1 Bayesian Inference and Computation: A Beginner's Guide

BRENDON J. BREWER

1.1 Introduction

Most scientific observations are not sufficient to give us definite answers to all our questions. It is rare that we get a dataset which *totally* answers every question with certainty. Even if that did happen, we would quickly move on to other questions. What a dataset usually *can* do is make hypotheses more or less plausible, even if we do not achieve total certainty. Bayesian inference is a model of this reasoning process, and also a tool we can use to make quantitative statements about how much uncertainty we should have about our conclusions. This takes the mystery out of data analysis, because we no longer have to come up with a new method every time we face a new problem. Instead, we simply specify exactly what information we are going to use, and then compute the results. In the last 2 decades, Bayesian inference has become immensely popular in many fields of science, and astrophysics is no exception. Therefore it is becoming increasingly important for researchers to have at least a basic understanding of these methods.

Accessible textbooks for those with a physics background include those by Gregory (2005) and Sivia and Skilling (2006), and parts of the textbook by MacKay (2003).[1] The online tutorial by Vanderplas is also useful.[2] For those with a strong statistics background, I recommend the books by O'Hagan and Forster (2004) and Gelman et al. (2013). I also maintain a set of lecture notes for an undergraduate Bayesian statistics course.[3] The aim of this chapter is to present a fairly minimal yet widely applicable set of techniques to allow you to start using Bayesian inference in your own research.

Any particular application of Bayesian inference involves making choices about what data you are analysing, what questions you are trying to answer, and what assumptions you are willing to make. Data analysis problems in astronomy vary widely, so in this chapter we cannot cover a huge variety of examples. Instead, we only study a single example, but spend a lot of time looking at the methods and thinking that go into such an analysis, which will be applicable in other examples. The specific assumptions we make in the example will not always be appropriate, but they should be sufficient to show you the points at which assumptions are needed, and what you need to consider when you work on a particular problem.

In principle, it is usually best to work with your data in the most raw form possible, although this is often too difficult in practice. Therefore, most scientists work with data that has been processed (by a 'pipeline') and reduced to a manageable size. While many Bayesian practitioners often have strong ideals about data analysis, a large dose of pragmatism is still very necessary in the real world.

To do Bayesian inference, you need to specify what *prior information* you have (or are willing to assume) about the problem, in addition to the data. Prior information is necessary; what you can learn from a dataset depends on what you know about how it was produced. Once you have your data, and have specified your prior information, you

[1] The MacKay text is freely available online at `www.inference.phy.cam.ac.uk/itila`.
[2] Available online at `jakevdp.github.io/blog/2014/03/11/frequentism-and-bayesianism-a-practical-intro`.
[3] Available at `www.github.com/eggplantbren/STATS331`.

are faced with the question of how to calculate the results. Usually you want to calculate the *posterior distribution* for some unknown quantities (also known as 'parameters') given your data. This posterior distribution describes your uncertainty about the parameters, but takes the data into account.

Since we are often dealing with (potentially) complicated probability distributions in high-dimensional spaces, we need to *summarise* the posterior distribution in an understandable way. In certain problems, the summaries can be calculated analytically, but numerical methods are more general, so are the focus of this chapter. The most popular and useful numerical techniques are the Markov Chain Monte Carlo methods, often abbreviated as MCMC.[4] The rediscovery of MCMC in the 1990s is one of the main reasons why Bayesian inference is so popular now. While there were many strong philosophical arguments in favour of a Bayesian approach before then, many people were uncomfortable with the subjective elements involved. However, once MCMC made it easy to compute the consequences of Bayesian models more easily, people simply became more relaxed about these subjective elements.

