# Data analysis recipes: Using Markov Chain Monte Carlo*

**David W. Hogg**
*Simons Center for Data Analysis, Simons Foundation*
*Center for Cosmology and Particle Physics, Department of Physics, New York University*
*Center for Data Science, New York University*
*Max-Planck-Institut für Astronomie, Heidelberg*

**Daniel Foreman-Mackey**
*Sagan Fellow*
*Department of Astronomy, University of Washington*

**Abstract**

Markov Chain Monte Carlo (MCMC) methods for sampling probability density functions (combined with abundant computational resources) have transformed the sciences. Here we give a brief overview of the most basic MCMC method and then turn to practical advice for the use of MCMC in real inference problems. We give advice on method choice, tuning for performance, methods for initialization, tests of convergence, troubleshooting, and use of the chain output to produce or report parameter estimates with associated uncertainties.

## 1 When do you need MCMC?

Markov Chain Monte Carlo (MCMC) methods are methods for sampling probability distribution functions or probability density functions (pdfs). These pdfs may be either probability mass functions on a discrete space, or probability densities on a continuous space, though we will concentrate on the latter in this *Chapter*. MCMC methods don't require that you have a full analytic description of the properly normalized pdf for sampling to proceed; they only require that you be able to compute ratios of the pdf at pairs of locations. This makes MCMC methods ideal for sampling *posterior pdfs* in probabilistic inferences:

In a probabilistic inference, the posterior pdf $p(\theta \,|\, D)$, or pdf for the parameters $\theta$ given the data $D$, is constructed from the likelihood $p(D \,|\, \theta)$, or

*The notes begin on page 41, including the license[1] and the acknowledgements[2]

pdf for the data given the parameters, and the prior pdf $p(\theta)$ for the parameters by what's often known as "Bayes rule",

$$p(\theta \,|\, D) \;=\; \frac{1}{Z}\, p(D \,|\, \theta)\, p(\theta) \quad . \tag{1}$$

In these contexts, the constant $Z$, sometimes written as $p(D)$, is known by the names "evidence", "marginal likelihood", "Bayes integral", and "prior predictive probability", and is usually *extremely hard to calculate*.[3] That is, you often know the function $p(\theta \,|\, D)$ up to a constant factor; you can compute ratios of the pdf at pairs of points, but not the precise value at any individual point.

In addition to this normalization-insensitive property of MCMC, in its simplest forms it can be run without computing any derivatives or integrals of the function, and (as we will show below in Section 3) in its simplest forms it is *extremely easy to implement*. For all these reasons, MCMC is ideal for sampling posterior pdfs in the real situations in which scientists find themselves.

Say you are in this situation: You have a huge blob of data $D$ (think of this as a vector or list or heterogeneous collection of observations[4]). You also have a model sophisticated enough—a probabilistic, generative model, if you will[5]—that, given a setting of a huge blob of parameters[6] $\theta$, you can compute a pdf for data (or likelihood[7]) $p(D \,|\, \theta)$. Furthermore, say also that you can write down some kind of informative or vague prior pdf $p(\theta)$ for the parameter blob $\theta$. If all these things are true, then—even if you can't compute anything else—in principle a trivial-to-implement MCMC can give you a fair sampling of the posterior pdf. That is, you can run MCMC (for a very long time—see Section 5 for how long) and you will be left with a set of $K$ parameter-blob settings $\theta_k$ such that the full set $\{\theta_k\}_{k=1}^{K}$ constitutes a fair sampling from the posterior pdf $p(\theta \,|\, D)$.

All that said, and adhering to the traditions of the *Data Analysis Recipes* project[8], we are compelled to note at the outset that MCMC is in fact *overused*. Because MCMC provably (under assumptions[9], some of which will be discussed) samples the full posterior pdf in all of parameter space, many investigators use MCMC because it will sample *all* of the parameter space ($\theta$-space). That is, they are using MCMC because they want to *search the parameter space for good models*. This is not a good reason to use MCMC! Another bad use case is the following: Because MCMC samples the parameter representatively, it spends most of its time near very good models; models

that (within the confines of the prior pdf) do a good job of explaining the data. For this reason, many investigators are using MCMC because it effectively *optimizes the posterior pdf*, or, for certain choices of prior pdf, *optimizes the likelihood*.[10] This is another bad reason!

Both of these reasons for using MCMC—that it is a parameter-space search algorithm, and that it is a simple-to-code effective optimizer—are *not good reasons*. MCMC is first and foremost a *sampler*. If you are trying to find the optimum of the likelihood or the posterior pdf, you should use an *optimizer*, not a sampler. If you want to make sure you search all of parameter space, you should use a *search algorithm*, not a sampler. MCMC is good at one thing, and one thing only: Sampling ill-normalized (or otherwise hard to sample) pdfs.

In what follows, we are going to provide a kind of "user manual" or advice document or folklore capture regarding the use of MCMC for data analysis. This will not be a detailed description of multiple MCMC methods (indeed, we will only explain one method in detail), and it will not be about the mathematical properties or structure of the method or methods. It will be about how to use MCMC, including diagnosis and trouble-shooting.

The first couple of Sections will describe what a sampling is, and how the simplest MCMC method, the Metropolis-Hastings algorithm, can provide one. The next few Sections will provide ideas about how to initialize, tune and operate MCMC methods for good performance. The last few Sections will provide advice for making decisions among the myriad MCMC methods implementations, how to implement a good likelihood function and prior pdf function for inference, and how to trouble-shoot standard kinds of problems that arise in operating MCMC methods on real problems. We will leave parenthetical and philosophical matters to the end-notes, which appear after the main text.

# 2 What is a sampling?

Heuristically, a sampling $\{\theta_k\}_{k=1}^K$ from some pdf $p(\theta)$ is a set of $K$ values $\theta_k$ that are independent draws from the pdf. Heuristically, if an enormous (large $K$) sampling could be displayed in a finely binned $\theta$-space histogram, the histogram would look—up to a total normalization—just like the original function $p(\theta)$.

Not being mathematicians, we don't know the precise definition of a pdf[11],

but for our purposes here a pdf is any single-valued (scalar) function that is non-negative everywhere (the entire domain of $\theta$), and obeys a normalization condition

$$0 \leq p(\theta) \quad \text{for all } \theta \tag{2}$$

$$1 = \int p(\theta)\,\mathrm{d}\theta \quad , \tag{3}$$

where, implicitly, the integral is over the full domain of $\theta$. Importantly, although $p(\theta)$ is single-valued, $\theta$ can be a scalar or a multi-element vector or list or blob; it can be arbitrarily large and complicated. In data-analysis contexts, $\theta$ will often be the full blob of free parameters in the model. Implicitly, the integral in equation (3) is high-dimensional; it has as many dimensions as there are elements or entries or components of $\theta$. Also, if there are elements or entries or components of $\theta$ that are discrete (that is, take on only integer values or equivalent), then along those dimensions the integral becomes a discrete sum. This latter is a detail to which we return below.

Given this pdf $p(\theta)$, we can define *expectation values* $E_{p(\theta)}[\theta]$ for $\theta$ or for any quantity that can be expressed as a function $q(\theta)$ of $\theta$:

$$E_{p(\theta)}[\theta] \equiv \int \theta\, p(\theta)\,\mathrm{d}\theta \tag{4}$$

$$E_{p(\theta)}[q(\theta)] \equiv \int q(\theta)\, p(\theta)\,\mathrm{d}\theta \quad , \tag{5}$$

where again the integrals are implicitly definite integrals over the entire domain of $\theta$ (all the parts of $\theta$ space in which $p(\theta)$ is finite) and the integrals are multi-dimensional if $\theta$ is multi-dimensional. These expectation values are the mean values of $\theta$ and $q(\theta)$ under the pdf. A good sampling—and really this is the definition of a good sampling—makes the sampling approximation to these integrals accurate. With a good sampling $\{\theta_k\}_{k=1}^K$ the integrals get replaced with sums over samples $\theta_k$:

$$E_{p(\theta)}[\theta] \approx \frac{1}{K}\sum_{k=1}^{K}\theta_k \tag{6}$$

$$E_{p(\theta)}[q(\theta)] \approx \frac{1}{K}\sum_{k=1}^{K} q(\theta_k) \quad . \tag{7}$$

That is, a sampling is good when any expectation value of interest is accurately computed via the sampling approximation. The word "accurately"

here translates into some kinds of theorems about limiting behavior; the general idea is that the sampling approximation becomes exact as $K$ goes to infinity. The size $K$ of the sampling you need *in practice* will depend on the expectations you want to compute, and the accuracies you need.

As we noted above (Section 1), in the context of MCMC, we are often using some *badly normalized* function $f(\theta)$. This function is just the pdf $p(\theta)$ multiplied by some unknown and hard-to-compute scalar. In this case, for our purposes, the conditions on $f(\theta)$ are that it be non-negative everywhere and have *finite* integral $Z$

$$0 \leq f(\theta) \quad \text{for all } \theta \tag{8}$$

$$Z = \int f(\theta)\,\mathrm{d}\theta \quad . \tag{9}$$

When the sampling $\{\theta_k\}_{k=1}^K$ is of one of these badly normalized functions $f(\theta)$—as it usually will be—the sampling-approximation expectation values are the expectation values under the properly-normalized corresponding pdf, even though you might *never learn that normalization*. When we run MCMC sampling on $f(\theta)$, a function that differs from a pdf $p(\theta)$ by some unknown normalization constant, then the sampling permits computation of the following kinds of quantities:

$$E_{p(\theta)}[q(\theta)] \equiv \frac{\int q(\theta)\,f(\theta)\,\mathrm{d}\theta}{\int f(\theta)\,\mathrm{d}\theta} \tag{10}$$

$$E_{p(\theta)}[q(\theta)] \approx \frac{1}{K}\sum_{k=1}^{K} q(\theta_k) \quad . \tag{11}$$

That is, the sampling can be constructed (as we will show below in Section 3) from evaluations of $f(\theta)$ directly, and it permits you to compute expectation values without ever requiring you to integrate either the numerator integral or the denominator integral, both of which are generally intractable.[12]

The "correctness" of a sampling is defined (above in Section 2) in terms of its use in performing integrals or approximate computation of integrals. In a deep sense, the *only* thing a sampling is good for is computing integrals. There are many uses of the MCMC sampling, some of them good and some of them bad. Most of the good or sensible uses will somehow involve integration.

For example, one magical property of a sampling (in a $D$-dimensional space) is that a *histogram* (a $D$-dimensional histogram) of the samples (divided, if you like, by the number of samples in each bin and the bin width[13]

looks very much like the pdf from which the samples were drawn. This is a way to "reconstruct" the pdf from the sampling: Make a histogram from the samples. Even in this case, the sampling is being used to do *integrals*; the (possibly odd) idea is that the approximate or effective value of the pdf in each bin of the histogram is an average over the bin. That average is obtained by performing an integral.

Integrals are also involved in finding the mean, median, and any quantiles of a pdf. They are not involved in finding the *mode* of a pdf. For this reason (and others), in what follows, when we talk about what to report about the outcome of your MCMC sampling, we will advise in favor of mean, median, and quantiles, and we will advise against mode.

Finally—and perhaps most importantly—a magical property of a sampling (in a $D$-dimensional space) is that if some of your $D$ dimensions are totally uninteresting (nuisance parameters, if you will) and some of your $D$ dimensions are of great interest, the sampling in the full $D$-space is trivially converted into a sampling in the subspace of interest: You just drop from each $\theta_k$ vector (or blob or list) the dimensions of no interest! That is, the projection of the sampling to the subspace of interest produces a sampling of the marginalized pdf, marginalizing (or projecting) out the nuisance parameters.[14] That is extremely important for inference, where there are always parameters with very different levels of importance to the scientific conclusions. This point will return again below (Section 8).

