A black and white logo

Description automatically generated

**StratoBayes Workshop**

**18th – 20th March 2024, Durham, UK**

**Worksheet for Day 2 Session 2:**

**Bayes theorem and MCMC**

Prepared by Andrew Millard, Department of Archaeology, Durham University.

This work is licensed under a   
[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](http://creativecommons.org/licenses/by-nc-sa/4.0/).

[Creative Commons License](http://creativecommons.org/licenses/by-nc-sa/4.0/)

# Session 2: Bayes theorem and MCMC

This worksheet introduces some basic methods for investigating the output of MCMC. We will use some simple samplers based on normal distributions so that we can generate results very quickly, and concentrate on analysing the output.

## A simple Gibbs sampler

The file bvn\_gibbs.R provides a function to draw samples from a standard bivariate normal distribution using Gibbs sampling. Load the function by downloading the file into your working directory and using:

source("bvn\_gibbs.R")

If you like you can examine the code to see how it draws x and y samples alternately using a conditional probability distribution based on the value of the other variable. The parameters of the function are the number of iterations, the correlation of the bivariate normal, and optionally the initial values for the chain.

Run the sampler like this to generate a first set of outputs to examine:

mcmc\_run <- bvn\_gibbs(n = 2000, rho = 0.5)

With these settings it should run perfectly. In the next few steps we will check this!

Now let’s examine the output. In this case we know that the distribution should be normal, which we can check quickly with a histogram:

hist(mcmc\_run$x,breaks = 25)

hist(mcmc\_run$y,breaks = 25)

They should look roughly normally distributed. We can also look at the correlation to see if it is close to the value we specified:

cor(mcmc\_run)

We usually use MCMC because the posterior distribution is intractable, and we don’t know what it will look like, so these are not general tools for checking the output is satisfactory.

## Traceplots

The first tool for checking most MCMC output is a traceplot, showing the parameter value versus iteration number.

plot(mcmc\_run$x, type='l')

plot(mcmc\_run$y, type='l')

You can see that the chain of samples tends to move around the range of values quite well and does not get stuck in one place for long.

We can also visualise how the chain moves around the two-dimensional space:

plot(mcmc\_run, type = 'l')

The longer the run the less effective this is, but it can still sometimes be useful.

## Autocorrelation

Ideally we each sample would be independent of the previous one, but as MCMC samples the conditional posterior this is not the case. We can examine the extent of dependence by looking at the autocorrelation – the correlation between values at different lags

mcmc\_acf <- acf(mcmc\_run$x)

The acf function defaults to plotting the correlation at a range of lags. Here we can see that after lag 1 there is very little correlation between the samples.

## Effective sample size

Dependent samples in a chain do not convey the same amount of information as independent samples, especially if we want to calculate things like the standard error of the mean of the distribution. One approach to dealing with this is to assess the effective sample size. There are numerous proposals of how to do this, but one useful approach uses this formula based in the autocorrelations.

A number and number symbols

Description automatically generated with medium confidence

Here *n* is the number of samples and *ρ(k)* is the correlation at lag *k*.

Using the acf function we can calculate this quite easily (up the maximum lag it chose):

length(mcmc\_run$x) / (2\*sum(mcmc\_acf$acf) - 1)

The variable mcmc\_acf$acf lists the correlations from lag 0, but the lag 0 value is always 1, so we subtract 1 after doubling all the values rather adding 1 after doubling those from lag 1 onwards.

In this case we see that although we had 2000 iterations, the effective sample size is smaller. With a short chain like this the effective sample size is also quite variable between different runs.

This formula is not perfect, as sometimes it can give effective sample sizes larger than *n*, and it does not work if we need to combine two independent chains.

## Burn-in

When we initialise the chain with specific values we may not know where they lie within the distribution, and we may start the chain far from the high density region(s) of the distribution.

Rerun the sampler but this time start it far from the origin:

mcmc\_run <- bvn\_gibbs(n = 2000, rho = 0.5, inits = c(10,10))

Examine the traceplots and acf. The acf is not much different, but the traceplots show that the early iterations are different to the rest. This burn-in period where the chain has not reached a stationary distribution needs to be discarded. Deciding how long the burn-in should be is more of an art than a science, although there are some quantitative measures to guide decisions. Here we are safe discarding the first 100 iterations.

Reexamine the traceplots and acf leaving out the first 100 samples, for example, applying them to mcmc\_run$x[101:2000]. They should look more like those from the first run we did.

## Thinning

Having covered the basics of examining the output of MCMC, lets look at an example that more closely mimics the situation with complex models. Run with these settings:

mcmc\_run <- bvn\_gibbs(n = 2000, rho = 0.99, inits = c(10,10))

Look at traceplots, acf and effective sample size. They should all look rather different. This sampler needs to be run for a much longer time to obtain good results.

Re-run for 20000 samples and look at the traceplot. To be sure we have a sample uninfluenced by the initial values, we shall discard the first 5000 iterations as burn-in, and look again at traceplots and acf. The chain seems to have reached a stationary distribution, but it is highly autocorrelated.

This is a toy example, so we haven’t run the chain for so long that use of memory or filespace becomes an issue, but in a model with hundreds of parameters running for hundreds of millions of iterations this can be a problem. The solution is to thin the chain, either only saving every so many iterations, or post-processing to thin the sample.

