_a^b \phi(x)\,dx \left(K - \int_a^b \phi(x)\ dx\right)/n.$$

Thus if K is close to $\int_a^b \phi(x)\ dx$ the variance can be small.

Exercise 3.4. *Consider a distribution on* $[0,\pi]$ *given by the cdf*

$$F(u) = \frac{1 - e^{-u^2}/2}{1 - e^{\pi^2/2}}.$$

The simulation from this distribution is easily achieved by inverting the cdf.

One can use the discussed method to evaluate the integral of

$$\phi(x) = x\sqrt{\sin x}e^{-x^2}, \quad x \in [0,\pi].$$

Perform the analysis comparing variance of the method with the variance of the method based on uniform sampling over the interval $[0,\pi]$.

The idea of variance reduction that is evident in our rejection algorithm is further explored by a similar but slightly more advanced and more popular method that is discussed next.

3.4 Variance reduction by importance sampling

Importance sampling is a technique that might substantially decrease the variance of the Monte-Carlo error. It can also be used as a tool for estimating $E(\phi(X))$ in cases where X can not be sampled easily.

We want to calculate

$$\tau = \int \phi(x)f(x)dx \tag{3.11}$$

which can be re–written

$$\tau = \int \psi(x)g(x)dx \tag{3.12}$$

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

$$t_n = n^{-1}\sum_{i=1}^{n}\psi(x_i), \tag{3.13}$$

for which the variance is

$$\mathrm{Var}(T_n) = n^{-1}\int\{\psi(x) - \tau\}^2 g(x)dx. \tag{3.14}$$

This variance can be very low, much lower than the variance of an estimate based on draws from f, provided 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. Another important advantage of importance sampling comes in problems where drawing from f is difficult. Here draws from f can be replaced by draws from an almost arbitrary density g (though it is essential that $\phi f/g$ remain bounded).

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

$$f(x) = \frac{1}{\pi(1 + x^2)} \tag{3.15}$$

so we require the integral

$$\int \mathbf{1}\{x > 2\}f(x)dx. \tag{3.16}$$

We could simulate from the Cauchy directly and apply basic Monte-Carlo integration, 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. Put differently, we are spending most of our simulation budget on estimating the integral of $\mathbf{1}\{x > 2\}f(x)$ over an area (i.e. around the origin) where we know it equals zero.

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:

$$t_n = n^{-1}\sum_{i=1}^{n}\frac{x_i^2}{2\pi(1 + x_i^2)}, \tag{3.17}$$

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

```
impsamp=function(n){
  x=2/runif(n)
  psi=x^2/(2*pi*(1+x^2))
  tn=mean(psi)
  cum=cumsum(psi)/seq(1,n,by=1)
  impsamp=list(tn=tn,cum=cum)
}
```

and processing

```
 is=impsamp(1000);
 plot(is$cum, type='l')\
```

Matlab function

```
function [tn,cum]=impsamp(n);
x=2./rand(1,n);
psi=x.^2./(2*pi*(1+x.^2));
tn=mean(psi);
cum=cumsum(psi)./(1:n);
```

and processing

```
 [tn,cum]=impsamp(1000);
 plot(cum)
```

gave the estimate $t_n = .1478$. The exact value is $.5 - \pi^{-1} \tan 2 = .1476$. In Figure 3.3 the convergence of this sample mean to the true value is demonstrated as a function of n by plotting the additional output vector `cum`.

For comparison, in Figure 3.4, 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 (the importance sampled estimator looks like a straight line).

3.4.1 Unknown constant of proportionality

To be able to use the above importance sampling techniques, we need to know $f(x)$ explicitly. Just knowing Mf for an unknown constant of proportionality M is not sufficient. However, importance sampling can also be used to approximate M. Note that,

$$M = \int Mf(x)\,dx = \int \frac{Mf(x)}{g_2(x)} g_2(x)\,dx, \tag{3.18}$$

for a density g_2. Thus, based on a sample $x_1, x_2, \ldots, x_N$ from g_2, we can approximate M by

$$t_N = \frac{1}{N} \sum_{i=1}^{N} \frac{Mf(x_i)}{g_2(x_i)}. \tag{3.19}$$

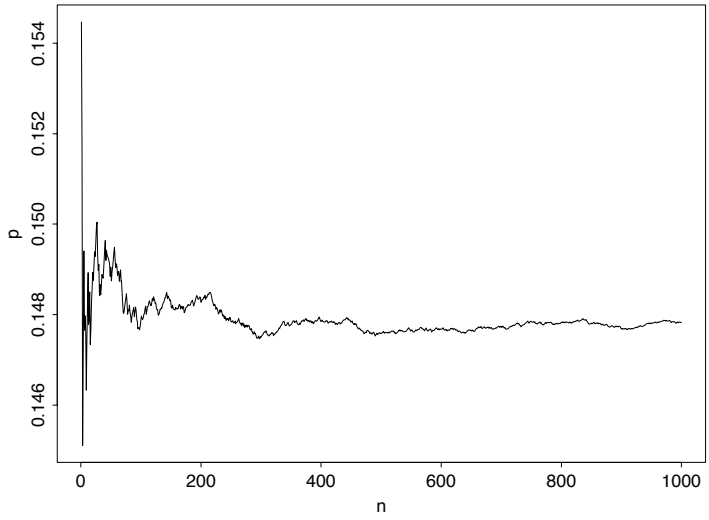

Figure 3.3: Convergence of importance sampled mean

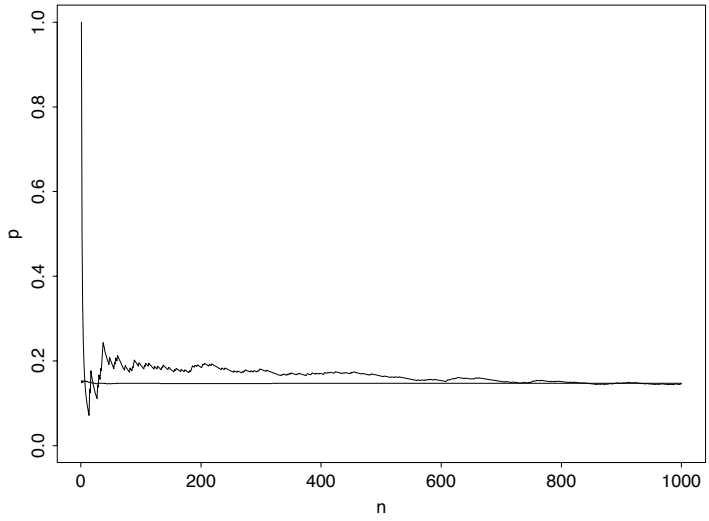

Figure 3.4: Comparison of importance sampled mean with standard estimator

It should be noted that this approximation puts some restrictions on the choice of g_2. To have a finite variance, we need (with $X' \sim g_2$)