Although the discussion in this *Chapter* is general, the most common use of MCMC sampling (for us, anyway) is in probabilistic inference. For this reason, we will often refer to the function $f(\theta)$ colloquially as "the posterior pdf"[15] even though it is implicitly ill-normalized and might not be a posterior pdf in any sense. We will also occasionally assume—just because it is true in inference—that the function $f(\theta)$ is the product of two functions, one called "the prior pdf" and one called "the likelihood". Again, this usage is colloquial and is only strictly correct in inference contexts with proper inputs. That the function $f(\theta)$ can be thought of as a product of a prior pdf and a likelihood is only necessary for what follows in the context of advanced sampling techniques like tempering or nested sampling, both mentioned briefly below (Section 9).

**Problem 1:** Look up (or choose) definitions for the mean, variance, skewness, and kurtosis of a distribution. Also look up or compute the analytic

values of these four statistics for a top-hat (uniform) distribution. Write a computer program that uses some standard package (such as *numpy*[16]) to generate $K$ random numbers $x$ from a uniform distribution in the interval $0 < x < 1$. Now use those $K$ numbers to compute a the sampling estimate of the mean, variance, skewness, and kurtosis (four estimates) Make four plot of these four estimates as a function of $1/K$ or perhaps $\log_2 K$, for $K = 4^n$ for $n = 1$ up to $n = 10$ (that is, $K = 4$, $K = 16$, and so on up to $K = 1048576$). Over-plot the analytic answers. What can you conclude?

# 3 Metropolis–Hastings MCMC

The simplest algorithm for MCMC is the Metropolis–Hastings algorithm (M–H MCMC).[17] It is so simple that we recommend that any reader of this document who has not previously implemented the algorithm take a break at the end of this Section and implement it forthwith, in a short piece of computer code, in the context of some simple problems.[18]

The M–H MCMC algorithm requires two inputs. The first is a handle to the function $f(\theta)$ that is the function to be sampled, such that the algorithm can evaluate $f(\theta)$ for any value of the parameters $\theta$. In data-analysis contexts, this function would be the prior $p(\theta)$ times the likelihood $p(D \,|\, \theta)$ evaluated at the observed data $D$. The second input is a proposal pdf $q(\theta' \,|\, \theta)$ that can be sampled, such that the algorithm can draw a new position $\theta'$ in the parameter space given an "old" position $\theta$. This second function must meet a symmetry requirement (detailed balance) we discuss further below. It permits us to random-walk around the parameter space in a fair way.

The algorithm[19] is the following: We have generated some set of samples, the most recent of which is $\theta_k$ To generate the next sample $\theta_{k+1}$ do the following:

- Draw a proposal $\theta'$ from the proposal pdf $q(\theta' \,|\, \theta_k)$.
- Draw a random number $0 < r < 1$ from the uniform distribution.
- If $f(\theta')/f(\theta_k) > r$ then $\theta_{k+1} \leftarrow \theta'$; otherwise $\theta_{k+1} \leftarrow \theta_k$.

That is, at each step, either a new proposed position in the parameter space gets accepted into the list of samples or else the previous sample in the parameter space *gets repeated*. The algorithm can be iterated a large number $K$ of times to produce $K$ samples.

Why does this algorithm work? The answer is not absolutely trivial[20], but there are two components to the argument: The first is that the Markov process delivers a unique stationary distribution. The second is that that stationary distribution is proportional to the density function $f(\theta)$.

This algorithm—and indeed any MCMC algorithm—produces a *biased random walk* through parameter space. It is a random walk for the same reason that it is "Markov": The step it makes to position $\theta_{k+1}$ depends only on the state of the sampler at position $\theta_k$ (and no previous state). It is a biased random walk, biased by the acceptance algorithm involving the ratios of function values; this acceptance rule biases the random walk such that the amount of time spent in the neighborhood of location $\theta$ is proportional to $f(\theta)$. Because of this local (Markov) property, the nearby samples are not independent; the algorithm only produces fair samples in the limit of arbitrary run time, and two samples are only independent when they are sufficiently separated in the chain (more on this below in Section 5).

The principal user-settable knob in M–H MCMC is the proposal pdf $q(\theta' \,|\, \theta)$. A typical choice is a multi-variate Gaussian distribution for $\theta'$ centered on $\theta$ with some simple (diagonal, perhaps) variance tensor. We will discuss the choice and tuning of this proposal distribution below (Section 6 and Section 10).

Importantly—for the algorithm given above to work correctly—the proposal pdf must satisfy a "detailed balance" condition; it must have the property that

$$q(\theta' \,|\, \theta) \quad = \quad q(\theta \,|\, \theta') \quad ; \tag{12}$$

that is, it must be just as easy to go one way in the parameter space as the other. You can break this property, if you like, and then adjust the acceptance condition accordingly (which is truly the Metropolis–Hastings algorithm), but we *do not recommend this* except under the supervision of a trained professional.[21] The reason is: It is one thing to draw samples from $q(x' \,|\, x)$. It is another thing to correctly write down $q(x' \,|\, x)$. If you violate detailed balance in $q(x' \,|\, x)$ then you have to be able to *both* draw from *and* write down $q(x' \,|\, x)$ and you are subject to a set of new possible bugs for your code. If the detailed balance condition frightens you (as it frightens us), then just stick with pdfs that are symmetric in $\theta'$ and $\theta$, like the Gaussian, or a centered uniform distribution and forget about it.

In what follows, when we discuss tuning of the proposal pdf, we will often cycle through the parameter blob components and propose changes in only

one dimension or element or component at a time. That is, at step $k+1$ you might only propose a one-dimensional move in the $i$th dimension, not in all $D$ dimensions of the parameter space. That won't change anything significant; but if you are implementing this right now you might want to do that. It will help us tune the proposal pdf (Section 6) and diagnose problems (Section 10).

One note to make here is that you must *propose* in the same coordinates (parameterization or transformation of parameters) as that in which your priors are *specified*, or else multiply in a (possibly nasty) Jacobian. That is, if your prior is "flat in $\theta$" but you find it easier to run your sampler by proposing steps in $\ln\theta$, if you don't modify your acceptance probability by the Jacobian, your *real* prior won't be flat in $\theta$. These think-os can get subtle; the best way to avoid them is to write your code and your likelihood function and your prior pdf function and your proposal distribution all in precisely the same parameterization. We will return to this point below in our comments on testing (Section 10).

As we discuss briefly below (Section 9), there are many MCMC methods more advanced than M–H. However, they all share characteristics with M–H (initialization, tuning, judging convergence), such that it is very valuable to understand M–H well before using anything more advanced. Furthermore, a scientist new to MCMC benefits enormously from building, tuning, and using her or his own MCMC software. One piece of advice we give then, to the new user of MCMC, is to code up, tune, and use a M–H MCMC sampler for a scientific project. A huge amount is learned in doing this, and it is very often the case that the home-built M–H MCMC does everything needed; you often don't need any more advanced tool. Only after you have concluded that your home-built M–H MCMC sampler is not suited to your project (or not developed properly into a properly versatile software package) should you download and start to use any professionally developed alternative. That is, even if—in the end—you want to leave the sampling code to the experts and run an industrial-strength code, it is still valuable to build your own, given the simplicity of the algorithm, and given the intuition you gain by doing it (at least once) yourself.[22]

Because many problems involve a huge amount of dynamic range in the density function $f(\theta)$, and we like to avoid underflows and overflows (in, say, ratios computed for accept–reject steps), it is often advisable to work in the (natural) logarithm of the density rather than the density. When working in logarithmic density, the accept–reject step would change from a comparison

of a random deviate with a ratio of probabilities to comparison of the log of a random deviate with a difference of log probabilities. That is, the accept–reject step becomes:

- If $\ln f(\theta') - \ln f(\theta_k) > \ln r$ then $\theta_{k+1} \leftarrow \theta'$; otherwise $\theta_{k+1} \leftarrow \theta_k$.

This protects you from underflow, but exposes you to some (negative) infinities, if you end up taking the logarithm of a zero. It is important to write your code to be infinity-safe.[23]

HOGG SAY HERE: ...You must describe your priors in the same parameterization that you "step" in. That is, if you change your parameters, you *must* change your priors correspondingly. Give the example of going from $A$, $\phi$ to $A\cos\phi$, $A\sin\phi$.

**Problem 2:** In your scientific programming language of choice, write a very simple M-H MCMC sampler. Sample in a single parameter $x$ and give the sampler as its density function $p(x)$ a Gaussian density with mean 2 and variance 2. (Note that variance is the *square* of the standard deviation.) Give the sampler a proposal distribution $q(x' \,|\, x)$ a Gaussian pdf with mean zero and variance 1. Initialize the sampler with $x = 0$ and run the sampler for more than $10^4$ steps. Plot the results as a histogram, with the true density over-plotted sensibly. The resulting plot should look something like Figure 1.

**Problem 3:** Re-do Problem 2 but now with an input density that is uniform on $3 < x < 7$ and zero everywhere else. The plot should look like Figure 2. What change did you have to make to the initialization, and why?

**Problem 4:** Re-do Problem 2 but now with an input density that is a function of two variables $(x, y)$. For the density function use two different functions. *(a)* The first density function is a covariant two-dimensional Gaussian density with variance tensor

$$V = \begin{bmatrix} 2.0 & 1.2 \\ 1.2 & 2.0 \end{bmatrix} \quad . \tag{13}$$

*(b)* The second density function is a rectangular top-hat function that is uniform on the joint constraint $3 < x < 7$ and $1 < y < 9$ and zero everywhere

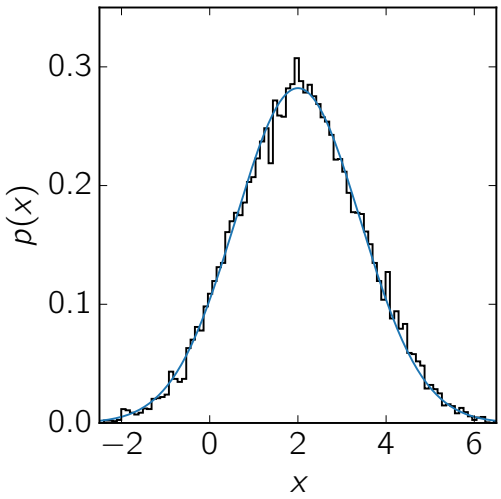


Figure 1.— Solution to Problem 2: MCMC samples from a Gaussian (black) and the true distribution (blue).

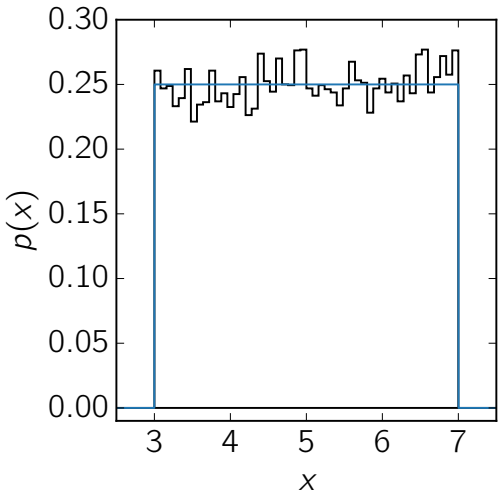


Figure 2.— Solution to Problem 3: MCMC samples from a uniform distribution (black) and the true distribution (blue).

else. For the proposal distribution $q(x', y' \,|\, x, y)$ a two-dimensional Gaussian density with zero mean and variance tensor set to the two-dimensional identity matrix. Plot the two one-dimensional histograms and also a two-dimensional scatter plot for each sampling. Figure 3 shows the expected results for the Gaussian. Make a similar plot for the top-hat.