A large number of MCMC methods exist, and it would be unwise to try to cover them all here. Therefore I focus on a small number of methods that are relatively simple to implement, yet quite powerful and widely applicable. I try to emphasise methods that are *general*, i.e. methods that work on most problems you might encounter. One disadvantage of this approach is that the methods we cover are not necessarily the most efficient methods possible. If you are mostly interested in one specific application, you will probably be able to achieve better performance by using a more sophisticated algorithm, or by taking advantage of the particular properties of your problem. There are many popular software packages (and many more unpopular ones) available for Bayesian inference, such as JAGS (Just Another Gibbs Sampler), Stan, emcee, MultiNest, my own DNest4, and many more. Please see the appendix for a brief discussion of the advantages and disadvantages of some of these packages.

1.2 Python

Due to its popularity and relatively shallow learning curve, I have implemented the algorithms in this chapter in the Python language. The code is written so that it works in either Python 2 or 3. The programs make use of the common numerical library `numpy`, and the plotting package `matplotlib`. Any Python code snippets in this chapter assume that the following packages have been imported:

```
import numpy as np
import numpy.random as rng
import matplotlib.pyplot as plt
import copy
import scipy.special
import numba
```

Full programs implementing the methods (and the particular problems) used in this chapter are provided online.[5]

[4] Some people claim that MCMC stands for Monte Carlo Markov Chains, but they are wrong.
[5] `https://github.com/eggplantbren/NSwMCMC`.

1.3 Parameter Estimation

Almost all data-analysis problems can be interpreted as *parameter estimation* problems. The term *parameter* has a few different meanings, but you can usually think of it as a synonym for *unknown quantity*. When you learned how to solve equations in high school algebra, you found the value of an unknown quantity (often called x) when you had enough information to determine its value with certainty. In science, we almost never have enough information to determine a quantity without any uncertainty, which is why we need probability theory and Bayesian inference.

Here, we denote our unknown parameters by θ, which could be a single parameter or perhaps a vector of parameters (e.g. the distance to a star and the angular diameter of the star). To start, we need to have some idea of the set of possible values we are considering. For example, are the parameters integers? Real numbers? Positive real numbers? In some examples, the definition of the parameters already restricts the set of possible values. For example, *the proportion of extrasolar planets in the Milky Way that contain life* cannot be less than 0 or greater than 1. Strictly speaking, it has to be a rational number, but it probably will not make much difference if we just say it is a real number between 0 and 1 (inclusive). The distance to a star (measured in whatever units you like) is presumably a positive real number, as is its angular diameter. The set of possible values you are willing to consider is called the *hypothesis space*.

To start using Bayesian inference, you need to assign a *probability distribution* on the hypothesis space, which models your initial uncertainty about the parameters. This probability distribution is called the *prior*. We then use Bayes' rule, a consequence of the product rule of probability, to calculate the *posterior* distribution, which describes our updated state of knowledge about the values of the parameters, after taking the data D into account.

For a prior distribution $p(\theta|I)$ (read as 'the probability distribution for θ given I'), and a sampling distribution $p(D|\theta, I)$, Bayes' rule allows us to calculate the *posterior distribution* for θ:

$$p(\theta|D, I) = \frac{p(\theta|I)p(D|\theta, I)}{p(D|I)}. \tag{1.1}$$

The I in Equation 1.1 refers to background information and assumptions; basically, it stands for everything you know about the problem apart from the data. The I appears in the background of all of the terms in Equation 1.1, and is often omitted. Note also that the notation used in Equation 1.1 is highly simplified but conventional among Bayesians; see the appendix for a discussion of notation. For brevity we can suppress the I (remove it from the right-hand side of all equations) and just write:

$$p(\theta|D) = \frac{p(\theta)p(D|\theta)}{p(D)}. \tag{1.2}$$

The result is a probability distribution for θ which describes our state of knowledge about θ after taking into account the data. The denominator, since it does not depend on θ, is a normalising constant, usually called the *marginal likelihood* or alternatively the *evidence*. Since the posterior is a probability distribution, its total integral (or sum, if the hypothesis space is discrete) must equal 1. Therefore we can write the marginal likelihood as:

$$p(D|I) = \int p(\theta|I)p(D|\theta, I)\, d\theta, \tag{1.3}$$

where the integral is over the entire N-dimensional parameter space.