$$E\left(\frac{(Mf(X'))^2}{g_2(X')^2}\right) = \int \frac{(Mf(x))^2}{g_2(x)}\,dx,$$

to be finite, i.e. f^2/g_2 is integrable. Hence, a natural requirement is that f/g_2 is bounded. This can now be used to approximate $\tau = E(\phi(X))$ using sequences $x_1, x_2, \ldots, x_N$ from g_2 and $y_1, y_2, \ldots, y_N$ from g through

$$\frac{t'_N}{t_N} = \left(\sum_{i=1}^{N} \frac{\phi(y_i)Mf(y_i)}{g(y_i)}\right) \Big/ \left(\sum_{i=1}^{N} \frac{Mf(x_i)}{g_2(x_i)}\right), \tag{3.20}$$

where the numerator approximates $M\tau$ and denominator M. Of course we could use $g = g_2$ and $x_i = y_i$ in (3.20), but this is not usually the most efficient choice.

Chapter 4

Markov Chain Monte-Carlo

Today, the most-used method for simulating from complicated and/or high-dimensional distributions is *Markov Chain Monte Carlo* (MCMC). The basic idea of MCMC is to construct a Markov Chain that has f as stationary distribution, where f is the distribution we want to simulate from. In this chapter we introduce the algorithms, more applications will be given later.

4.1 Markov chains - basic concepts

The sequences of random values, say X_n's, that we have obtained so far were obtained by independent sampling from a certain distribution. In our context this type of sampling was referred to as Monte Carlo sampling. The simplest but important case of this was a sequence of independent Bernoulli variables that models a random flip of a not necessarily symmetric coin. The limiting results of probability theory such as the law of large numbers or the central limit theorem have been used to establish some fundamental asymptotic properties (approximation errors) of the Monte Carlo method. Markov chains can be viewed as simplest models for obtained sequence of random observations that does not involve direct independent samples. The dependence in a sequence of experiments affecting the next value is only through the most recent value. Simplest Markov chains are those that takes values in a discrete (finite or countable) state-space.

More specifically, we take a sequence X_n's such that the distribution of X_{n+1} given that we obtained $X_n = x^{(n)}, \ldots, X_0 = x^{(0)}$ depends only on the value $x^{(n)}$ and not on $x^{(i)}$'s for $i < n$. The transition probabilities from the state i to j are given by

$$q(j|i) = P(X_{n+1} = j|X_n = i).$$

They together with the initial distribution distribution X_0 given by $\pi(i) = P(X_n = i)$ on the states i's fully described distributions of the model.

Example 4.1. For a simple example of a Markov chain, let us consider a simple case of three states -1,0,1 and the following matrix $\mathbf{P} = (p_{ij})$

representing the transition probabilities $p_{ij} = q_{j|i}$

$$\mathbf{P} = \begin{bmatrix} 1-2p & 2p & 0 \\ p & 1-2p & p \\ 0 & 2p & 1-2p \end{bmatrix}.$$

The following program simulates from this Markov chain that start from a state x_0.

```
SMC=function(n,p,x0){
  x=vector("numeric",n)
  x[1]=x0
  for(i in 2:n)
    {
   z=rmultinom(1,1,prob=c(p,1-2*p,p))
   if(x[i-1]==0){
     x[i]=z[1,1]-z[3,1]
   }else{
     if(x[i-1]==1){
       x[i]=x[i-1]-z[1,1]-z[3,1]
     }else{
       x[i]=x[i-1]+z[1,1]+z[3,1]
     }
   }
  }
  SMC=x
}
```

An example of sample can be obtained by running

```
n=100
p=1/4
x0=0
x=SMC(n,p,0)
plot(x)
```

and is shown in Figure 4.1 *Left*.

The theory of Markov chains demonstrates that much of asymptotics observed for independent samples are still valid for Markov chains. For example, in Figure 4.1 *Right* it is observed that a sort of law of large numbers should be valid for the Markov chain in hand as the asymptotic frequency of observing the state "1" is evidently converging. One can utilize the above program to observe the asymptotics

```
n=2000
p=1/4
x=SMC(n,p,1)
P1=cumsum(x==1)/cumsum(rep(1,n))
plot(P1,type='l')
```

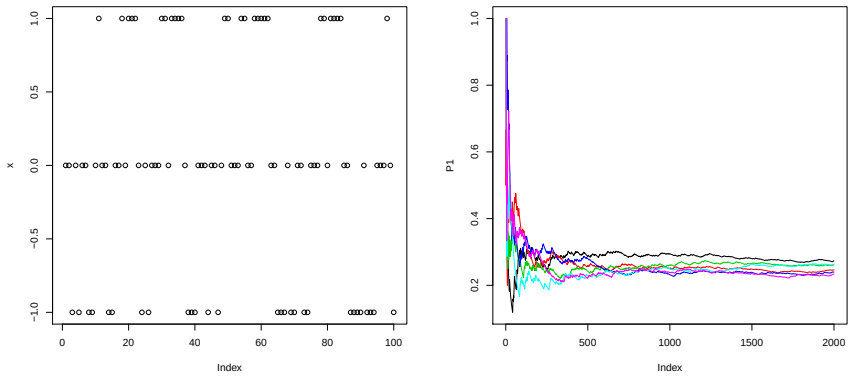

Figure 4.1: A simple 3-state Markov chain. *Left:* a trajectory of size 100 starting from $x^{(0)} = 0$, *Right:* asymptotic frequency of state "1" based on several trajectories of size 2000.

Markov Chains for which the law of large numbers holds are called ergodic.

Exercise 4.1. *Using the provided example of a Markov chain, make some claims about asymptotical values of the frequencies of states and provided with analysis of error of your claims based on Monte Carlo study.*

Markov Chains can serve often as simple models of real phenomena occurring in time. The following exercise can lead the reader through an attempt to model weather in her/his town. For this one needs a definition of a stationary state.