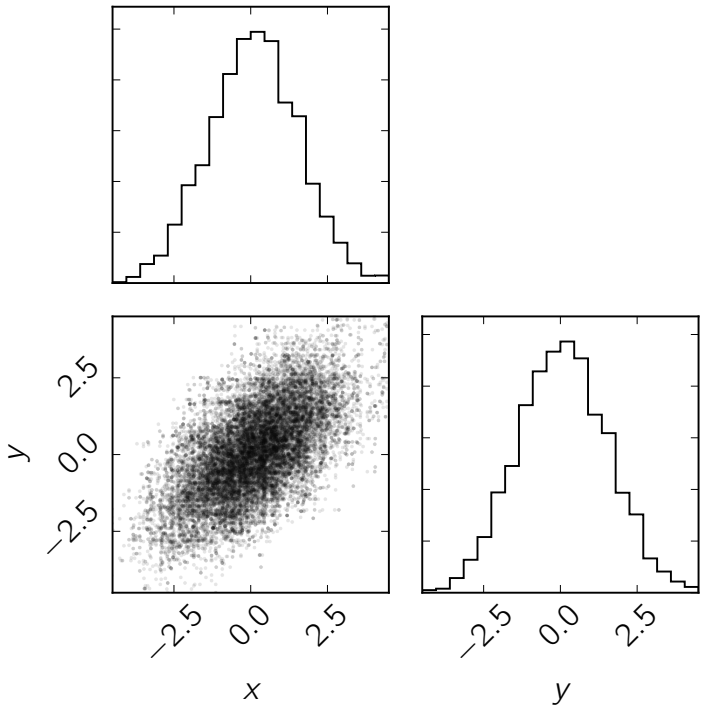


Figure 3.— Solution to Problem 4a: MCMC samples from a two-dimensional Gaussian (scatter plot) and the one-dimensional marginalized distributions (histograms).

**Problem 5:** Re-do Problem 4a but with different values for the variance of the proposal distribution $q(x' \,|\, x)$. What happens when you go to very extreme values (like for instance $10^{-1}$ or $10^2$)?

**Problem 6:** Why, in all the previous problems, did we give the proposal distributions $q(x' \,|\, x)$ zero mean? What would be bad if we hadn't done that? Re-do Problem 4a with a proposal $q(x' \,|\, x)$ with a non-zero mean and

see what happens. Bonus points: Modify the acceptance–rejection criterion to deal with the messed-up $q(x' \mid x)$ and show that everything works once again.

# 4 Likelihoods and priors

MCMC is used to obtain samples $\theta_k$ from a pdf $p(\theta)$, given a badly normalized all-positive function $f(\theta)$ that is different from the pdf by an unknown factor $Z$. In the context of data analysis, MCMC is usually being used to obtain samples $\theta_k$ that are *parameter values* $\theta$ for a probabilistic model of some data. That is, MCMC is sampling a pdf *for the parameters*.

Ideally, if you are using MCMC for inference, your code should input to the MCMC function or routine a probability function which is a product of a prior times a likelihood. As we noted above—in terms of implementation—it is usually advisable to work in the logarithm of the density function, so the function input to the MCMC code would be called something like `ln_f()`. This function `ln_f()` internally would compute and return the sum of a log prior pdf `ln_prior()` and a log likelihood function `ln_likelihood()`.

If you are using "flat" (improper) priors, the `ln_prior()` function can just return a zero no matter what the parameters. If it is flat with bounds (that is, proper), the `ln_prior()` should check the bounds and return `-Inf` values if the parameter vector is out of bounds. The pseudo-code for the `ln_f()` function should look something like this:

```
def ln_f(pars, data):
    x = ln_prior(pars)
    if not is_finite(x):
        return -Inf
    return x + ln_likelihood(data, pars)
```

This pseudo-code ensures that you only compute the likelihood when the parameters are within the prior bounds; it assumes implicitly that the prior pdf is easier to compute than the likelihood. If you find yourself in the opposite regime, adjust accordingly. Of course the above pseudo-code presumes that when you perform the accept–reject step of the MCMC method, the programming language handles properly the `-Inf` values.[†]

[†]If you are working in a language that doesn't have an `Inf` or doesn't evaluate compar-

MCMC *cannot sample a likelihood* (which is a probability *for the data* given parameters).[24] Despite this, in many cases, data analysts believe they are sampling the likelihood. This is because (we presume) they have put a likelihood function (or log-likelihood function) in as the input to the MCMC code, where the probability function (or log-probability function) should go. Then is it the likelihood that is being sampled? No, not really; it is a posterior probability that is directly proportional to the likelihood function. That is, it is a posterior probability for some implicit (and improper) "flat" priors.

It should be outside of the scope of this document to note here that it is a good idea to have proper priors.[25] Proper priors obey the integral constraint (9), with $Z$ finite. It isn't a requirement that priors be proper for the posterior to be proper, so many investigators and projects violate this rule. This is outside the current scope, except for the following point: It is a great functional test of your sampler and your data analysis setup to take the likelihood function to a very small power (much less than one) or multiply the log-likelihood by a very small number (much less than one) and check that the sampler samples, properly and correctly, the prior pdf. This test is only possible if the prior pdf is proper.

**Problem 7:** Run your M-H MCMC sampler from Problem 2, but now with a density function that is precisely unity *everywhere* (that is, at any input value of $x$ it returns unity). That is, an improper function (as discussed in Section 4). Run it for longer and longer and plot the chain value $x$ as a function of timestep. What happens?

**Problem 8:** For a real-world inference problem, read enough of Hogg *et al.* (2010a) to understand and execute Exercise 6 in that document.

**Problem 9:** Modify the sampler you wrote in Problem 2 to take steps not in $x$ but in $\ln x$. That is, replace the Gaussian proposal distribution $q(x' \,|\, x)$ with a Gaussian distribution in $\ln x$ $q(\ln x' \,|\, \ln x)$, but make no other changes. By doing this, you are no longer sampling the Gaussian $p(x)$ that you were in Problem 2. What about your answers change? What distribution are you

---

isons correctly when the `Inf` appears, you might have to write some case code, and have your `ln_f` function return a value *and* some kind of flag which indicates "zero probability" or $f(\theta) = 0$.

sampling now? Compute the analytic function that you have sampled from – this will no longer be the same $p(x)$ – and over-plot it on your histogram.

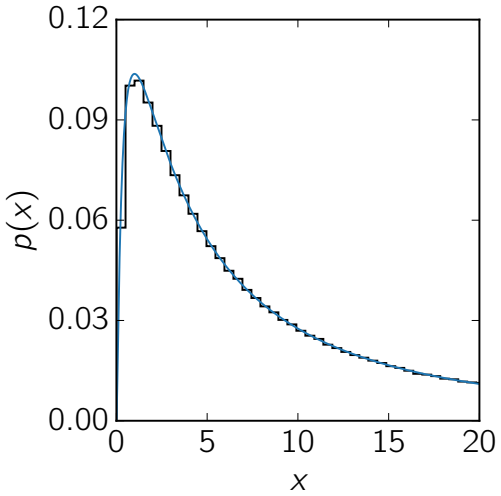


Figure 4.— Solution to Problem 9: The black histogram shows the MCMC samples and the blue curve is the analytic target density.

# 5 Convergence

A key question for an MCMC operator—*the* key question in some sense—is *how long to run* to be sure of having reliable results. It is disappointing and annoying to many that there is no extremely simple and reliable answer to this question.[26]

The reason that there is no simple answer is that you can't really ever know that you have sampled the full posterior pdf, and the reason for *that* is that if you *could* know that, you would also be able to solve the famously difficult discrete optimization problem.[27] Another way to put it is that you might have two modes—two separated regions of substantial total probability—in the posterior pdf that are both important but separated by a large region of low probability. If you are using a simple sampler, you could sample one of these modes very well but the sampler will take effectively infinite time to find the other mode. Any test of convergence would be passed, but in fact the sampling would be incomplete and not representative. That is, knowing that you have a complete and representative sampling is effectively impossible.

In this context, it is *simply not fair* to require an MCMC run to have fully and completely sampled the posterior pdf, at least not in any provable sense. The question of whether a sampling is converged to a representative sampling of the posterior pdf is actually outside the domain of science, because the domain of science is the domain of questions answerable by scientists. That is, you can *never know* that you have correctly sampled the posterior pdf, despite many statements in the literature to the contrary.[28] The upshot of this is that we don't and can't require any kind of absolute convergence; indeed, it would be impossible even to test for it.

In this pragmatic situation—the Real World, as it is called—we have to rely on heuristics. Heuristically, you have sampled long enough when you can see that the (or each) walker has traversed the high-probability parts of the parameter space many times in the length of the chain. Or, equivalently, you have sampled long enough when the first half of the chain shows very much the same posterior pdf morphology as the second half of the chain, or indeed as any substantial subset.

The above heuristics can be made more precise (in terms of the amount of deviation one expects between the means and variances, say, of two disjoint subsets of the chain. The premier tool for making this heuristic precise, however, is to look at the *autocorrelation time* of the chain. In general, when one has a sequence $(\theta_1, \theta_2, \theta_3, \cdots)$ generated by a Markov Process in the $\theta$ space, nearby points in the sequence will be similar, but sufficiently distant points will not "know about" one another; the autocorrelation function measures this.

The autocorrelation function is defined by

$$C_f(\Delta) \equiv \frac{E[(f(\theta_k) - \mu_f)\,(f(\theta_{k+\Delta}) - \mu_f)]}{E[(f(\theta) - \mu_f)^2]} \tag{14}$$

$$\mu_f = E[f(\theta)] \quad ; \tag{15}$$

where $f(\theta)$ is a scalar valued function of the parameters. In other words, equation (14) is the covariance of points in the chain separated by $\Delta$ steps, taken relative to the variance. In the above equation, the assumption is that the samples $\theta_k$ are given in the order that they were generated by the MCMC algorithm. By construction

$$C_f(0) = 1 \tag{16}$$

$$\lim_{\Delta\to\infty} C_f(\Delta) = 0 \quad , \tag{17}$$

and by definition, the autocorrelation "time" $\tau$ is

$$\tau_f = 1 + 2 \sum_{\Delta=1}^{\infty} C_f(\Delta) \quad . \tag{18}$$

Heuristically, the autocorrelation time is the number of samples you need to take before you get a new independent sample. More precisely, the autocorrelation time is the factor by which you have to reduce the number of samples when you compute the variance on a sampling-based expectation value estimate.

That is, if the expectation value $E[q]$ is computed by the sampling approximation given in equation (11), there is an uncertainty on that expectation given by

$$\begin{aligned} \mathrm{Var}[E_{p(\theta)}[q(\theta)]] &= \frac{\tau_q}{K} \mathrm{Var}_{p(\theta)}[q(\theta)] & (19) \\ &\approx \frac{\tau_q}{K^2} \sum_{k=1}^{K} \left(q(\theta_k) - E_{p(\theta)}[q(\theta)]\right)^2 \quad , & (20) \end{aligned}$$

where we are being a bit loose with terminology because this is the variance not on $E_{p(\theta)}[q(\theta)]$ but on our sampling-based *estimator* for $E_{p(\theta)}[q(\theta)]$, and the $E_{p(\theta)}[q(\theta)]$ inside the sum is that estimator too. It is slightly off-topic here, but related to this, if you have a long chain of length $K$ and with autocorrelation time $\tau$ and you thin the chain down to every $\tau$th sample (instead of using all $K$ samples), you will get just as good sampling-based estimates as if you had used all $K$ samples.

It is also worth noting that there is not just one autocorrelation function or autocorrelation time for an MCMC chain. Instead, there is an autocorrelation time $\tau_f$ for every function $f(\theta)$ and these will, in general, be different. Therefore, the autocorrelation time used in equation (20) is the time computed on the chain of $q(\theta)$ values instead of the autocorrelation time for $\theta$.