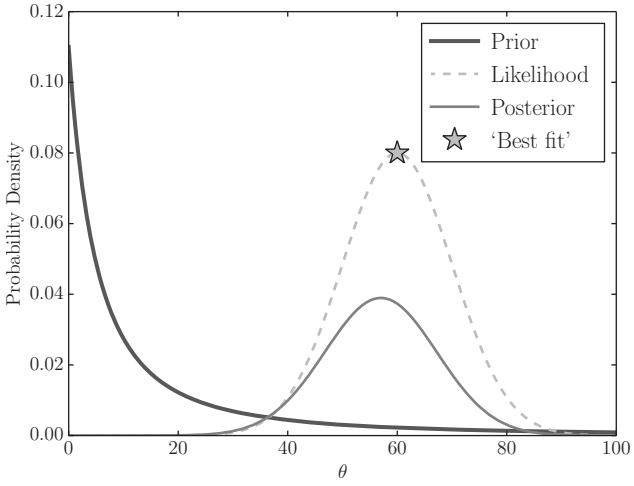

FIGURE 1.1 An example prior distribution for a single parameter (blue) gets updated to the posterior distribution (red) by the data. The data enters through the likelihood function (cyan dotted line). Many traditional 'best fit' methods are based on finding the maximum likelihood estimate, which is the peak of the likelihood function, here denoted by a star.

The posterior distribution is usually narrower than the prior distribution, indicating that we have learnt something from the data, and our uncertainty about the value of the parameters has decreased. See Figure 1.1 for an example of the qualitative behaviour we usually see when updating from a prior distribution to a posterior distribution.

1.4 Transit Example

To become more familiar with Bayesian calculations, we will work through a simple curve-fitting example. Many astronomical data-analysis problems can be viewed as examples of curve fitting. Consider a transiting exoplanet, like one observed by the Kepler space observatory. The light curve of the star shows an approximately constant brightness as a function of time, with a small dip as the exoplanet moves directly in front of the star. Clearly, real Kepler data is much more complex than this example, as stars vary in brightness in complicated ways, and the shape of the transit signal itself is more complex than the model we use here. Nevertheless, this example contains many of the features and complications that arise in a more realistic analysis.

The dataset, along with the true curve, is shown in Figure 1.2. The equation for the true curve is:

$$\mu(t) = \begin{cases} 10, & 2.5 \leq t \leq 4.5 \\ 5, & \text{otherwise.} \end{cases}$$

Assume we do not know the equation for the true curve (as we would not in reality), but we at least know that it is a function of the following form:

$$\mu(t) = \begin{cases} A - b, & (t_c - w/2) \leq t \leq (t_c + w/2) \\ A, & \text{otherwise,} \end{cases}$$

where A is the brightness away from the transit, b is the depth of the transit, t_c is the time of the centre of the transit, and w is the width of the transit.

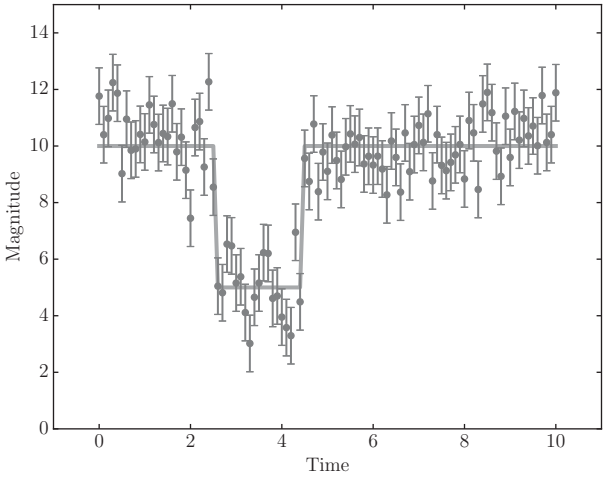

FIGURE 1.2 The 'transit' dataset. The red curve shows the model prediction based on the true values of the parameters, and the blue points are the noisy measurements.