Definition 4.1. A distribution π_0 on the state space is called stationary if the process starting from that distribution remains in this distribution over the entire time, or more technically the row vector of probabilities given by π_0 satisfies the equation

$$\pi_0 \mathbf{P} = \pi_0.$$

Exercise 4.2. *Consider the following simplistic model for certain aspect of the weather that assumes the lack of memory property, i.e. that cloudeness and rain depends only on the next day depends only on what it was on the previous day. We consider five states: sunny (S) or partly sunny (P), cloudy (C), rainy (R), heavy rain (H). Because of the lack of memory property, this weather model is fully described by providing the matrix of transition probabilities, that describe what are chances for tomorrow to be in one of the five states under the conditions that today we observe one of these states.*

1. *Propose the values of the transition probabilities for summer weather in your town (use your own judgement, not necessarily scientific evidence).*

2. *Using the proposed values generate a 90 days long trajectory of weather pattern.*

3. *Based on your sample estimate the probabilities that on a randomly chosen day in summer the weather will be in one of its five states.*

4. *Check if your estimated values of probabilities satisfies the stationary state equation for the proposed Markov process.*

5. *Compare the estimated values with the evaluated theoretical values for the stationary state (the latter can be found by solving a proper linear equation).*

It should be remembered that not always a Markov chain leads to the asymptotics observed for independent sampling and the conditions for this have to be examined. An interested reader can do the following exercise to see possible problems.

Exercise 4.3 (*Random Walk).* *Consider a Markov Chain with infinite but discrete state space*

$$\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$$

and with transition probabilities given by

$$p_{ij} = \begin{cases} p & : \ j = i+1; \\ 1-p-q & : \ j = i; \\ q & : \ j = i-1; \\ 0 & : \ otherwise \end{cases}$$

Generate sample paths of such a Markov chain. By analyzing sample paths, discuss if such a Markov chain is ergodic.

4.2 Markov chains with continuous state-space

The theory for Markov chains that take values in a continuous state-space is complex. Much more so than the theory for finite/countable state-spaces, that you might have already been exposed to. We will here not pretend to give a full account of the underlying theory, but be happy with formulating a number of sufficient conditions under which the methods work. Links to more advanced material will be provided on the course web-page.

In this chapter, we will consider Markov-Chains $X_1, X_2, \dots, X_N$ defined through a starting value x_0 and a transition density $\tilde{q}$. The transition density $x_i \mapsto \tilde{q}(x_{i-1}, x_i) = g(x_i|x_{i-1})$ is the density of $X_i|X_{i-1} = x_{i-1}$, and this determines the evolution of the Markov-chain across the state-space. We would ideally like to draw the starting value x_0 and choose $\tilde{q}$ in such a way that the following realisations $x_1, x_2, \dots, x_N$ are independent draws from f, but this is a too ambitious task in general. In contrast to the previous chapter, here our draws will neither be independent nor *exactly* distributed

according to f. What we will require is that f is the *asymptotic distribution* of the chain, i.e. if X_i takes values in $\mathcal{X}$,

$$P(X_n \in A) \to \int_A f(x)\,dx, \tag{4.1}$$

as $n \to \infty$, for all subsets A of $\mathcal{X}$ and independently of the starting value $x_0 \in \mathcal{X}$.

A condition on the transition density that ensures the Markov chain has a stationary distribution f, is that it satisfies the *global balance condition*

$$f(y)\tilde{q}(y,x) = f(x)\tilde{q}(x,y). \tag{4.2}$$

Exercise 4.4. *Check that (4.2) results in a stationary distribution, i.e. if f satisfies it then*

$$f(y) = \int f(x)\tilde{q}(x,y)dx.$$

This says roughly that the flow of probability mass is equal in both directions (i.e. from x to y and vice versa). Global balance is not sufficient for the stationary distribution to be unique (hence the Markov chain might converge to a different stationary distribution). Sufficient conditions for uniqueness are irreducibility and aperiodicity of the chain, a simple sufficient condition for this is that the support of $f(y)$ is contained in the support of $y \mapsto \tilde{q}(x,y)$ for all x (minimal *necessary* conditions for $\tilde{q}$ to satisfy (4.1) are not known in general, however our *sufficient* conditions are far from the weakest known in literature).

4.3 Markov chain Monte-Carlo integration

Before going into the details of how to construct a Markov chain with specified asymptotic distribution, we will look at Monte-Carlo integration under the new assumption that draws are neither independent not exactly from the target distribution. Along this line, we need a Central Limit Theorem for Markov chains. First we observe that if $X_1, X_2, \ldots, X_n$ is a Markov chain on $\mathbf{R}^d$ and $\phi : \mathbf{R}^d \mapsto \mathbf{R}$, then with $Z_i = \phi(X_i)$, the sequence $Z_1, Z_2, \ldots, Z_n$ forms a Markov chain on $\mathbf{R}$. As before, we want to approximate $\tau = E(Z) = E(\phi(X))$ by $t_N = N^{-1}\sum_{i=1}^N z^{(i)}$, for a sequence $z^{(i)} = \phi(x^{(i)})$ of draws from the chain.

4.3.1 Burn-in

An immediate concern is that, unless we can produce a single starting value x^0 with the correct distribution, our sampled random variables will only *asymptotically* have the correct distribution. This will induce a bias in our estimate of τ.

In general, given some starting value $x^{(0)}$, there will be iterations $x^{(i)}$, $i = 1, \ldots, k$, before the distribution of $x^{(i)}$ can be regarded as "sufficiently close" to the stationary distribution $f(x)$ in order to be useful for further

analysis (i.e. the value of $x^{(i)}$ is still strongly influenced by the choice of $x^{(0)}$ when $i \leq k$). The values $x^{(0)}, \ldots, x^{(k)}$ are then referred to as the *burn-in* of the chain, and they are usually discarded in the subsequent output analysis.

For example when estimating $\int \phi(x) f(x)\, dx$, it is common to use

$$\frac{1}{N-k} \sum_{i=k+1}^{N} \phi(x^{(i)}).$$

An illustration is given in Figure 4.2. We now provide the CLT for Markov

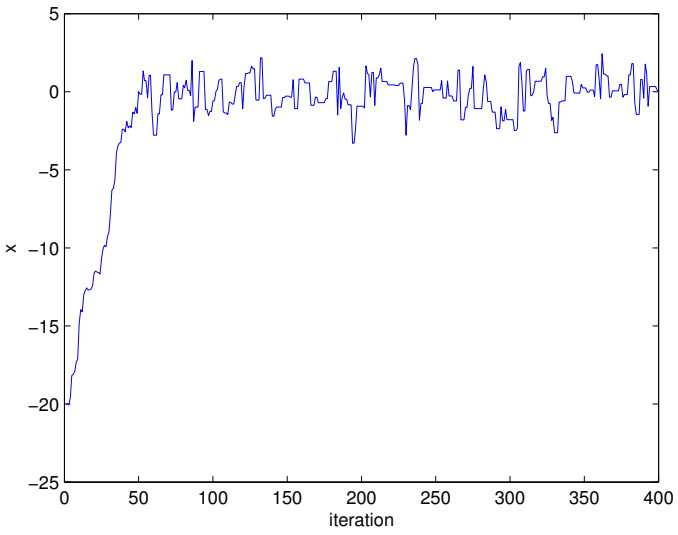

Figure 4.2: A Markov chain starting from $x^{(0)} = -20$ that seems to have reached its stationary distribution after roughly 70 iterations

Chains.