A sampler with a smaller autocorrelation time is just better; you have to do fewer $f(\theta)$ calls per independent sample, and you have to run less time to get accurate sampling-based integral estimates. A sampler that takes an independent sample every time would have an autocorrelation time of unity, which is the best possible value; this optimal sampling is only possible for problems where the sampling is analytic (for example if the posterior

is perfectly Gaussian with known mean and variance). In general, the best sampler will be different for different problems, and we will (below; Section 9) tune the samplers we have to do the best on the problems we have; this tuning will also be problem-specific. There are many heuristic bases on which different samplers might be compared or tuned, but fundamentally it is lower autocorrelation time that separates good samplers from bad ones and is the ultimate basis on which we compare performance.[29]

Of course, the autocorrelation-time definition (18) does not by itself provide an *estimator* for the time. We might first try the naïve estimator

$$\tau \stackrel{?}{=} 1 + 2\sum_{\Delta=1}^{N} C(\Delta) \tag{21}$$

where the sum is over the autocorrelation function estimated based the chain using equation (14). In practice, this is a high variance estimator of the integrated autocorrelation time and this variance doesn't go to zero for an infinitely long chain.[30] Instead, we can get a much more robust estimate of the autocorrelation time by restricting the sum in equation (21) to $\Delta \leq M$ where $\tau \leq M \ll N$

$$\tau = 1 + 2\sum_{\Delta=1}^{M} C(\Delta) \tag{22}$$

In practice, $M$ can be chosen iteratively to satisfy these constraints but it will only be as reliable as the chain itself. An estimate of $\tau$ using equation (22) should only be trusted if the chain is many times ($\sim 10$) longer than $\tau$.

One simple and sensible test of convergence is the Gelman–Rubin diagnostic [31], which compares the variance (in one parameter, or your most important parameter, or all parameters) *within* a chain to the variance *across* chains. This requires running multiple chains, and looking at the empirical variance of the parameter away from its mean within each individual chain, and comparing it to the variance in the mean of that parameter across chains, inflated to be a mean per-sample variance. What Gelman and Rubin do with these variances specifically is sensible, but the important question of convergence is whether, as the chains get longer, these two variances asymptotically reach stable values, and that those two values agree. The Gelman–Rubin diagnostic is related to the autocorrelation time, in that it will only deliver success when the chains are much longer than an autocorrelation time. It can

also be used to assess the accuracy of the chain for performing integrals—by replacing the mean of the parameter with whatever integral is of most interest.

Finally one last point about convergence: Since all convergence tests are fundamentally heuristic, it is useful to just make the heuristic visualization of the samples $\theta_k$ as a function of $k$, in the order they were generated. The chain is likely to be converged only if the random-walk process crossed the domain of $\theta$ fully many times during the MCMC run. Often some parameter directions are much worse than others; it is worth looking at the chain in all parameter directions.

**Problem 10:** Re-do Problem 2 but now look at convergence: Plot the $x$ chain as a function of timestep. Also split the chain into four contiguous segments (the first, second, third, and fourth quarters of the chain). In each of these four, compute the empirical mean and empirical variance of $x$. What do you conclude about convergence from these heuristics?

**Problem 11:** Write a piece of code that computes the empirical autocorrelation function. You will probably want to speed this computation up by using a fast Fourier transform[32]. Run this on the chain you obtained from Problem 2. Plot the autocorrelation function you find at short lags ($\Delta < 100$). This plot should resemble Figure 5.

**Problem 12:** Write a peice of code that estiamtes the integrated autocorrelation time for a chain of samples using an estimate of the autocorrelation function and a given "window" size $M$ (see equation 22). Plot the estimated $\tau$ as a function of $M$ for several contiguous segments of the chain and overplot the sample function based on the full chain. What can you conclude from this plot? Implement an iterative procedure for automatically choosing $M$.[33] Overplot this estimate on the plot of $\tau(M)$ and the result should look like Figure 6.

# 6 Tuning

Most MCMC methods make use of something like a "proposal distribution" which determines what kinds of steps the walker can take as it random-walks through the parameter space. The user generally has a lot of control over

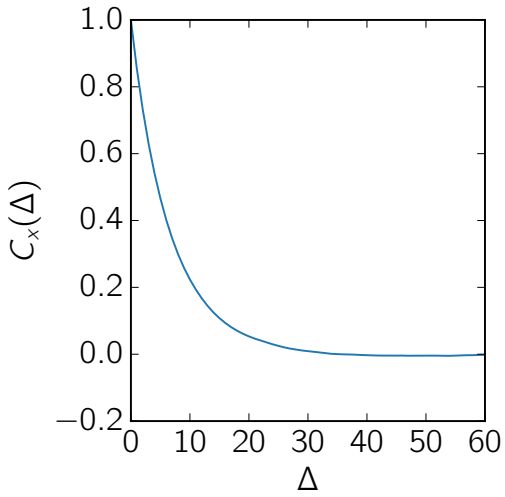


Figure 5.— Solution to Problem 11: The autocorrelation function of the chain from Problem 2.

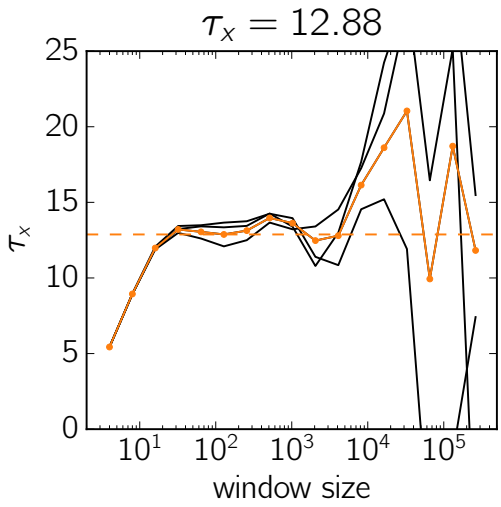


Figure 6.— Solution to Problem 12: Estimates of the integrated autocorrelation time of different segments of the MCMC chain from Problem 2 (black lines) and for the full chain (orange line) as a function of window size $M$. The "optimized" value computed using an iterative procedure is overplotted as a dashed line and its value is listed in the title.

what this proposal distribution might be, and how to choose the parameters. For example, in a $D$-dimensional parameter space, a zero-mean but anisotropic Gaussian (normal distribution) is often used to draw offsets for the walker to take from point to point. Even within this choice (which is of one function among many possibilities), there are $D\,(D+1)/2$ parameters in the $D \times D$ symmetric, positive definite covariance matrix to set "by hand". How to choose this function and set these parameters?

The key idea here is that if the proposal distribution is too narrow—it proposes steps too small—almost all steps will be accepted (REFERENCE ALGORITHM HERE), but it will take a long time to move anywhere because of timidity. If the proposal distribution is too wide—it proposes steps too large—the moves will cover parameter space easily, but almost no steps will be accepted; it will tend to jump to much lower probability regions. There is a Goldilocks step size (proposal distribution root-variance) that is "just right". In one dimension this might be easy to find, but, as we say, in large numbers of dimensions, there is a lot of freedom in choosing the parameters of the distribution.

In a deep sense, the only scalar that makes sense to optimize, when choosing proposal distribution (or, loosely speaking, step size), is the autocorrelation time, described above (Section 5). The optimal step size is the step size that makes for the shortest autocorrelation time. This is easy to state, but hard to use in practice; being a second-order statistic of the MCMC chain, the autocorrelation time is a hard thing to measure without a lot of data, so it is hard to quickly estimate it and adjust, much less put it into some kind of optimization loop. We usually use proxies for this of various kinds.

The simplest heuristic proxy statistic for tuning is the *acceptance fraction*. If you are accepting almost all proposed steps, your step sizes are too small on average. If you are accepting almost none, your step sizes are too large. The Goldilocks value is between a half and about a quarter, with an argument floating around that it should be 0.234 for best performance in high dimensional problems[34] (though you could never tune it precisely enough to warrant that third digit of accuracy). In the burn-in phase (discussed below in Section 7) of an MCMC run, it makes sense to track the acceptance ratio, and adjust the proposal distribution variance as you get acceptance ratios that are far from the Goldilocks ratio. This process can be automated easily; such automation is part of many projects that use MCMC.

A very common—and very useful—kind of proposal distribution is one that cycles through parameters, taking a random step in just one parameter

at a time.[35] Aside from involving some persistence of state, this kind of proposal distribution can be valuable, in part because it reduces the (almost impossible) $D$-dimensional tuning problem to $D$ one-dimensional problems: In this form of proposal distribution, it is possible to track a separate acceptance fraction for every parameter. Code can be built that uses the burn-in phase to tune all $D$ proposal variances such that each of them, individually, obtains acceptance at the same Goldilocks ratio.

One note to make here—because it is relevant to tuning—is that tuning can *only* take place during the burn-in phase (that is, some part of your chain you will discard later); you cannot tune *while* you run your final MCMC run. Why not? Because tuning the proposal distribution based on the past history of the chain violates the "Markov" property that each step depends *only* on the state at the previous step. Violating the Markov property can be very bad; when you violate it, you lose all the provable properties of MCMC on which all our righteous power is based.

Another proxy for autocorrelation time useful for tuning is the Expected Squared Jump Distance.[36] This is the mean squared distance the walker moves, per step. It is maximized when the acceptance ratio is reasonable and the step size is large; it is large when the exploration of the space is fast. This proxy is easy to measure and use for tuning, and more directly related to autocorrelation time than the acceptance ratio. We recommend using it for tuning, though we have never used it ourselves. Like the acceptance fraction, it can also be used to tune in the case that the proposal distribution loops over parameters; once again, the user gets (in this case) $D$ one-dimensional tunings.

When sampling really gets very slow, there are various tricks and tips to work through. The huge bag of possible tricks is so large, it goes way beyond the scope of this introductory *Chapter*. In our experience, it is valuable to make friends with at least one statistician, one applied mathematician, and one computer scientist. Between them, they ought to span the relevant literature.

**Problem 13:** Run the MCMC sampling of Problem 4 with the covariant Gaussian density. Give the proposal density $q(x' \,|\, x)$ a diagonal variance tensor that is $Q$ times the two-dimensional identity matrix. Assess the acceptance fraction as a function of $Q$. Find (very roughly) the value of $Q$ that gives an acceptance fraction of about 0.25. Don't try to optimize precisely;

just evaluate the acceptance fraction on a logarithmic grid of $Q$ with values of $Q$ separated by factors of 2.

**Problem 14:** Re-do Problem 13 but instead of trying to reach a certain acceptance fraction, try to minimize the autocorrelation time. You will need one of the autocorrelation-time estimators you might have built in a previous Problem. (This, by the way, is the Right Thing To Do, but often expensive.) What do you get as the best value of $Q$ in this case? Again, just evaluate on a coarse logarithmic grid.

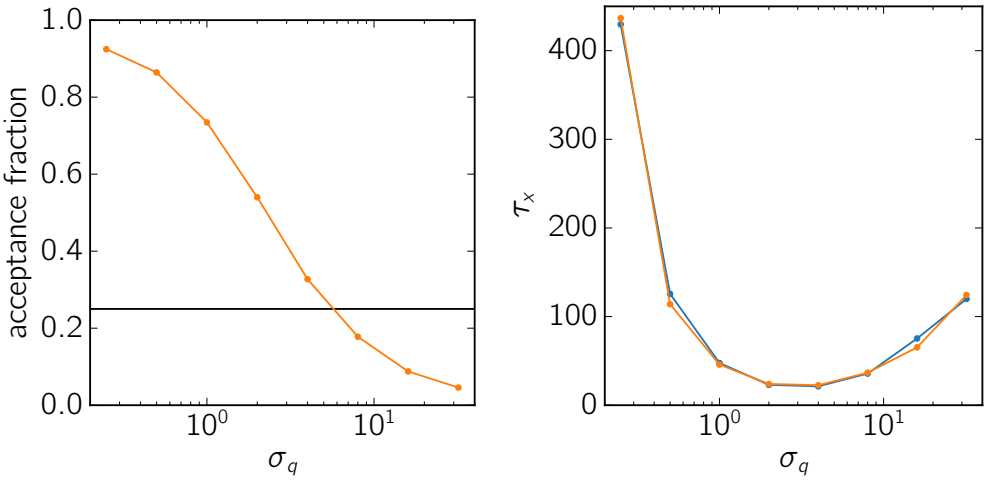


Figure 7.— Solutions to Problems 13 and 14: Two methods of tuning the M–H proposal parameters. *left:* The acceptance fraction as a function of proposal scale for the distribution from Problem 4a. *right:* The integrated autocorrelation time for each parameter (indicated by the different colors) as a function of proposal scale parameter.