Thus, the problem has been reduced from not knowing the true curve $\mu(t)$ to not knowing the values of 4 quantities (parameters) A, b, t_c, and w. Applying Bayesian inference to this problem involves calculating the posterior distribution for A, b, t_c, and w, given the data D. For this specific setup, Bayes' rule states:

$$p(A, b, t_c, w|D) = \frac{p(A, b, t_c, w)p(D|A, b, t_c, w)}{p(D)}. \tag{1.4}$$

So, in order for the posterior distribution to be well defined, we need to choose a prior distribution $p(A, b, t_c, w)$ for the parameters, and a sampling distribution $p(D|A, b, t_c, w)$ for the data. Since the denominator $p(D)$ is not a function of the parameters, it plays the role of a normalising constant that ensures the posterior distribution integrates to 1, as any probability distribution must.

1.4.1 Sampling Distribution

The *sampling distribution* is the probability distribution we would assign for the data if we knew the true values of the parameters. A useful way to think about the sampling distribution is to write some code whose input is the true parameter values, and whose output is a simulated dataset. Whatever probability distribution you use to simulate your dataset is your sampling distribution.

In many situations, it is conventional to assign a normal distribution (also known as a Gaussian distribution) to each data point, where the mean of the normal distribution is the noise-free model prediction, and the standard deviation of the normal distribution is given by the size of the error bar. Later, we will see how to relax these assumptions in a useful way. The probability density for the data given the parameters (i.e. the sampling distribution) is:

$$p(D|A, b, t_c, w) = \prod_{i=1}^{N} \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left[-\frac{1}{2\sigma_i^2}\left(D_i - \mu(t_i)\right)^2\right]. \tag{1.5}$$

This is a product of N terms, one for each data point, and is really a probability distribution over the N-dimensional space of possible datasets. We have assumed that each data point is *independent* (given the parameters). That is, if we knew the parameters and a subset of the data points, we would use *only* the parameters (not the data points) to predict the remaining data points.

When the dataset is known, Equation 1.5 becomes a function of the parameters only, known as the *likelihood function*. The curve predicted by the model, here written as $\mu(t_i)$ (where I have suppressed the implicit dependence on the parameters), provides the mean of the normal distribution. Remember that the independence assumption is not an assumption about the actual dataset, but an assumption about our prior information about the dataset. It does not make sense to say that a particular dataset is or is not independent. Independence is a property of probability distributions.

Equation 1.5 is the sampling distribution (and the likelihood function) for our problem, but it is fairly cumbersome to write. Statisticians have developed a shorthand notation for writing probability distributions. This is extremely useful for communicating your assumptions without having to write the entire probability-density equation. To communicate Equation 1.5, we can simply write:

$$D_i \sim \mathcal{N}\left(\mu(t_i), \sigma_i^2\right), \tag{1.6}$$

i.e. each data point has a normal distribution (denoted by $\mathcal{N}$) with mean $\mu(t_i)$ (which depends on the parameters) and standard deviation σ. For the normal distribution, it is traditional to write the *variance* (standard deviation squared) as the second argument, but since the standard deviation is a more intuitive quantity (being in the same units as the mean), we often literally write the standard deviation, squared (e.g. 3^2). For other probability distributions the arguments in the parentheses are whatever parameters make sense for that family of distributions.

1.4.2 Priors

Now we need a prior for the unknown parameters A, b, t_c, and w. This is a probability distribution over a 4-dimensional parameter space. To simplify things, we can assign independent priors for each parameter, and multiply these together to produce the joint prior:

$$p(A, b, t_c, w) = p(A)p(b)p(t_c)p(w). \tag{1.7}$$

This prior distribution models our uncertainty about the parameters before taking into account the data. The independence assumption implies that if we were to learn the value of one of the parameters, this would not tell us anything about the others. This may or may not be realistic in a given application, but it is a useful starting point.