Theorem 4.1. *Suppose a geometrically ergodic Markov chain X_i, $i = 1, \ldots, n$, on $\mathbf{R}^d$ with stationary distribution f and a real-valued function $\phi : \mathbf{R}^d \mapsto \mathbf{R}$ satisfies $E(\phi^{2+\epsilon}(X)) \leq \infty$ for some $\epsilon > 0$, then with $\tau = E(\phi(X))$ and $T_n = n^{-1} \sum_{i=1}^{n} \phi(X_i)$*

$$P\big(\frac{\sqrt{n}(T_n - \tau)}{\sigma} \leq x\big) \to \Phi(x) \tag{4.3}$$

where Φ is the distribution function of the $N(0,1)$ distribution and

$$\sigma^2 = r(0) + 2 \sum_{i=1}^{\infty} r(i) \tag{4.4}$$

where $r(i) = \lim_{k \to \infty} Cov\{\phi(X_k), \phi(X_{k+i})\}$, the covariance function of a stationary version of the chain.

Note that the above holds for an arbitrary starting value x_0. The error due to "wrong" starting value, i.e. the bias $E(T_n) - \tau$, is of $O(n^{-1})$. Hence squared bias is dominated by variance, and asymptotically negligible. This does not suggest that it is unnecessary to discard burn-in, but rather that it is unnecessary to put too much effort into deciding on how long it should be. A visual inspection usually suffices.

4.3.2 After burn-in

If we manage to simulate our starting value $x^{(0)}$ from the target distribution f, all subsequent values will also have the correct distribution. Would our problems then be solved if we were given that single magic starting value with the correct distribution? As the above discussion suggests, the answer is no. The "big problem" of MCMC is that (4.4) can be very large, especially in problems where the dimension d is large. It is important to understand that converging to the stationary distribution (or getting close enough) is just the very beginning of the analysis. After that we need to do sufficiently many iterations in order for the variance σ^2/n of T_n to be small.

It is actually a good idea to choose starting values $x^{(0)}$ that are *far away* from the main support of the distribution. If the chain then takes a long time to stabilize you can expect that even after reaching stationarity, the autocorrelation of the chain will be high and σ^2 in (4.4) large. Here we give a rough guide for estimating σ^2, which is needed want we to produce a confidence interval:

1. We assume your draws $x^{(i)}$, $i = 1, \ldots, N$ are d-variate vectors. First make a visual inspection of each of the d trajectories, and decide a burn-in $1, \ldots, k$ where *all* of them seems to have reached stationarity. Throw away the first k sample vectors.

2. Now you want to estimate $E(\phi(X))$. Compute $z_i = \phi(x^{(i+k)})$, $i = 1, \ldots, N-k$ and estimate the autocorrelation function of the sequence z_i, i.e. the function $\rho(t) = \mathrm{Corr}(Z_1, Z_t)$ over a range of values $t = 1, \ldots, L$, this can be done with R-function `acf` (though it uses the range $-L, \ldots, L$). A reasonable choice of L is around $(N-k)/50$ (estimates of $\rho(L)$ tends to be too unreliable for larger values). If the estimated autocorrelation function does not reach zero in the range $1, \ldots, L$, go back to the first step and choose a larger value for N.

3. Divide your sample into m batches of size l, $ml = N-k$. Here l should be much larger than the time it took for the autocorrelation to reach zero in the previous step. The batches are $(z_1, \ldots, z_l)$,$(z_{l+1}, \ldots, z_{2l})$, ..., $(z_{m(l-l)+1}, \ldots, z_{ml})$. Now compute the arithmetic mean $\bar{z}_j$ of each batch, $j = 1, \ldots, m$. Estimate τ by the mean of the batch means and σ^2 by s^2/m, where s^2 is the empirical variance of the batch means.

This procedure is based on the fact that if batches are sufficiently large, their arithmetic means should be approximately uncorrelated. Hence, since our estimate is the mean of m approximately independent batch means it should have variance σ_b^2/m, where σ_b^2 is the variance of a batch mean.

We now turn to the actual construction of the Markov chains.

4.4 Two continuous time Markov chain models

4.4.1 Autoregressive model

Probably, the simplest continuous state is an autoregressive time series X_n that is given by

$$X_{n+1} = \rho X_n + \epsilon_n,$$

where ϵ_n are iid normal random variables with the mean zero and variance σ^2. One can easily argue that this is a Markov chain. Derivation of the transition densities is left for the reader, who can also study the model through simulations as suggested in the following exercise.

Exercise 4.5. *Program a simulator of the autoregressive model.*

- *By its means simulate trajectories of from such a model and consider for which values of the parameter ρ the model is stable.*
- *Using autocorrelation facilities of R approximate the autocorrelation function of X_n as well as its variance.*
- *Illustrate the central limit theorem that have been discussed in Theorem 4.1.*
- *Illustrate the burn in sample for this model.*
- *For the autoregressive model perform analysis suggested in 1.-3. in Subsection 4.3.2.*

4.4.2 Modeling cloud coverage

Daily cloud coverage is an important characteristics in weather studies. It can be expressed in the percentage of sunlight passing through the clouds relatively to the amount recorded when the sky is completely clear. Modeling such a process is extension of the simple model of Exercise 4.2. For modeling percentages the beta distribution is a natural family of distributions. The densities of this distributions are given up to a proportionality constant by

$$f(x; \alpha, \beta) \sim x^{\alpha-1}(1-x)^{\beta-1}, \quad x \in [0, 1].$$

We leave to the reader to develop a Markov model that would model cloud coverage using continuous state space and beta distributions in the spirit of Exercise 4.2 and Example 4.1

Exercise 4.6. *Propose a Markov chain approach to modeling cloud coverage using beta distributions. For the model develop programs and study its stability and asymptotic behavior.*