**Problem 15:** In Problem 13 you varied only the parameter $Q$, but really there are three free parameters (two variances and a covariance). If the problem was $D$-dimensional, how many tuning parameters would there be, in principle?

**Problem 16:** The Rosenbrock density used as a demonstration case for many samplers (see, for example, Goodman & Weare 2010). Test your sam-

pler on this density:

$$p(\theta_1, \theta_2) \quad \propto \quad \exp\left(-\frac{100\,(\theta_2 - {\theta_1}^2)^2 + (1-\theta_1)^2}{20}\right) \quad . \tag{23}$$

Tune the Gaussian proposal distribution in your M–H MCMC sampler to sample this density efficiently. What autocorrelation time do you get? Compare to what *emcee* gets.

# 7 Initialization and burn-in

Just as most (though not all) MCMC methods require a choice of proposal distribution, most (though not all) require a choice about initialization: The walker needs (or walkers need) to be started somewhere in the parameter space. In many use cases for MCMC, the investigator wants to use MCMC to obtain uncertainty information about or propagate uncertainty into parameter estimates. In these cases, it makes sense to initialize the walker or walkers at sensible parameter estimates, found by optimizing a likelihood or a posterior pdf in advance of sampling. In extremely high dimensions (large numbers of parameters), this can be a bad idea, since the optimal parameters are not necessarily near typical posterior samples[37], but in low dimensions (few to tens) this is often sensible.

Ideally, you will initialize the walker not at a completely irrelevant point, nor at the optimum of the posterior pdf, but at a typical or pretty good place in the posterior pdf. That would minimize burn-in! But, in general, being at the optimum or anywhere near a good place is usually better than a mindless initialization.

As to burn-in: If you have started your sampler in a non-typical place—or if you are concerned that you *might* have started your sampler in a non-typical place—then you should discard the beginning of your MCMC run before you do your inferences. This discarded part is called the "burn-in". Some practitioners insist that burn-in cannot exist[38]: So long as the initial point is a conceivable sample, you are fine! Ensemble methods (like *emcee*[39]; and mentioned below in Section 9) require a burn-in phase and discard, because the whole point is that the ensemble must grow (or shrink) to fill the posterior pdf volume, and you can't initialize the ensemble sensibly if you don't know that volume *a priori*.

If you suspect, or if it is even possible, that your problem is badly multi-modal, then you will have to start at multiple points in parameter space and compare the resulting chains. By "badly multi-modal" we mean that there are peaks in the posterior pdf that are connected by low-enough valleys that it is very unlikely or takes a long time for a walker to traverse from one peak to the next. If this might be an issue, then the best diagnosis is to start many chains in parallel and check that they lead to identical (or very similar) posterior inferences (same pdf location and shape and statistics).

Of course if you find out the worst—if you find out that different initializations lead to different posterior inferences—you are in trouble: There is no trivial way to combine together the samplings you get of different modes! You can either wait a *very long time* so that you see a single walker in a single MCMC run traverse from mode to mode enough times to make a representative sampling (dozens or hundreds of times, ideally), or you can use a high-end sampler (like nested sampling; see Section 9) that is designed for such problems. In principle there are methods that involve splitting the space into two spaces, sampling them separately, and then combining the chains according to their relative evidence afterwards. That's a research project beyond the scope of this *Chapter*.

Of course there are some kinds of multi-modalities that might not be a problem. For example, you can have two solutions to a problem that differ only in the *labeling* of the components—say the Gaussians making up a mixture of Gaussians—that make up the model. One solution has the $K$ Gaussian components in one order, and the other in another order, but they are identical Gaussians in all other respects. Because these two solutions make the same predictions for any data, and only differ in their irrelevant, latent, internal "naming" of components, they are not really different solutions. This suggests that when you ask whether the outputs of two MCMC chains are consistent with one another, you should do so in the realm of the parameters you *care about*, not irrelevant parameters that do not have impact on any present or future data.

But it is worth remembering that just as MCMC is not a good optimizer (generically, samples will not lie close to the maximum of the posterior pdf), it is also not a good *search algorithm*. There is no sense in which (standard) MCMC methods are efficient at searching, or engineered to search, all of parameter space. If you really need to check all of parameter space, you should put MCMC aside, and do an exhaustive search.[40]

**Problem 17:** Re-do Problem 2 but with different starting positions. What happens as you make the starting position extremely far from the origin? What is the scaling: As you move the initialization further away, how much longer does it take for the sampler to reach reasonable density?

**Problem 18:** Check the scaling you found in Problem 17 with a higher-dimensional Gaussian (try, for example, a 10-d Gaussian). The same or worse?

**Problem 19:** Import (or write) an optimizer, and upgrade the code you wrote for Problem 17 to begin by optimizing $\ln p(x)$ and only then start the MCMC sampler from that optimum. Use a sensible optimizer. Compare the scaling you found in Problem 17 to the same scaling for the optimizer. To make this test fair, don't use the awesome math you know about Gaussians to help you here; pretend that $p(x)$ is an unknown function with unknown derivatives.

# 8 Results, error bars, and figures

For a committed probabilistic scientist—frequentist or Bayesian—there is no "answer", there are only probability density functions (pdfs). Any time in a paper or in an email or in a conversation we say what "the answer is"—even if we say it with an error bar—we have departed the probabilistic program. There are many principled ways to depart the program; in another *Chapter*, we discuss the economic explanation of how we can make hard decisions in the context of probabilistic reasoning[41] But without going into that economic model, it is fair to say that a substantial problem with being a committed probabilist is that when we *publish*, we are not permitted (not now, at least) to publish a *probability distribution over publications*; we have to publish *one single, deterministic text*; we have to make a decision about what to write in the title, the abstract, the tables, figures, and results section. Given a MCMC sampling of the posterior pdf, what do we report as our *results*?

One thing we can keep in mind to guide us in this is what we said above (Section 2), which is that samplings are good for doing *integrals*. We should endeavor to use as our "results" outputs from the MCMC that are based on integrals computed with the sampling. This includes expectations, medians, quantiles, one-dimensional histograms, and multi-dimensional histograms.

This does *not* include the "best" sample or a mode or optimum. The latter things are not necessarily illegitimate outputs of the MCMC, but they do not make best use of the fact that the sampling is a tool for integration, and we do not recommend them.

Imagine that you are in the simplest case: You have a model with a small number of parameters (say three-ish), and only one of them is of great interest. What are your options for the "measurement" of this parameter? The only simple integral-based options are the posterior mean or posterior median value for the parameter.

For example, imagine that by MCMC you have generated $K$ samples $\theta_k$ of a parameter vector $\theta$. Now you have a scalar function $q(\theta)$ which takes the parameter vector and returns a scalar value[42] which could be as simple as a single component of the parameter vector (one parameter from the list), or something more complex. The mean-of-sampling value $\langle q \rangle$ is just

$$\langle q \rangle \leftarrow \frac{1}{K} \sum_{k=1}^{K} q(\theta_k) \quad , \tag{24}$$

and the median-of-sampling value is just the $[K/2]$th value of $q(\theta_k)$ when the $q(\theta_k)$ have been ordered (sorted). These are both produced by integrals; the first is the value returned by the expectation integral estimate, the second is the value past which half of the integrated pdf lies.

What are your best options for the "uncertainty" or "limits" on this parameter? The only simple integral-based options are either variances or else posterior quantiles. We usually use quantiles. That is, the "one-sigma" error bar can be taken to be the half-size of the central (or smallest[43]) interval of the parameter that contains 68 percent of the posterior samples. The "two-sigma" error bar would be the same but for the 95-percent interval. Again, these limits can be estimated[44] by ordering the $q(\theta_k)$ values, and finding the $\theta$ values such that some fraction (0.68 or 0.95) of the samples lie between them. Again, to form these limits, $q(\theta)$ must be a scalar function (so sorting is well defined). Because they are so afraid of being confused with frequentists, Bayesians often call these regions "credible" rather than "confidence"[45]

One amusing thing is that if you choose the 68-percent interval sensibly, but use as the "measurement" the posterior mean, you can get pathological situations in which the posterior mean measurement is actually *outside* the one-sigma confidence interval. For this reason (and others), we usually recommend as a default behavior—in the one-dimensional case—to choose

the *median* of sampling as the measurement value, the 16-percent quantile as the lower one-sigma error bar, and the 84-percent quantile as the upper one-sigma error bar. This has pathologies in higher dimensions (as we are about to see), but is pretty safe for one-dimensional answers.

The conservative scientist would show not just 68-percent error bars, but also 95-percent (to help readers visualize the skew of the posterior pdf). The very conservative scientist would also show a histogram of the posterior samples, with the relevant quantiles (2.5, 16, 50, 84, and 97.5 percent) indicated.

As we mentioned above, scientists often like to report the "best sample"; that is, the sample with the highest posterior pdf value. This is not usually a good idea. For one, MCMC is a sampling algorithm, not an optimization algorithm; there are no guarantees that it will find the optimum of the posterior pdf in reasonable time.[46] For two, the closeness of the best sample to the posterior mode is a strong function of sample size, and with very bad scaling.[47] For three, if you just want an optimum, use an optimizer! That is, we *strongly advise against* reporting, as "the" measurement or "the result", the parameter value or values for the best sample. Giving some posterior samples is a good idea (we return to this below).

If you feel drawn to give the best sample, instead give the parameter values found by optimizing the posterior pdf, starting at the best sample as an initialization (and sacrifice your probabilism[48]). This optimal-posterior parameter value is called the "maximum *a posteriori*" or "MAP" value for the parameters; it is like a maximum-likelihood value but regularized by the prior. It is an estimator with some good (and some bad) properties, but it is not the point of MCMC, and therefore outside the scope of this *Chapter*.

The parameter space $\theta$ is usually multi-dimensional, and usually it is more than one of those parameters that is of interest. In this case, the mean or median of sampling can be produced for each dimension. Because the sampling projected onto any dimension is a marginalization of the posterior pdf, any such one-dimensional mean or median is the mean or median of the marginalized posterior pdf.

There is one oddity to note here, as it catches many investigators by surprise: Even if the model is good and the posterior pdf is unimodal and well-behaved, the median (or even mean) of the posterior pdf for each of parameters, when taken together as a "best-fit" parameter vector $\theta^*$, will not itself necessarily be a good fit to the data! If there is substantial "curvature" to the pdf in the parameter space, the mean or median of sampling does not necessarily lie at a high-probability location of parameter space. This may

seem counterintuitive, but it is easy to see in the case of a "banana-shaped" posterior pdf. This all relates to the fact that MCMC is not an optimizer, it is a sampler. It is important, when reporting the output of an MCMC run by giving means or medians of the posterior pdf to remind the reader or user of that output that it does not necessarily represent (collectively) a good fit to the data; these outputs only give information that is useful one parameter at a time.