Another useful starting point for priors is the uniform distribution, which has a constant probability density between some lower and upper limit. Use 4 uniform distributions for our priors:

$$A \sim U(-100, 100) \tag{1.8}$$
$$b \sim U(0, 10) \tag{1.9}$$
$$t_c \sim U(t_{\min}, t_{\max}) \tag{1.10}$$
$$w \sim U(0, t_{\max} - t_{\min}). \tag{1.11}$$

The full expression for the joint prior probability density is:

$$p(A, b, t_c, w) = \begin{cases} \frac{1}{2000(t_{\max} - t_{\min})^2}, & (A, b, t_c, w) \in S \\ 0, & \text{otherwise,} \end{cases} \tag{1.12}$$

where S is the set of allowed values. Even more simply, we can ignore the normalising constant and the prior boundaries and just write:

$$p(A, b, t_c, w) \propto 1, \tag{1.13}$$

although if we use this shortcut, we must remember that the boundaries are implicit.

Now that we have specified our assumed prior information in the form of a sampling distribution and a prior, we are ready to go. By Bayes' rule, we have an expression for the posterior distribution immediately:

$$p(A, b, t_c, w|D) \propto p(A, b, t_c, w)p(D|A, b, t_c, w), \tag{1.14}$$

which is proportional to the prior times the likelihood. In the likelihood expression we substitute the actual observed dataset into the equation, so that it is a function of the parameters only. The main problem with using Bayes's rule this way is that a mathematical expression for a probability distribution in a 4-dimensional space is not very easy to understand intuitively. For this reason, we usually calculate summaries of the posterior distribution. The main computational tool for doing this is MCMC.

1.5 Markov Chain Monte Carlo

Monte Carlo methods allow us to calculate any property of a probability distribution that is an expectation value. For example, if we have a single variable x with a probability density $p(x)$, the expected value is

$$\mathbb{E}(x) = \int_{-\infty}^{\infty} xp(x)\, dx \tag{1.15}$$

which is a measure of the 'centre of mass' of the probability distribution.

If we had a set of points $\{x_1, x_2, \ldots, x_N\}$ 'sampled from' $f(x)$, we could replace the integral with a simple average:

$$\mathbb{E}(x) \approx \frac{1}{N} \sum_{i=1}^{N} x_i. \tag{1.16}$$

In 1 dimension, this may not seem very useful. Evaluating a 1-dimensional integral analytically is often possible, and doing it numerically using the trapezoidal rule (or a similar approximation) is quite straightforward. However, Monte Carlo really becomes useful in higher-dimensional problems. For example, consider a problem with five unknown quantities with probability distribution $p(a, b, c, d, e)$, and suppose we want to know the probability that a is greater than $b + c$. We could do the integral

$$P(a > b + c) = \int p(a, b, c, d, e)\mathbb{1}\left(a > b + c\right)\, da\, db\, dc\, dd\, de \tag{1.17}$$

where $\mathbb{1}\left(a > b + c\right)$ is a function that is equal to 1 where the condition is satisfied and 0 where it is not. However, if we could obtain a sample of points in the five-dimensional space, the Monte Carlo estimate of the probability is simply

$$P(a > b + c) \approx \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\left(a_i > b_i + c_i\right), \tag{1.18}$$

which is just the fraction of the samples that satisfy the condition.

Another important use of Monte Carlo is marginalisation. Suppose again that we have a probability distribution for 5 variables, but we only care about 1 of them. For example, the marginal distribution of a is given by

$$p(a) = \int p(a, b, c, d, e)\, db\, dc\, dd\, de \tag{1.19}$$

which describes your uncertainty about a, rather than your uncertainty about all of the variables. This integral might be analytically intractable. With Monte Carlo, if you have samples in the five-dimensional space but you only look at the first coordinate, then you have samples from $p(a)$. This is demonstrated graphically in Figure 1.3.

In Bayesian inference, the most important probability distribution is the posterior distribution for the parameters. We would like to be able to generate samples from the posterior, so we can compute probabilities, expectations, and other summaries easily. MCMC allows us to generate these samples.