4.5 The Metropolis-Hastings algorithm

How do you choose transition density $\tilde{q}$ in order to satisfy (4.1)? The idea behind the Metropolis-Hastings algorithm is to start with an (almost) arbitrary transition density q. This density will not give the correct asymptotic distribution f, but we could try to repair this by rejecting some of the moves it proposes. Thus, we construct a new transition density $\tilde{q}$ defined by

$$\tilde{q}(x,y) = \alpha(x,y)q(x,y) + (1-\alpha(x,y))\delta_x(y), \tag{4.5}$$

where $\delta_x(y)$ is a point-mass at x. This implies that we stay at the level x if the proposed value is rejected and we reject a proposal y^* with probability $1-\alpha(x,y^*)$. Simulating from the density $y \mapsto \tilde{q}(x,y)$ works as follows

1. Draw y^* from $q(x,\cdot)$.
2. Draw u from $U(0,1)$.
3. If $u < \alpha(x,y^*)$ set $y = y^*$,
   else set $y = x$.

We now have to match our proposal density q with a suitable acceptance probability α. The choice of the *Metropolis-Hastings algorithm*, based on satisfying the global balance equation (4.2), is

$$\alpha(x,y) = \min(1, \frac{f(y)q(y,x)}{f(x)q(x,y)}), \tag{4.6}$$

you might want to check that this actually satisfies (4.2).

Algorithm 4.1 (The Metropolis-Hastings algorithm)**.**

1. Choose a starting value $x^{(0)}$.
2. Repeat for $i = 1, \ldots, N$:
   - i.1 Draw y^* from $q(x^{(i-1)},\cdot)$.
   - i.2 Draw u from $U(0,1)$.
   - i.3 If $u < \alpha(x^{(i-1)},y^*)$ set $x^{(i)} = y^*$, else set $x^{(i)} = x^{(i-1)}$.
3. $x^{(1)}, x^{(2)}, \ldots, x^{(N)}$ is now a sequence of dependent draws, approximately from f.

There are three general types of Metropolis-Hastings candidate generating densities q used in practise; the *Gibbs sampler*, *independence sampler* and the *random walk sampler*. Below we will discuss their relative merits and problems.

Exercise 4.7. *The goal is to evaluate the integral* $\int_0^\infty \frac{x \log x}{1+x} e^{-\sqrt{x}-x}\, dx$ *by implementing Metropolis-Hastings algorithm. Consider the transition densities given by* $q(x,y) = \frac{y}{x^2} e^{-y/x}$.

1. *Implement the Metropolis-Hastings algorithm using the above transition densities to generate a Markov chain with the stationary distribution proportional to the absolute value of the integrand.*
2. *Run this algorithm until you observe 1000 observations after the burn-in period of the algorithm.*
3. *Use the obtained samples to estimate the integral in question.*

4.6 The Gibbs-sampler

The Gibbs-sampler is often viewed as a separate algorithm rather than a special case of the Metropolis-Hastings algorithm. It is based on partitioning the vector state-space $\mathbf{R}^d = \mathbf{R}^{d_1} \times \cdots \mathbf{R}^{d_m}$ into blocks of sizes d_i, $i = 1, \ldots, m$ such that $d_1 + \cdots + d_m = d$. We will write $z = (z_1, \ldots, z_m) = (x_1, \ldots, x_d)$, where $z_i \in \mathbf{R}^{d_i}$ (e.g. $z_1 = (x_1, \ldots, x_{d_1})$. With this partition, the Gibbs-sampler with target f is now:

Algorithm 4.2 (The Gibbs-sampler)**.**

1. Choose a starting value $z^{(0)}$.
2. Repeat for $i = 1, \ldots, N$:
   - i.1 Draw $z_1^{(i)}$ from $f(z_1 | z_2^{(i-1)}, \ldots, z_m^{(i-1)})$.
   - i.2 Draw $z_2^{(i)}$ from $f(z_2 | z_1^{(i)}, z_3^{(i-1)}, \ldots, z_m^{(i-1)})$.
   - i.3 Draw $z_3^{(i)}$ from $f(z_3 | z_1^{(i)}, z_2^{(i)}, z_4^{(i-1)}, \ldots, z_m^{(i-1)})$.
   - $\vdots$
   - i.m Draw $z_m^{(i)}$ from $f(z_m | z_1^{(i)}, z_2^{(i)}, \ldots, z_{m-1}^{(i)})$.
3. $z^{(1)}, z^{(2)}, \ldots z^{(N)}$, is now a sequence of dependent draws approximately from f.

This corresponds to an MH-algorithm with a particular proposal density and $\alpha = 1$. What is the proposal density q?

Note the similarity with Algorithm 2.3. The difference is that here we draw each z_i conditionally on all the others which is easier since these conditional distributions are much easier to derive. For example,

$$z_1 \mapsto f(z_1 | z_2, \ldots, z_m) = \frac{f(z_1, \ldots, z_m)}{f(z_2, \ldots, z_m)} \propto f(z_1, \ldots, z_m).$$

Hence, if we know $f(z_1, \ldots, z_m)$, we also know all the conditionals needed (up to a constant of proportionality). The Gibbs-sampler is the most popular MCMC algorithm and given a suitable choice of partition of the state-space it works well for most applications to Bayesian statistics. We will have the opportunity to study it more closely in action in the subsequent part on Bayesian statistics of this course. Poor performance occurs when there is a high dependence between the components Z_i. This is due to the fact that the Gibbs-sampler only moves along the coordinate axes of the vector $(z_1, \ldots, z_m)$, illustrated by Figure 4.3. One remedy to this problem is to merge the dependent components into a single larger component, but this is not always practical.

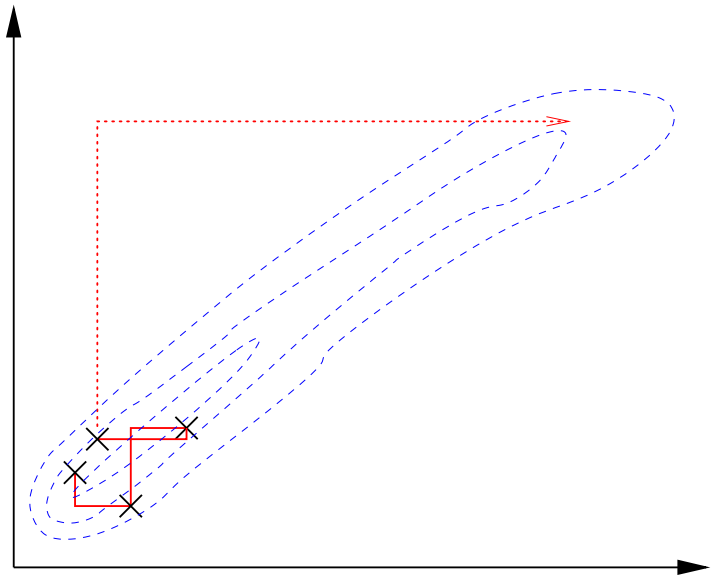

Figure 4.3: For a target (dashed) with strongly dependent components the Gibbs sampler will move slowly across the support since "big jumps", like the dashed move, would involve first simulating a highly unlikely value from $f(z_1|z_2)$.