In principle the only output from the sampling that safely gives both probabilistic information about the result of the inference and also good-fitting models is a few—randomly chosen—example samples. We strongly recommend this[49]; in particular this is a better thing to do than to give the "best sample", which isn't even guaranteed to be near the bulk of the posterior pdf or the samples therefrom.

If you *do* return posterior samples to your audience, how many should you return? There are only heuristic answers to this, but a guideline is that, if you only care about a single (one-dimensional) parameter, you don't need many samples; a dozen suffices to give you a reasonable posterior pdf mean and variance.[50] If you care about $K$ parameters or dimensions of the parameter space, in general the need for samples grows exponentially (or perhaps even factorially!) with the number of dimensions. So if you have a ten-dimensional space, expect to be publishing your samples in an electronic table on the web!

If you *do* publish sufficient numbers of posterior samples, you might find users who want to make serious use of them.[51]

Of course figures are always better than tables, at least in the printed form of scientific publications. In a $D$-dimensional parameter space, we recommend plotting all $D$ one-dimensional sample histograms, on a pretty fine binning, and all $D$-choose-2 two-dimensional histograms (scatter plots) to show the low-level covariances and non-linearities. If you are clever, all these plots can be arranged into a lovely and informative triangle.[52] These two-dimensional plots do not in any sense contain all the information in the sampling! However, they are remarkable for locating expected and unexpected parameter relationships, and often invaluable for suggesting re-parameterizations and transformations that simplify your problem.

**Problem 20:** Execute Problem 16, or—if you are lazy—just install and use *emcee* to do the hard work. Now plot the $x$ and $y$ histograms of a 10,000-

point sampling of this distribution (you might have to sample more than 10,000 and thin the chain), and also plot the two-dimensional scatter plot of $x, y$ samples. Overplot on all three plots an indicator of the means and medians of the samples along the $x$ and $y$ directions. Overplot on all three plots the (above) recommended quantiles of the samples. Comment on the results.

# 9 Gibbs, ensemble, Hamiltonian, and so on

As we have discussed above (Section 3), MCMC methods are proven to be correct "in the limit"; that is, when an infinite number of samples have been taken. When the function $f(\theta)$ has challenging properties, the results can approach this limit very slowly (or not at all[53]). For this reason, in the finite duration of a scientific project, most users of MCMC have one of two (related) problems: Either *(1)* it is taking too long—it is taking too many steps or executing too many calls of the function $f(\theta)$ to get independent samples—or *(2)* it is not exploring the full parameter space—there are local optima (or bad local geometry around those optima) "trapping" the algorithm.

There is no general, problem-independent solution for either of these problems. Indeed, there are many different strategies that work well or badly for different functions $f(\theta)$. In this Section, we will try to guide the reader towards methods that might help in different circumstances. We will not give a full description of all the MCMC algorithms out there, but just provide some field notes and pointers to references. New MCMC methods are being developed all the time, so a reader with a serious problem would do well to consult with the applied-mathematics or statistics literature.[54] The different methods to which we refer here are (in general) very different in their difficulty of implementation. We won't help with that either, except to say that there are more and more open-source or distributed packages every year. We will try to name some current implementations in the notes, but any list compiled today will be incomplete tomorrow.[55] The spaces of MCMC methods and MCMC method implementations are both growing rapidly, in part because MCMC has become such a core technology in the empirical sciences.

All we are going to do here is give a cursory mention to some methods we know about, with some keywords that can be used to search more deeply.

**Ensembles:** As we discussed (Section 6), one of the critical challenges in the use of MCMC is tuning the proposal pdf $q(\theta' \,|\, \theta)$. In problems in which all of the parameters (all of the components or entries of $\theta$) are somehow "equivalent", or the parameters can be seen as components of a vector in a vector space, there are *ensemble methods* that make use of not just one random-walker but instead many walkers to automatically generate a properly tuned proposal distribution. These methods make use of the distribution of a set of independent walkers in the parameter space ($\theta$-space) to gauge the typical step-sizes and directions at which new proposals should be made, obviating or greatly reducing the tuning requirements. We are co-authors on one of these methods, *emcee*[56], which is popular in the astrophysical sciences, but there are others[57]. It is worty of note that ensemble methods have particular peculiarities about initialization and convergence diagnostics that are worth understanding before use.[58] Because ensemble methods are updating independent MCMC chains, they are often easy to parallelize to make use of multiple cores.

**Gibbs:** In many problems, the parameters—the components or entries of the blob $\theta$—are not equivalent at all. Some are global parameters, which touch or have an effect on every sub-part of the data blob $D$, while others are local parameteres, which only touch a small subset of the data. Or, in another use case, some parameters are linear parameters that—at fixed values of the other parameters—will have a Gaussian likelihood or posterior pdf. In these problems, there are some directions in parameter space that are hard to move in (they require complete re-calculation of the full non-linear function $f(\theta)$, and other directions that are very easy to move in (they require only partial re-calculations or the sampling could even be analytic and exact). For problems with these structures, Gibbs samplers are ideal, and there are good implementations.[59]

A trivial kind of Gibbs sampling was mentioned above (Section 3) in the discussion of M–H MCMC: Even when running vanilla M–H MCMC, it is often useful to update only one parameter at a time; that is, do only axis-aligned moves in the $\theta$-space, cycling through axes as you go from step to step. This helps with tuning and diagnosis, but it also permits a good implementation to capitalize on simpler calculations for the simpler parameter directions.[60]

Gibbs sampling can be very good for exploiting multiprocessing: Global parameter updates might require a serial re-calculation of the full likelihood,

but local parameter updates can be done in parallel, with each local parameter working on its local bit of data all at the same time. This kind of structure is common in hierarchical inference, where local parameters touch only individual objects in some population, say, and global parameters are parameters *of* that population as a whole. Parallelization is built in to some Gibbs implementations.[61]

**Hamiltonian or Hybrid:** In many cases of interest, the function $f(\theta)$ can be analytically differentiated, such that it is possible to quickly execute a gradient $\mathrm{d}f/\mathrm{d}\theta$ evaluation. Indeed, in the modern world of computing, there are even auto-differentiation systems that deliver automatically code that produces the gradient of any function you can write.[62] There is a class of MCMC methods that make use of gradient information to effectively inform the proposal distribution and thus speed sampling.

DFM: more here! For example, does this work better in large numbers of dimensions? If so, we should say so. And cite (Neal *et al.*, 2011).

**Importance:** Sometimes it is possible to sample efficiently from a distribution that is close to the target function $f(\theta)$ you care about, even when the target $f(\theta)$ itself is hard to sample efficiently with MCMC. An example is a problem that has a simple (Gaussian, say) likelihood function times a non-trivial prior: It is easy to sample from a Gaussian likelihood times a Gaussian prior, even without MCMC, but it might be hard to do the same sampling under a more complex prior. Importance sampling is a direct sampling method (not really an MCMC method at all): You take a sampling from the approximate function (which by assumption was easy) and then you re-weight the sampling (or do a subsequent rejection of samples) using the ratio of the true function to the approximate function as a weight or probability. We use this quite a bit in our hierarchical inferences.[63]

Two of the significant technical details for importance sampling are the following: The approximate function must have the same (or greater) support as the true function, and the method degrades in efficiency as the two functions diverge. If the ratio of the true function to the approximate function is to be used as a probability for rejection of samples, then the true function must be greater than or equal to the approximate function everywhere!

**Slice:** DFM ... When is slice sampling a good idea? DFM, can you help here?

Slice sampling makes use of the point that (in one dimension) a fair sampling of $f(\theta)$ is a projection onto the $\theta$ axis of a uniform sampling of the two-dimensional region under the two-dimensional plot of $f(\theta)$ *vs.* $\theta$. The $D$-dimensional generalization is that a fair sampling of $f(\theta)$ is a projection onto the $\theta$ $D$-dimensional hypersurface of $\theta$ of a uniform sampling of the $D+1$-dimensional hypervolume under the hypersurface $f(\theta)$ in a hypothetical plot of $f(\theta)$ over the whole parameter space. Thinking in terms of the two-dimensional case (for simplicity), slice samplers make both "horizontal" moves in the $\theta$ direction at fixed $f(\theta)$ and also "vertical" moves in the $f(\theta)$ direction. Because the sampling in the higher-dimensional space is uniform, it is easy to make moves to very different parts of parameter space, or different peaks or modes in the function $f(\theta)$. In our own research, we use a variant of slice sampling for problems involving samplings governed by Gaussian Processes.[64]

**Tempering:** A huge problem for MCMC samplers—which make local moves based on local positioning in the parameter space—is that there can be multiple modes in the function $f(\theta)$, separated by valleys of zero or near-zero density. A vanilla MCMC method cannot easily cross such valleys with local moves. The idea behind simulated tempering and related methods is to "smooth out" the function $f(\theta)$ or reduce its dynamic range, to make the peaks less tall and the valleys less deep, to permit the MCMC random walk to cross the valleys. The standard method is to introduce a "temperature" variable and, while sampling, take the likelihood function to the power of the inverse temperature (or, if working in the logarithm, multiply the log likelihood by the inverse temperature). When the temperature is high, this reduces the influence of the likelihood function relative to the prior pdf, and (assuming that the prior pdf is easy to sample) makes movement in the parameter space easier. At each step, when the temperature has moved away from unity, the distribution being sampled is not exactly $f(\theta)$, but the results of the sampling are only kept for those samples taken at unit temperature; these in the end make a fair sampling.

**Nested:** Similar to tempering-like methods are a class of methods called "nested" samplers[65], which also smoothly change the target distribution

away from $f(\theta)$. In the nested case, instead of increasing the temperature, the samplings are of a censored version of the prior, censored by the value of the likelihoood function. When the likelihood censoring is strong, only the most high-likelihood parts of the prior get sampled; when the likelihood censoring is weak, almost all of the prior gets sampled. The idea behind nested sampling—like tempering—is that it is designed to be a good method for exploring the full parameter space or searching for all of the modes of the posterior pdf. It is also designed such that it produces not just a sampling but also an estimate of the integral $Z$ of the likelihood times the prior (the Bayes factor or fully marginalized likelihood).

**Data augmentation:** ...HOGG Look at the relevant chapter in Brooks.

**Reversible-jump:** Finally, there are some extreme problems in which it is not just hard to sample in the parameter space, but the parameter space itself is not of fixed dimension.

For example, if you are modeling an astronomical image with a collection of stars, and you don't know how many stars to use; in this case the number of stars itself is a parameter, so the number of parameters is itself a function of the parameters.[66] In these cases you can only use certain kinds of samplers (though M–H MCMC is one you can choose), and, in addition, you need to make special kinds of proposals that permit the system to jump from one parameter space to another that has different dimensionality. In these cases, the concept of detailed balance becomes non-trivial, but there are criteria for creating reversible-jump proposals between the parameter spaces. These methods are best built and operated under the supervision of a trained professional.

# 10 Troubleshooting and advice

When issues arise with MCMC sampling, they are sometimes difficult to diagnose: Is it the MCMC code, the priors, the likelihood function, the initialization, the tuning, or something else?

**functional testing:** In addition to a full set of unit tests for every part of your code (yeah *right*[67]), we recommend building some end-to-end functional tests that permit you to perform MCMC runs with output that must

meet certain expectations. The simplest kind of functional test is to sample a known distribution, and check that the sampling has the moments you expect (to within tolerances). For example, you can sample a $D$-dimensional Gaussian distribution with known non-trivial covariance, and check that the empirical covariance comes out as expected. This is an easy test to fail, so a success builds confidence in any MCMC code.