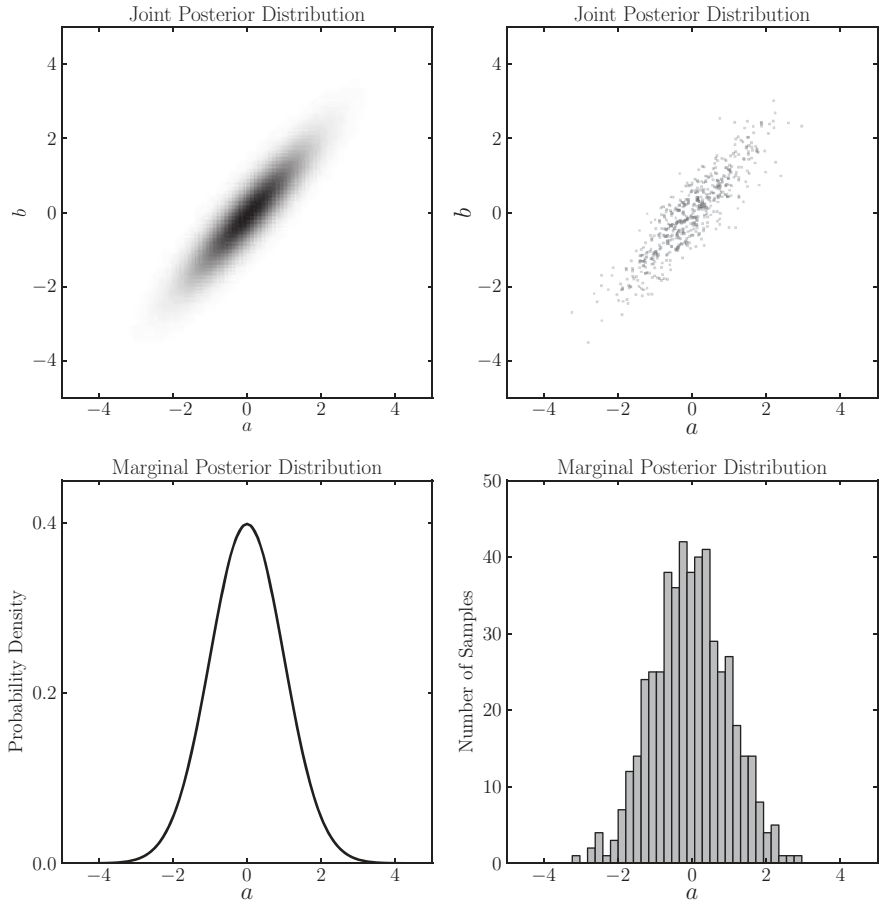

FIGURE 1.3 An example posterior distribution for 2 parameters a and b, taken from my STATS 331 undergraduate lecture notes. The full joint distribution is shown in the top left, and the marginal distribution for a (bottom left) is calculated by integrating over all possible b values, a potentially non-trivial calculation. The top right panel has points drawn from the joint posterior. Points drawn from the marginal posterior (bottom right) are obtained by ignoring the b values of the points, a trivial operation.

1.5.1 The Metropolis Algorithm

The Metropolis-Hastings algorithm, also known as the Metropolis algorithm, is the oldest and most fundamental MCMC algorithm. It is quite straightforward to implement, and works well on many problems. The version of the Metropolis algorithm presented here is sometimes called *random walk* Metropolis. More sophisticated choices are possible, but usually require problem-specific knowledge.

Consider a problem with unknown parameters θ. If the prior is some density $\pi(\theta)$ and the likelihood function is $L(\theta)$, then the posterior distribution will be proportional to $\pi(\theta)L(\theta)$. The marginal likelihood $Z = \int \pi(\theta)L(\theta)\, d\theta$ is unknown, but the Metropolis algorithm does not need to know it: all we need is the ability to evaluate π and L at a given position in the parameter space. The Metropolis algorithm tells us how to move a 'particle' around the parameter space so that we eventually sample the posterior distribution. That is, the amount of time spent in any particular region of parameter space will be approximately proportional to the posterior probability in that region. Note the change of notation from $p(\theta)$, $p(D|\theta)$, and $p(D)$ to π, L, and Z, respectively. This is a convention when discussing computational methods (as opposed to discussing priors, datasets, etc.).