Example 4.2 (Bivariate Normals)**.** Bivariate Normals can be drawn with the methods of Examples 2.4 or 2.2. Here we will use the Gibbs-sampler instead. We want to draw from

$$(X_1, X_2) \sim N_2\left(0, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\right),$$

and choose partition $(X_1, X_2) = (Z_1, Z_2)$. The conditional distributions are given by

$$Z_1|Z_2 = z_2 \sim N(\rho z_2, \sqrt{1-\rho^2}) \text{ and } Z_2|Z_1 = z_1 \sim N(\rho z_1, \sqrt{1-\rho^2}).$$

The following script draws 1000 values starting from $(z_1^{(0)}, z_2^{(0)}) = (0, 0)$.

```
bivngibbs=function(rho,n){
   z=matrix(0,nrow=n+1,ncol=2)

   for(i in 2:n+1)
   {
     z[i,1]=rnorm(1)*sqrt(1-rho^2)+rho*z[i-1,2]
     z[i,2]=rnorm(1)*sqrt(1-rho^2)+rho*z[i-1,1]
   }
   bivngibbs=z[2:(n+1),]
}

rho=0.9
n=1000
Z=bivngibbs(rho,n)

quartz()
par(mfrow=c(2,1))
plot(Z[,1],type='l')
plot(Z[,2],col='red',type='l')
```

In Figure 4.4 we have plotted the output for $\rho = 0.5$ and $\rho = 0.99$. Note the strong dependence between successive draws when $\rho = 0.99$. Also note that each individual panel constitute approximate draws from the same distribution, i.e. the $N(0, 1)$ distribution.

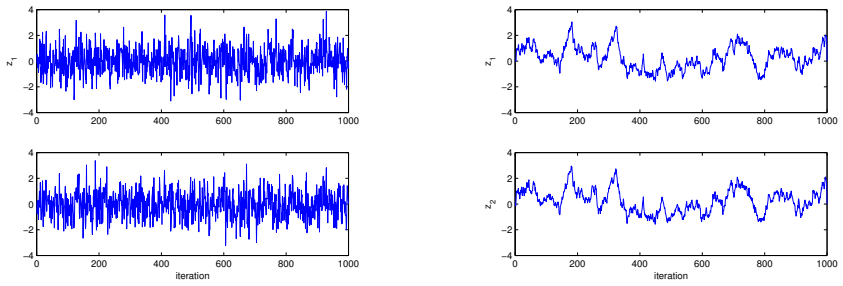

Figure 4.4: Gibbs-draws from Example 4.2, left with $\rho = 0.5$ and right with $\rho = 0.99$.

Exercise 4.8. *The above Gibbs sampler of normal distribution extends easily to higher dimensions. The advantage of it over previous methods of simulation is that one does not have to invert covariance matrices which maybe very beneficial for highly dimensional problems. Extend the algorithm to arbitrary number of dimensions and test the extension on an example in five dimensions. Plot estimates of variances and covariances over the iterations of the algorithm vs. their true values and explain how these plots can be used to determine the burn-in sample size.*

4.7 Independence proposal

The independence proposal amounts to proposing a candidate value y^* *independently* of the current position of the Markov Chain, i.e. we choose $q(x, y) = q(y)$. A necessary requirement here is that $\text{supp}(f) \subseteq \text{supp}(q)$; if this is not satisfied, some parts of the support of f will never be reached. This candidate is then accepted with probability

$$\alpha(x, y^*) = \begin{cases} \min\{\frac{f(y^*)q(x)}{f(x)q(y^*)}, 1\} & \text{if } f(x)q(y^*) > 0 \\ 1 & \text{if } f(x)q(y^*) = 0 \end{cases}$$

And we immediately see that if $q(y) = f(y)$, $\alpha(x, y) \equiv 1$, i.e. all candidates are accepted. Of course, if we really could simulate a candidate directly from $q(y) = f(y)$ we would not have bothered about implementing an MH algorithm in the first place. Still, this fact suggests that we should attempt to find a candidate generating density $q(y)$ that is as good an approximation to $f(y)$ as possible, similarily to the rejection sampling algorithm. The main difference is that we don't need to worry about deriving constants M or K such that $Mf < Kq$ when we do independence sampling. To ensure good performance of the sampler it is advisable to ensure that such constants *exists*, though we do not need to derive it explicitly. If it does not exist, the algorithm will have problems reaching parts of the target support, typically the extreme tail of the target. This is best illustrated with an example; assume we want to simulate from $f(x) \propto 1/(1+x)^3$ using an Exp(1) MH independence proposal. A plot of (unnormalised) densities q and $1/(1+x)^3$ in Figure 4.5 does not indicate any problems — the main support of the two densities seem similar. The algorithm is implemented as follows

```
indsamp=function(m,x0){
   x=vector('numeric',m)
   x[1]=x0
   acc=0
   for(i in 1:m)
   {
     y=rexp(1)
     a=((y+1)^(-3))*dexp(x[i])/dexp(y)/(x[i]+1)^(-3)
     a=min(a,1)

   u=runif(1)
   if(u<a){
     x[i+1]=y
     acc=acc+1
   }else{
     x[i+1]=x[i]
   }
   }
   indsamp=list(acc=acc,x=x)
}
```

```
x0=1
m=1000
IS=indsamp(m,x0)
IS$acc
```

and 1000 simulated values are shown in the left panel of Figure 4.6 with starting value `x0=1`. Looking at the output does not immediately indicate any problems either. However, a second run, now with starting value `x0=15`, is shown in the right panel of the same figure; 715 proposed moves away from the tail are rejected.

