Understanding MCMC, Lancaster 2003

Laboratory exercises

Why? The purpose of this lab session is to investigate some simple properties of MCMC algorithms by means of very simple algorithms. Though these are 'toy' problems, the issues that arise in these problems are characteristic of problems that occur widely in far more complex problems. Thus these exercises will give you a feel for issues to watch out for in MCMC implementation. So far, we have yet to say too much in lectures about convergence of algorithms. But soon, we shall describe some theory which allows us to describe the empirical behaviour you will observe in this session. So we hope that the lab session will serve to motivate the theory to come.

How? All the examples will be done in `R`. Many of you will already be experts in the use of `R` or its near neighbour `Splus`. However some will have never used the package before. So we'll just work with a relatively small set of simple functions that we've written. Hopefully, using these functions will be straightforward for all of you. More sophisticated users may wish to modify and improve these functions once you have some idea of what we're trying to illustrate.

What? There are many questions given in the exercises below. We're not looking for written answers, but the questions are really designed to focus your investigations. Please ask a tutor if you think you are missing the pointi, and also we'd encourage you to dicuss amoung yourselves your findings. Finally note that there are no definite 'right answers' to the questions, and there's always the chance (as in any stochastic experiment) of obtaining output which is atypical of the particular problem being looked at. So you might want to make sure you run each experiment at least twice!

Getting started To start off, enter `R` by typing `R`. The `R` prompt is just a forward arrow `>`. Load the time-series library by typing

```
> library(ts)
```

Now load the MCMC functions by typing

```
> source("Rcommands")
```

You are now ready to try the experiments described below.

1. The function `met` implements the random walk Metropolis algorithm on a one-dimensional target distribution. It takes 5 arguments which are respectively: the functional form of the target density, a function which produced a random proposed increment, starting value, number of iterations and a scaling parameter for the proposal distribution. Thus the command

   ```
   > met(dnorm, rnorm, 4, 1000, 3)
   ```

   will implement the random walk Metropolis algorithm on the standard normal density (with density given by `dnorm`) with $X_0 = 4$ and proposal increment distribution $N(0, 3^2)$. The algorithm is run for 1000 iterations outputing a vector of length 1001

(including X_0). Investigate the function and the algorithm it produces by first of all typing

```
> x<-met(dnorm, rnorm, 4, 1000, 3)
> plot(x, type="l")
> acf(x)
```

and then playing around with the arguments of the function to investigate different algorithms/target distributions/run lengths etc. You might wish to experiment with different proposals (eg Cauchy, `rcauchy`) and write your own target density, eg

```
> bimodal<-function(x)
> exp(-x^2/2) + exp(-(x-5)^2/2)
> x<-met(bimodal, rnorm, 4, 1000, 3)
```

In all cases, use the `acf` command to get a rough idea about the efficiency of your algorithm.

2. This experiment will investigate the choice of proposal scaling in Metropolis algorithms. Initially, we shall just focus on the problem of a standard normal density with standard normal proposal increment densities. By using acf plots or otherwise, try to find what you think is the best choice of scaling for the proposal in this problem. The proportion of accepted moves (the 'acceptance rate') is calculated using the `acceptance.rate` function, eg:

   ```
   > x<-met(dnorm,rnorm,0,10000,5)
   > acceptance.rate(x)
   ```

   What is the 'optimal acceptance rate in this problem?

   Now look at the function `opt.sca` and type:

   ```
   > opt.sca()
   ```

   The function plots a measure of the algorithms efficiency (its average squared jumping distance) as a function of the proposal distribution scaling, and as a function of the algorithm's acceptance rate. Running this function should confirm your findings above.

   If you understand the way this function works, try to modify the function and carry out the same analysis on other densities. What happens to the optimal acceptance rate for bimodal densities. Don't try this on the Cauchy density though. (Why?)

3. This exercise examines the transient phase of certain algorithms. The function `metGauss` is a specific version of `met` which just operates on a standard normal target density. For instance the command

   ```
   > x<-metGauss(rnorm, 12, 10000,3)
   ```

implements a Metropolis algorithm with Gaussian proposal with variance 3^2 started at 12 for 10000 iterations. Here we shall use `metGauss` in place of `met` since it is considerably more robust to numerical errors for algorithms in the tails of the target distribution.

Investigate the Metropolis distribution with Gaussian target with extremely large starting values, for instance

```
> x<-metGauss(rnorm,10000,20000,2.4)
> plot(x, type="l")
```

What's going on? Try different starting values, proposal scalings, and run lengths.

Now try the same experiment with Cauchy proposals. Comment on the difference between the output you observe.

4. This exercise will investigate the behaviour of the Metropolis algorithm on a heavy tailed distribution. Run a Metropolis algorithm started from the target distribution mode with Gaussian proposals and Cauchy target:

   ```
   > x<-met(dcauchy, rnorm, 0, 10000, 3)
   ```

   Examine your output in comparison to a sample of IID points from a standard Cauchy density:

   ```
   > y<-rcauchy(10000)
   > qqplot(x, y)
   ```

   Examine your output more closely and suggest what's really going wrong. Investigate trying to improve your algorithm by increasing the proposal variance.

   Now try starting the algorithm in the target distribution tail:

   ```
   > x<-met(dcauchy, rnorm, 10000, 10000, 100)
   ```

   Comment on the qualitative features of the output produced from this algorithm.

   Try some other heavy tailed distributions you can think of. Would you recommend the use of the Metropolis algorithm with Gaussian proposals on heavy tailed target distributions?

   Now consider other proposal types on the same problem including Cauchy proposals:

   ```
   > met(dcauchy, rcauchy, 10000, 0, 1)
   ```

   Comment on any differences in algorithm behaviour you observe.

5. The independence sampler on and Exponential(1) target density with Exponential(λ) proposals is implemented using the `independence.sampler` function: eg

   ```
   > independence.sampler(3, 1, 100)
   ```

   will run 100 iterations of the independence sampler with proposal density

$$q(y) = 3e^{-3y}, \quad y \geq 0.$$

The 1 in the arguments of the function is the initial value, the 3 refers to the proposal density, and the 100 refers to the run length. For different values of lambda (ranging from less than 1 to greater than 1) investigate the behaviour of the algorithm by examining trace plots, acf plots and anything else you feel is appropriate. Make sure you do multiple runs for each value of the proposal parameter λ. Can you see any qualitative difference between algorithms run with λ values greater than or less than 1? What happens with starting values close to 0? ∞? You should definitely investigate the effect of different starting values.

6. Following on from the above independence sampler investigation, `clt.i.s` is a function designed to examine the distribution of the Monte Carlo average produced from the independence sampler in this case. Specifically, `clt.i.s.(m, lambda )` starts m independent chains at 1 (the mean of the target distribution). Each chain is run for 10000 iterations:

$$\{X_{1,j}, X_{2,j} \ldots, X_{10000,j}\}, \quad 1 \leq j \leq m$$

and the function outputs the mean value of each chain:

$$\bar{X}_j = \frac{1}{10000} \sum_{i=1}^{10000} X_{ij}, \quad 1 \leq j \leq m \ .$$

Investigate the distribution of the Monte Carlo average for estimating the mean of X under the target distribution (remember the true value is 1) by examining the sample $\{\bar{X}_j,\ 1 \leq j \leq m\}$. You might find the function `density` useful for estimating the density of the Monte Carlo average, eg

```
> plot(density(clt.i.s.(30,5)))
```

which takes $\lambda = 5, m = 30$. Comment on your findings, and in particular in what situations might you expect a Markov chain central limit theorem to hold for this problem?