The Metropolis algorithm can be summarised as follows:

(i) Choose a starting position θ, somewhere in the parameter space.

(ii) Generate a proposed position θ' from a proposal distribution $q(\theta'|\theta)$. A common choice is a 'random walk' proposal, where a small perturbation is added to the current position.

(iii) With probability $\alpha = \min\left(1, \frac{q(\theta|\theta')}{q(\theta'|\theta)} \frac{\pi(\theta')}{\pi(\theta)} \frac{L(\theta')}{L(\theta)}\right)$, accept the proposal (i.e. replace θ with θ'). Otherwise, do nothing (i.e. remain at θ).

(iv) Repeat steps (i)–(iii) until you have enough samples.

When a proposed move is rejected (and the particle remains in the same place), it is important to count the particle's position again in the output. This is how the algorithm ends up spending more time in regions of high probability: moves *into* those regions tend to be accepted, whereas moves *out* of those regions are often rejected. The Metropolis algorithm is quite straightforward to implement and I encourage you to attempt this yourself if you haven't done so before.

Python code implementing the Metropolis algorithm is given below (this code has been stripped of features for keeping track of the output, and shows just the algorithm itself). Note several features. First, the functions used to measure the prior density and likelihood of any point, and the function to generate a proposal in the first place, are problem specific and assumed to have been implemented elsewhere. Second, for numerical reasons we deal with the (natural) log of the prior density, the likelihood, and the acceptance probability. Third, note how the `log_prior` and `log_likelihood` functions only need to be called once per iteration, not twice as one might naively think. Finally, no q ratio is required in the acceptance probability: we assume that we are working with a *symmetric* proposal distribution, where the probability of proposing a move to position a given the current position is b is the same as the probability of the reverse (proposing b when at position a).

```
# Generate a starting point (if you have a good guess, use it)
# In the full version of the code, the initial point is drawn
# from the prior.
params = np.array([1., 1., 1., 1.])
logp, logl = log_prior(params), log_likelihood(params)
```

```
# Total number of iterations
steps = 100000

# Main loop
for i in range(0, steps):
  # Generate proposal
  new = proposal(params)

  # Evaluate prior and likelihood for the proposal
  logp_new = log_prior(new)
  logl_new = -np.Inf
  # Only evaluate likelihood if prior prob isn't zero
  if logp_new != -np.Inf:
    logl_new = log_likelihood(new)

  # Acceptance probability
  log_alpha = (logl_new - logl) + (logp_new - logp)
  if log_alpha > 0.:
    log_alpha = 0.

  # Accept?
  if rng.rand() <= np.exp(log_alpha):
    params = new
    logp = logp_new
    logl = logl_new
```

The 'random walk' proposal generates a proposed value θ' from a normal (Gaussian) distribution centred around the current position θ. The user is free to choose the width of the normal distribution. Here is a Python code snippet showing a proposal with width `L`:

```
# Generate a proposal
proposal = theta + L*rng.randn()
```

The performance of the Metropolis algorithm depends quite strongly on the width of the proposal distribution. If the width is too small, most moves will be accepted, but will not move very far. If the width is too large, most moves will be rejected, so you end up stuck in 1 place. Some authors recommend using preliminary runs to find an optimal width. Instead, I recommend that you use a mixture of widths. Basically, every time we make a proposal, the width is drawn from some range, rather than being constant. The biggest possible width we would ever want should be roughly the order of magnitude of the width of the prior (since the posterior is usually narrower than the prior). It is rare that we would need proposals many orders of magnitude smaller than that. My default suggestion is to randomise the logarithm of the step size, as in this code snippet:

```
# A heavy-tailed proposal distribution
# Generate a standard deviation
L = 10.**(1.5 - 6.*rng.rand())

# Use the standard deviation for the proposal
proposal = theta + L*rng.randn()
```