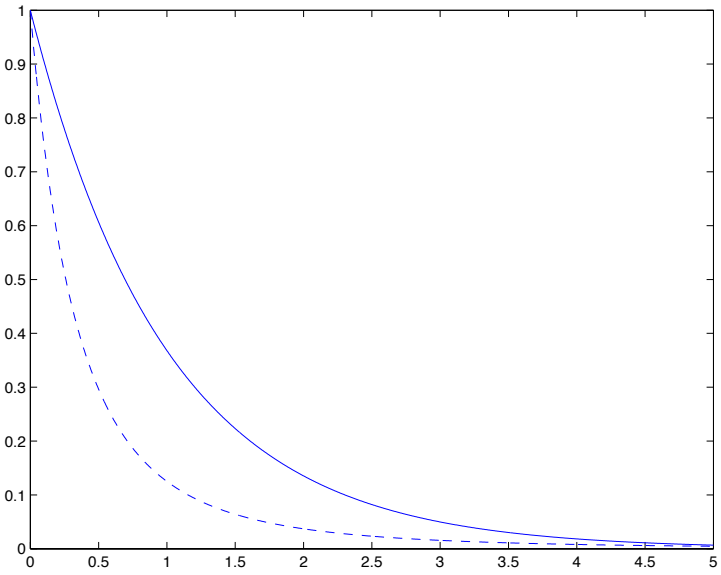

Figure 4.5: Unnormalized target $1/(1+x)^3$ (dashed) and Exp(1) independence proposal (solid).

The problem here is that since the light-tailed proposal density generates large values too seldom, the chain needs to stay in the tail for a very long time once it gets there to preserve stationarity. As a consequence this induces large autocorrelation which reduces the information contained in the output.

The main problem with the independence sampler is that unless q is a good approximation of f (and *especially so in high dimensional problems*), most proposals will be rejected and as a consequence autocorrelations high.

4.8 Random walk proposal

While the independence sampler needs careful consideration when choosing a candidate generating kernel q, random walk kernels are more "black box". As a drawback they will never be quite as efficient as a finely tuned independence proposal.

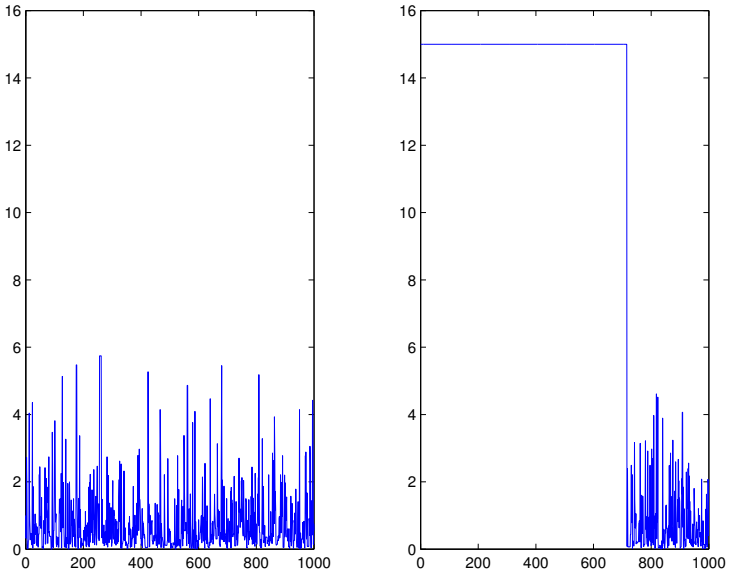

Figure 4.6: Output from independence sampler. Left with small starting value and right starting from the tail.

Random walk proposals are characterised by $q(x, y) = g(|x-y|)$, i.e. they are symmetric centered around the current value, and as a consequence α simplifies to

$$\alpha(x, y) = \min\{f(y)/f(x), 1\}.$$

Note especially that moves to areas with higher density, $f(y^*) > f(x)$, are always accepted.

A common choice is to let g be an $\mathrm{N}(0, s^2\Sigma)$ density with a suitably chosen covariance matrix Σ and a scaling constant s. In this case proposals are generated by adding a zero-mean Gaussian vector to the current state, $y^* = x + s\epsilon$ where $\epsilon \in N(0, \Sigma)$. What remains for the practitioner in this case is the choice of scaling constant s and covariance matrix Σ. The latter choice is difficult and in practise Σ is often chosen to be a diagonal matrix of ones. For s some general rules of thumb can be derived though. Lets first look at a simple example:

The function `rwmh()` implements a Gaussian random-walk for a standard Gaussian target density given input `N` number of iterations, `x0` starting value and `s` scaling constant:

```
rwmh=function(N,x0,s){
   x=vector('numeric',N)
   x[1]=x0
   acc=0
   for(i in 1:m)
   {
```

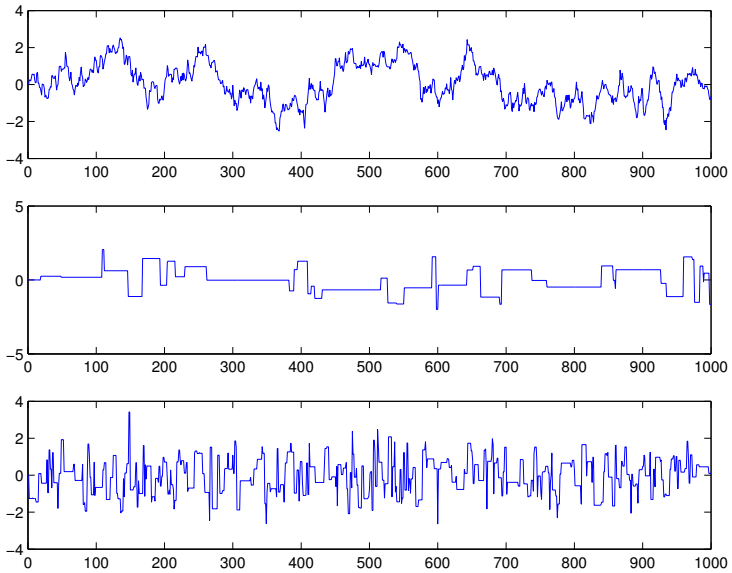

Figure 4.7: Random walks with too small s (top), too large s (middle) and well-tuned s (bottom)

```
    y=x[i]+s*rnorm(1)
    a=exp(-y^2/2+x[i]^2/2)
    a=min(a,1)
  u=runif(1)
  if(u<a){
    x[i+1]=y
    acc=acc+1
  }else{
    x[i+1]=x[i]
  }
  }
  rwmh=list(acc=acc,x=x)
}

x0=1
N=1000
s=30

RW=rwmh(N,x0,s)
RW$acc

quartz()
plot(RW$x,type='l')
plot(Z[,2],col='red',type='l')
```