In the most extreme form of this test, for a Gaussian in $D$ dimensions, build the covariance matrix input to the Gaussian distribution by taking the sum of the outer product of $D + 1$ randomly generated vectors. Then, before sampling, get the eigenvalues and eigenvectors of the matrix. Set up the MCMC sampler with the Gaussian function with this covariance matrix as the probability function (or the log of it as the log-probability function). Sample for many autocorrelation times, or until you are confident you are getting good samples. Check that the empirical $D$-dimensional covariance of the samples has the same (or similar) eigenvectors and eigenvlues to the input matrix. Also check that the individual one-dimensional samplings (along each axis or any projection in the $D$-space) look Gaussian and have nearly vanishing skew and kurtosis. It takes some thinking and some calculating to see "how well" these things all ought to match expectations, but one thing that can be said with confidence is that agreement should improve as the sampling proceeds. If it doesn't, it is likely that there is something broken about the sampler itself.

Another important functional test is to *sample the prior pdf*. This tests the sampler, tests that your priors really are what you think they are (and there are so many ways for them *not* to be[68]), and tests that your prior pdfs are proper (functions that can be normalized). This test is most effective as a true end-to-end test of your model if you structure your code such that the log probability function `lnprob` input to the MCMC scheme is set up as a sum of a log likelihood function `lnlikelihood` and a prior function `lnprior`. The sampler can be switched to sampling the prior by a simple replacement of the `lnlikelihood` function with a trivial function that always returns zero.

MCMC should run well in this flat-likelihood case (priors are usually easy to sample) and the output sample histograms should look like what you expect given the prior pdfs you coded. If the walkers run off to positive or negative infinity, and nothing seems to converge, your priors are probably not proper. If the priors don't look as you expect, you either have a bug in your code, or else a think-o about how your priors ought to look. Either way,

diagnosis in order.

**visualization:** HOGG: What plots do you make to diagnose problems?

HOGG: Plot parameters vs "time" in the chain.

HOGG: Plot a cut through the density function and the samples near that cut.

HOGG:...Make triangle plots. Great value!... Show example here... Mentioned above; drop here or reiterate?

**likelihood issues:** Sometimes there can be problems with the likelihood function itself. For example, when you re-call the likelihood function with the same parameters, do you get *exactly* the same answer? If you don't, your likelihood function effectively depends on some random numbers; maybe it includes within it an integral performed by Monte Carlo method? In general, you want to change your likelihood function such that, even if it includes a numerical or stochastic integral, it is designed such that it produces *precisely* the same value when it returns to the same point in parameter space. One way to make this happen is to choose the random numbers (those used for any internal integration inside the likelihood function) in advance and re-use them on every likelihood call. This provides the same integral according to the same approximation, but both speeds up the code (no re-generation of random numbers) and also makes the likelihood function single valued. Even better is to replace internal Monte Carlo integrals with deterministic numerical (or even better analytic) integrals.

If things are giving trouble and you suspect the likelihood function, another important test is to visualize slices through parameter space. When you call the likelihood function on a grid in each parameter (with the others fixed, say, to reasonable values), you should see a likelihood value that is a smooth function of each parameter, or at least as smooth as you expect. Things like numerical integration, truncated expansions, or adaptive approximations inside likelihood functions can make the function noisy or jagged at small scales in parameter space. These problems hurt optimizers more than samplers, but they usually point to code issues. In general, plotting the likelihood function as a function of parameter along slices in parameter space often reveals bugs or think-os.

DFM:...SHOW EXAMPLES?

**low acceptance fraction:** If you find that you are getting low acceptance fraction (in standard M-H MCMC or other varieties for which there is such a concept) you can simply reduce the sizes of the steps you consider to increase the fraction that are accepted. This means decreasing the variance of the proposal distribution, or whatever parameter or parameters control the width of that distribution. If reducing the variance of the proposal distribution does *not* increase the acceptance fraction, then *you have a bug*. You must find that bug by auditing your code or else returning to the functional tests listed above.

Similarly for high acceptance fraction: If you are finding a high acceptance fraction, then increasing the variance of the proposal distributon *must* decrease the acceptance fraction. If it does *not*, then *you have a bug*.

**sticky chains:** If you are concerned about acceptance fraction or convergence, it makes sense to plot parameter values as a function of "step number" or iteration number. That is, plot the ordered chain in each parameter dimension. These plots will have long horizontal patches if the chain is getting stuck; a converged chain will show the walker traversing the parameter space in every dimension fully many times over the length of the run. Not a few times; many times. If you are using an ensemble sampler[69] make plots showing all $M$ walkers, each a different color. A converged run for an ensemble sampler will show every single walker traversing the parameter space in every dimension many times.

DFM: SHOW EXAMPLES OF GOOD AND BAD?

**initialization-dependence:** If you initialize your MCMC runs in different places in parameter space, do you get the same (or very similar) final parameter samplings, in mean and variance? If you don't, then your posterior pdf is badly multi-modal; trivial MCMC methods probably won't fix your problem. Not to put too fine a point on it: If your results are initialization-dependent, then your MCMC runs *are not converged*.

You either have to go to a method that tries to explore the posterior more liberally, such as nested sampling or simulated tempering, or else split your "model" up into several sub-models, each of which contains different posterior modes. In general, this problem is hard to fix; as we mentioned above, it is provably impossible to explore all of parameter space if the posterior pdf is complex in morphology. We don't have very useful advice here, except

to spend time learning about the multiple modes in the posterior pdf and what each "means" or corresponds to, and to do your best to express in your results the multiple optima. One of the miracles of MCMC is that if you *can* fairly sample as complex posterior pdf, the fraction of samples in each mode tends to the relative total integrated posterior probability inside each mode, as the sampling converges.

Ensemble samplers react to multiple modes by obtaining very low acceptance ratio, because the samplers in the different modes are not easily able to "help one another" move efficiently. Again, if the acceptance ratio is low and there appear to be multiple modes, it is likely that the sampling will also be initialization dependent, and it is very unlikely that your sampling is going to be converged.

**bad initialization:** If you initialize your sampler in a disallowed region of parameter space—that is, a part of the space with $f(\theta) = 0$ (so your `ln_f()` function returns `-Inf`)—it might have a very hard time random-walking out of that location. At initialization time, it is important to test that the initialization is permitted. For ensemble methods, it is important to test that *all* the walkers in the ensemble are initialized to permitted values.

**parameterization problems:** It helps to think carefully about how to parameterize. For example, if there is an angle $\phi$ in your problem, and your prior requires it to be between 0 and $2\pi$, and the posterior mode is very near 0, a little bit of probability leaking below 0 plus the prior cutoffs plus mod-$2\pi$ symmetry makes an intrinsically unimodal problem multimodal! In these cases, we usually advise reparameterization from a (say) amplitude $A$ and angle $\phi$ to two vector components $a \equiv A\cos\phi$ and $b \equiv A\sin\phi$. Of course this re-parameterization changes significantly the *form* of the prior pdf (the transformation brings in a Jacobian); it has to be adjusted with care.

DFM: SHOULD WE DO THIS ALL EXPLICITLY HERE IN MATH? Choose a $p(A)$ and a $p(\phi) = [2\pi]^{-1}$, and then transform to $p(a, b)$.

For another example, sometimes there is almost an exact degeneracy between two parameters, say $a$ and $b$. Then it makes sense to switch to parameters $a + b$ and $a - b$, or some other linear combinations where correlations or near-degeneracies are likely to be reduced. Again, such transformations require also prior transformations, which must be made with care. There are some affine invariant samplers[70] that are invariant to such transformations,

but if you aren't using such a sampler, it is worth getting out ahead of these problems. They can be found by performing an exploratory sampling, making a triangle plot as described above, and looking for narrow diagonal lines in the two-dimentional scatter plots.

HOGG: ...How to deal with discrete or integer or binary parameters?

**model checking:** *All models are wrong, but some are useful.* That's not a quote from Box, but it's close.[71] The point (for our purposes) is that if you check your model hard enough—that is, you take enough data—you will certainly rule it out.[72] "Model checking" is an enormous subject that goes way beyond the scope of this *Chapter*, for which reason we won't say much about it here except to make two points:

The first—and most important—point about model checking is that you always learn a huge amount by checking the model. What this entails is looking at the distribution of residuals or the quality of the predictions of the model when compared to the data. There are methods (like the chi-squared statistic and the Bayesian evidence integral) that check the absolute quality of the model in a scalar way. These are usually uninformative, because (as we noted), all models are wrong, and eventually the data will be good enough to return a bad statistic here.[73]

Much more informative are methods (like plotting residuals of the data away from the model in the space of the data) that look at what parts of the data the model "explains well" and what parts of the data the model "explains badly". These kinds of experiments usually reveal inadequacies of the model in a rich way, or confirm that certain assumptions are bad. Cross-validation[74] is a framework for constructing valuable tests of this form, as is posterior predictive checks[75] Importantly, you want to find ways to compare your model to your data *in the space of the data*. Don't just look at the parameter space of your model! It is the data that really exist, especially when all models are wrong!

There are model checking methods (like varying the prior and re-running MCMC) that test the sensitivity of results to assumptions. Not only are these very valuable for testing the model, they are essential for writing any paper about your results: The reader always wants to know the sensitivity of results to assumptions.[76] Once again, it is important to find ways to display the results of such tests in the space of the data. All we will say here about these kinds of rich model tests is that you should do them, and report the

results.

The second—and less important—point about model checking is that the ur-Bayesian thing to be doing is computing a Bayesian evidence integral. However, it is usually inadvisable: The integral is expensive to compute. Most MCMC methods are cleverly designed to *avoid* computing this integral (as we note above in Section 1); so it usually adds expense to the project. At the end of the day, it returns only a single, scalar number, with no guidance about how to use it (and indeed there are only heuristics to reach for). If your model is wrong, the richer tests of visualization of residuals or sensitivity to assumptions are much more likely to lead to insight about what changes to make or what new directions to explore.

# Notes

[1] 

[2]We would like to thank Jo Bovy (IAS), Brendon Brewer (Auckland), Will Farr (Birmingham), Jonathan Goodman (NYU), Fengji Hou (NYU), Dustin Lang (CMU), Phil Marshall (KIPAC), and Iain Murray (Edinburgh) for valuable advice and comments over the years, and Alessandro Gentilini for a very close read. This project benefitted from various research grants, especially NSF grants AST-0908357, IIS-1124794, and AST-1517237, NASA grant NNX12AI50G, and the The Moore-Sloan Data Science Environment at NYU.

[3]Some would call the factor $Z$ in equation (1) something like "$p(D)$" or "the fully marginalized likelihood" or the probability of the data. Of course it is only the probability of the data within the model space under consideration! We (intend to) discuss this object and these issues elsewhere (in various things in preparation). The factor $Z$ is often hugely difficult to compute, because the likelihood (or the prior) can have extremely complex structure, with multiple arbitrarily compact modes, arbitrarily positioned in the (presumably high dimensional) parameter space $\theta$. Elsewhere, we discuss the computation of this object (Hou *et al.* 2014), and so have many others before us. Perhaps more importantly, we also have views about why you probably never really *want* to compute it. These views relate to the fact that the value of this integral very rarely delivers the answer to a question you care about (see future *Chapter*s in this series). These philosophical matters and the computational issues are both outside the scope of this *Chapter*, but suffice it to say: If the computation of $Z$ is in your travel plans, you will need to think about implementing some non-trivial integrating or sampling technology; we will give some brief description of options in Section 9.

[4]In general in this *Chapter* we will treat any variable as being capable of containing not just a value but some huge arbitrarily structured set of values in matrix, vector, or an ordered list of any set of such objects, or perhaps some even more complex form.

[5]For more discussion of our definition of a *model*, see [some other as-yet-unwritten *Chapter*] in this series. Briefly, a "model" for us is a likelihood

function (a pdf for the data given model parameters), and a prior pdf over the parameters. Because, under this definition, the model can always generate (by sampling, say) parameters and parameters can generate (again by sampling, say) data, the model is effectively (or actually, if you are a true subjective Bayesian) a probability distribution (pdf) over all possible data.