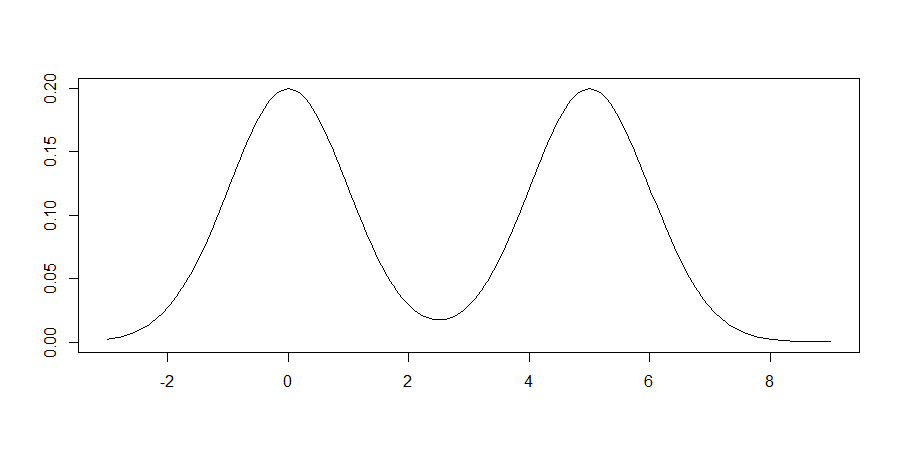
Let’s take every 20th sample after the first 5000:

mcmc\_thinned <- mcmc\_run[seq(5001, 20000, 20), ]

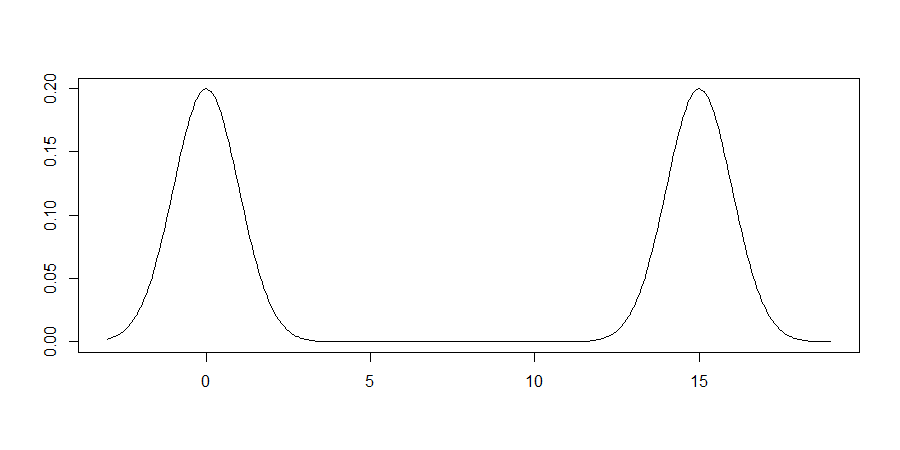
Now we have 750 samples from the 20,000 we generated, which is less than 4%, but the reduction in effective sample size is not nearly that drastic, and the other properties also look satisfactory.

## Multi-modal distributions

The file bimodal\_walker.R contains a function to run a random-walk Metropolis-Hastings algorithm to sample from bimodal distributions like the ones below, combining two normal distributions centred on zero and a supplied parameter b.



b = 5



b = 15

Downloading the file into your working directory and use source to load the function.

This Metropolis-Hastings algorithm samples by proposing a move from the current position. The move is drawn from a normal distribution with standard deviation prop\_sd. The proposal is accepted or rejected dependent on the relative probability densities at the current and proposed positions.

To sample the first distribution we can use:

mcmc\_bw <- bimodal\_walker(n = 20000, b = 5, prop\_sd = 1)

Then look at the output using:

plot(mcmc\_bw$x, type = 'b')

acf(mcmc\_bw$x)

hist(mcmc\_bw$x, breaks = 25)

There is significant autocorrelation, but the histogram looks good.

If we rerun with a wider proposal distribution, the results are much better:

mcmc\_bw <- bimodal\_walker(n = 20000, b = 5, prop\_sd = 4)

Finding the right proposal distribution can make a big difference to the efficiency of the Metropolis-Hastings algorithm.

Now let’s try the other distribution, with peaks centred on 0 and 15:

mcmc\_bw <- bimodal\_walker(n = 20000, b = 15, prop\_sd = 1)

The traceplot and autocorrelation of mcmc\_bw$x don’t look too bad, but the histogram shows that the chain never visited the peak at x = 15.

Increase prop\_sd to 4 and try again. This time the distribution should look better but the autocorrelation is high.

Other useful diagnostics are the Hastings ratios, which control whether the chain moves, and the proportion of proposals that were accepted:

plot(log(mcmc\_bw$HR), type='l')

mean(mcmc\_bw$x[-1]!=mcmc\_bw$x[-20000])

Ideally the Hastings ratios should rarely be very small. You might anticipate that the ideal acceptance rate is close to 1, but the accepted rule of thumb is that rates between 10% and 60% are best for MH algorithms, depending on the target distribution, and theory shows the optimal rate for MH sampling of a normal distribution is 23%.

If we increase prop\_sd to 14, acf and traceplots get a bit better still, but the acceptance rate drops.

For distributions like this, with widely spaced modes with low probability between them, a simple random walk sampler can never be efficient. There is always a risk that this sort of sampler fails to find all the modes of a multimodal distribution. A lot of the work in creating MCMC samplers goes into finding clever move-proposals to increase efficiency.

## Further reading

For a more comprehensive look at Bayesian modelling in an entertaining style, I recommend:

McElreath, Richard. 2016*. Statistical Rethinking: A Bayesian Course with Examples in R and Stan*. <https://xcelab.net/rm/statistical-rethinking/>