In Figure 4.7 we have plotted outputs from 1000 iterations with `x0=0`

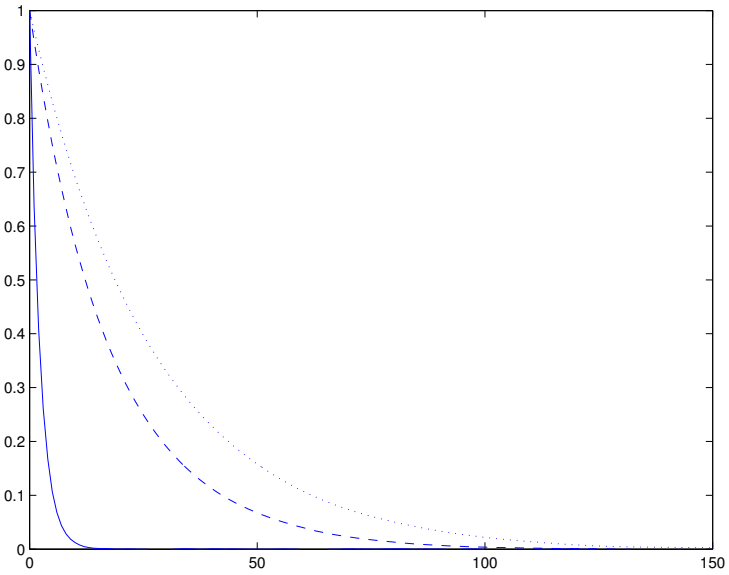

Figure 4.8: Autocorrelation functions for the output with $s = .3$ (dotted) $s = 30$ (dashed) and $s = 3$ (solid)

and scales `s` set to 0.3, 30 and 3. For the smallest scaling constant almost all of the proposed moves are accepted (908 out of 1000) but they are too small and the chain travels slowly across the support of f. For $s = 30$ very few proposals are accepted (48 out of 1000), but they are all very "innovative". Finally the result using $s = 3$ looks most promising. The methods can be compared by estimating the auto-correlation functions of the output as is done in Figure 4.8. Here it is clear that if our goal is to estimate the mean, this will be most efficient for $s = 3$.

Are there any general rules of thumb on how to choose random-walk scale s? The answer is yes, at least asymptotically. It turns out monitoring the acceptance rate is the key, and that for a large class of densities $f : \mathbf{R}^d \mapsto \mathbf{R}$, asymptotically, as $d \to \infty$, it is optimal to choose s in such a way that $23.4\ldots\%$ of the proposed moves are accepted (when $d = 1$ a slightly higher acceptance rate is often favourable). This is optimal for any fixed Σ.

Returning to the choice of Σ, note in Figure 4.9 that if we set Σ to be a diagonal matrix of ones, the random-walk sampler will have similar problems with dependent components as the Gibbs-sampler.

In addition, orthogonalising does not help unless we also standardise variances. A good choice of Σ is one that is similar to the covariance matrix of the target (up to a proportionality constant). An approximation that is often useful is to let Σ be proportional to the Hessian matrix of $\log f$ (i.e. $H(x)$ with entries $H_{i,j} = d(\log f)^2/(dx_i\, dx_j)$) evaluated at e.g. a mode of f if this can be found.

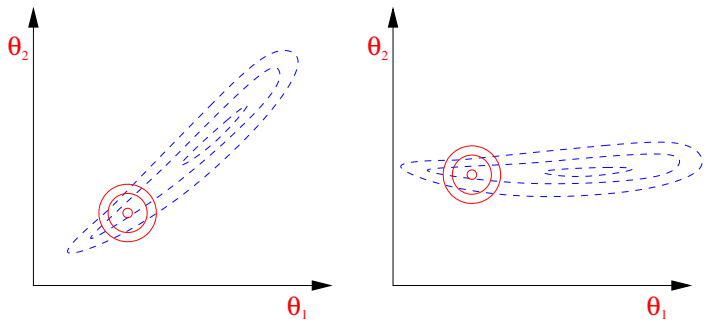

Figure 4.9: Left panel: With a symmetric random-walk proposal (solid contours) tuned to the optimal acceptance rate, movement will be slow if correlation of the posterior (dashed contours) is strong. Right panel: With variances different the similar problem occurs. .

4.8.1 Multiplicative random walk

If the target has a very heavy tail, the random walk proposal will generally perform poorly. In a similar fashion to the independence sampler with a light-tailed proposal, since it takes the chain a long time to travel all the way to the tail, it will stay there for a long time when it reaches it.

In this situation, a *Multiplicative random-walk proposal* is often more efficient. The random-walk proposal is formed by adding an independent component, $y^* = x^{(i-1)} + \epsilon$, the *multiplicative* random-walk instead multiplies with an independent component, $y^* = x^{(i-1)}\epsilon$. If we denote the density of ϵ by g, the proposal-generating density is $q(x, y) = g(y/x)/x$ and we accept/reject with probability

$$\alpha(x, y) = \min(1, \frac{f(y)g(x/y)x}{f(x)g(y/x)y}).$$

4.9 Hybrid strategies

In statistical problems there is often a natural choice of partition for the Gibbs-sampler, however one or more of the conditional distributions in Algorithm 4.2 might be difficult to sample from directly. In this case, an exact draw from the tricky conditionals can be replaced by one iteration of the Metropolis-Hastings algorithm. This strategy is often necessary in complex problems and is sometimes referred to as the Metropolis–within–Gibbs algorithm.

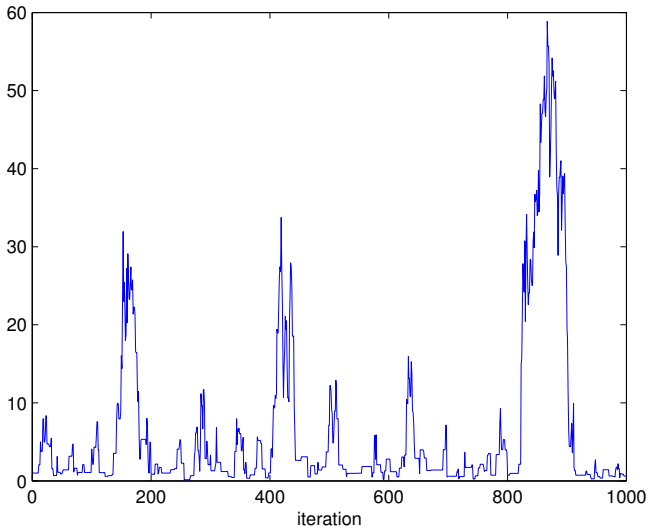

Figure 4.10: Typical behaviour of a random walk Metropolis Hastings on a heavytailed target. Seemingly stable behaviour is exchanged with long excursions in the tails.