[6]Again, the parameter blob $\theta$ should also be thought of as an immense, arbitrarily structured set or list of parameters, possibly even with no fixed dimension!

[7]We will bash the terminology "likelihood" in another, as-yet-unwritten, *Chapter* in this series. Techically, this thing $p(D \mid \theta)$ is only properly a likelihood function when we are thinking of the data $D$ as being fixed, and the parameters $\theta$ as being permitted to vary.

[8]Every *Chapter* in the *Data Analysis Recipes* series begins with a rant in which we argue that most uses of the methods in question are not appropriate!

[9]The assumptions include things like: The algorithm is run "long enough", where this phrase is undefined (since the convergence requirements are set by precision requirements on particular integrals), and that the density that is being sampled has some connectedness properties: There aren't distant islands of finite density separated by regions of zero (or exceedingly low) density.

[10]Of course a committed Bayesian would argue that any time you are optimizing a posterior pdf or optimizing a likelihood, you *should* be sampling a posterior pdf. That is, for some, the fact that when someone "wants" to optimize, it is actually useful that they choose the "wrong" tool, because that "wrong" tool gives them back something far more useful than the output of any optimizer!

[11]Seriously. It has something to do with measure theory.

[12]As we will see, the intractability comes from various directions, but one is that the dimension of $\theta$ gets large in most realistic situations. Another is that the support of $p(\theta)$ tends to be, in normal inference situations, *far smaller* than the full domain of $\theta$. That is, the pdf is at or very close to zero over all but some tiny and very hard-to-find part of the space.

[13]We are referring here to the point that if you want a histogram of samples

to look just like the posterior pdf from which the samples are drawn, the histogram (thought of as a step function) must integrate to unity.

[14]This point about marginalization is, once again, a use of the sampling to perform an *integral*; the marginalized pdf is obtained from the full pdf by an integration. The sampling performs this integration automatically.

[15]We will also sometimes refer to expectations under $f(\theta)$ as posterior means, and medians as median-of-posterior values, and so on.

[16]There isn't a full citation for this package but there is van der Walt *et al.* (2011).

[17]There are many claims that this algorithm is incorrectly named, with claims that it should be credited to Enrico Fermi or Stan Ulam. We don't have any opinions of this matter, but encourage the reader to follow this up. The original paper is Metropolis *et al.* (1953), and there are a few sketchy historical notes in Geyer (2011).

[18]We request this in a trivial case below in Problem 2 and the following problems in this Section. We make the same request in a less trivial context in our model-fitting *Chapter* (Hogg *et al.* 2010a, Problem 6 and 7), where we ask the Reader to implement MCMC for a useful mixture model.

[19]Technically this is the *Metropolis* algorithm rather than the Metropolis–Hastings algorithm. The difference is that in the Metropolis–Hastings algorithm, we permit the proposal distribution to disobey detailed balance and then we correct the acceptance/rejection step to recover detailed balance.

[20]There is a very mathematical discussion in Geyer (2011) that we do not fully understand. There is a more heuristic answer on *Wikipedia* that we are happier with!

[21]Actually, our own favorite method, *emcee*, has a proposal pdf that does violate detailed balance in this way and has a compensated acceptance probability. Read the paper Foreman-Mackey *et al.* (2013) for more details.

[22]We also make this point in a previous *Chapter* (Hogg *et al.* 2010a); the ambitious reader will find in that *Chapter* a solid example project for using MCMC in a real context.

[23]Most code will be infinity safe if written in the form in this paragraph.

However, there can be issues if *both* $f(\theta')$ *and* $f(\theta_k)$ are negative infinity; in (present-day) Python this will return a `NaN` rather than an infinity or zero. That's a problem and is a case that should get caught (probably way before the accept–reject step!).

[24]Well, technically, MCMC *can* be used to sample a likelihood function, but the samples would be samples of possible *data* not possible *parameters*. The purist might say that even this is not sampling a likelihood function, because you should only call it a "likelihood function" in contexts in which you are treating the data as fixed (at the true data values) and the parameters as variable. In this context, the likelihood function is *not* a pdf for the parameters, so it can't be sampled.

[25]This point—that you should be using proper priors—is just basic Bayesian good practice, often violated but never for good reasons.

[26]There is very good coverage of most of the points in this Section, and more, in a set of lecture notes by Patrick Lam up at http://www.people.fas.harvard.edu/~plam/teaching/methods/convergence/convergence_print.pdf.

[27]Famously, there is "no free lunch" (Wolpert & Macready, 1997): You can't find the global optimum of a general optimization problem without exhaustive search of all possibilities. This is very closely related—somehow—to the difficulty of sampling.

[28]For example, you can (almost) never say that your chains are definitely converged, or that you have the posterior pdf correct to some given level of accuracy. The reason is related to the "no free lunch" theorem of discrete optimization: In most problems there is no way you could have searched your parameter space finely enough to know this. There are some exceptions of course. In one kind of exception, your problem is convex or Gaussian, and you can analytically show that there is exactly one mode in the density and where it is. Of course in these cases you rarely need MCMC to solve your problems! In another kind of exception, you can say something about the finite width or shape of the modes of the likelihood function (and prior pdf). For example, sometimes the Cramér–Rao bound tells you that modes of the density must be smoother than some finite amount. Then an exhaustive search of parameter space followed by MCMC can in principle have provable properties. But, as we say, these situations are rare.

[29]We sometimes see MCMC methods compared according to burn-in times, which, for one, depends strongly on initialization, and, for two, depends strongly on tuning and dynamics. Similarly we often see comparisons in terms of acceptance ratio. In principle the only question is how precise are one's inferences given a certain amount of computation. This is set by the autocorrelation time and (pretty much) nothing else, for reasonably converged chains. In the real world, of course, one must add to the CPU time the investigator time spent tuning the method (and thinking about initialization) but there is never (to our knowledge) a condition in which burn-in time is the dominant consideration in choosing an MCMC method.

[30]A. Sokal has a clear discussion of this topic and several proposals for robust autocorrelation time estimation in a set of lecture notes `http://www.stat.unc.edu/faculty/cji/Sokal.pdf`.

[31]Gelman & Rubin (1992).

[32]The calculation of the autocorrelation function can be seen as a convolution and it can, therefore, be computed using the fast Fourier transform in $\mathcal{O}(N \log N)$ operations instead of $\mathcal{O}(N^2)$ for a naive implementation.

[33]The recipe given on page 16 of A. Sokal's notes (`http://www.stat.unc.edu/faculty/cji/Sokal.pdf`) might be helpful. Note that the definition of $\tau$ in that document is half of the definition that we use.

[34]The argument and assumptions underlying the 0.234 fraction (and higher acceptance fractions at lower numbers of dimensions) are laid out in Gelman *et al.* (1996b).

[35]This has a lot to do with Gibbs sampling, which is discussed in Section 9.

[36]See Pasarica & Gelman (2010).

[37]When the dimensionality of the parameter space gets very large, almost all samples will be *very far* from the optimum of the posterior pdf. One way to think about this is to think of a random Gaussian draw in $D$ dimensions: A typical draw will be $\sqrt{D}$ standard deviations Euclidean distance from the maximum of the Gaussian. That's a long way when $D = 100$ or 1000! Another way to think about it is that there are a lot of ways to move *away* from the point you care about, but very few ways to move close to it. The upshot is that even with an *enormous* sampling, if you are in large dimensions, not

a single sample will be very near the optimum.

[38]Geyer (2011) takes this view.

[39]Foreman-Mackey *et al.* (2013).

[40]You might want to open a bottle of wine or two while the code runs.

[41]We intend to write a subsequent *Chapter* in this series about these issues. The idea is that you can make principled decisions by optimizing the expected utility under the posterior, given the data. This is a great idea! Of course we also have very deep, fundamental reasons that you *can't know your utility precisely*. The big issue is that your only sensible utility involves an integral out to the "long term", and the long term (by definition) includes outcomes that are outside your present-day quantitative model.

[42]For the median-of-sampling value, $q(\theta)$ needs to return a scalar, but technically, for the mean-of-sampling, $q(\theta)$ can be a vector or something high-dimensional.

[43]We usually create a 68-percent (or 95-percent) interval or region by excluding the top and bottom 16 percent (or 2.5 percent) of the posterior samples. Some more aggressive investigators find the smallest interval that contains 68 (or 95) percent of the samples. Both are legitimate, and they are near-identical for distributions that are near-symmetric around the mean.

[44]One amusing thing about samplings, which are so beloved of us probabilistic (Bayesian) reasoners, is that anything we *do* with a sampling, like estimate an integral or a quantile, is just an old-school frequentist estimator. A frequentist estimator of a Bayesian quantity, to be sure, but a frequentist estimator nonetheless.

[45]We don't like this "credible" terminology, though we sometimes adopt it to avoid confusion.

[46]The sampler samples the pdf fairly. The tallest peak in the posterior is not guaranteed to be—and in general is not—also the peak with the greatest posterior "mass". That is, the samples in a fair sampling will not necessarily come predominantly from the *tallest* peak in the posterior pdf. They will come from the peak with the greatest total integral of the likelihood over the prior. These issues might sound "academic", but when the number of parameters gets large (greater than, say, 10), many normal intuitions one

might have (if any) about what is reasonably likely in a pdf are regularly violated.

[47]If you have a $K$-point sampling in a $D$-dimensional parameter space, there is some best sample $\theta_k^{(\mathrm{best})}$. Now imagine that you want to make that best sample 10 times better—that is, you want to find a point $\theta^{(\mathrm{better})}$ that is ten times closer to the true optimum. How many more samples do you need to take? In the limit that $K$ is large, on average, you have to take a factor of $10^D$ more samples! Any reasonable optimizer is far, far better than this (in terms of compute time).

[48]We choose not to count the number of papers out there that claim to be "Bayesian" but then deliver the optimum of a posterior PDF. That sacrifices probabilism, but also all the useful things you get from delivering a posterior, like protection from over-fitting, and non-approximate uncertainty propagation. You certainly don't need to be a Bayesian—you can do most of the same science as a frequentist—but if you are using MCMC you probably are doing so because you want the benefits of Bayesianism.

[49]And we do it ourselves. For examples, you can look at Lang & Hogg (2012) and Foreman-Mackey *et al.* (2014).

[50]This point is made well by MacKay (2003).

[51]We have been pushing this approach in our own work; our hierarchical inferences (Hogg *et al.*, 2010b; Foreman-Mackey *et al.*, 2014) make use of importance samplings, starting at posterior samplings from interim prior pdfs. We have used this technology for computational convenience, but it is a framework for building subsequent analyses on the posterior samplings provided by other investigators. One key—if you want your posterior samplings to be useful—is that you must release (along with your posterior sampling), the value of the prior pdf at the locations of the samples. That is, you must "decorate" the samples (augment them) with prior pdf values or calls, or else release an executable version of your prior pdf, into which all your samples can be put as inputs. In *principle* it is enough to release a *description* of your prior pdf, but in practice it is rarely correct or specific enough for another investigator to duplicate exactly. So just augment your samples!

[52]Our favorite package these days is our own *corner.py*. Find it on the internets.

[53]You can get essentially infinite autocorrelation times if the function has isolated regions of finite density separated by large seas of zero or near-zero density. One disturbing thing about these situations is that any empirical measure of autocorrelation time will give a finite answer, but the density won't be properly sampled, ever. In principle you can find these with multiple re-starts of the chain and (things like) the Gelman–Rubin diagnostic (Gelman & Rubin, 1992).

[54]Your best interface to these literatures is often a colleague in another department, like applied mathematics, statistics, or computer science. Break down those walls!

[55]One check-point in this literature is the compilation volume by Brooks *et al.* (2011), but much has been developed and changed even since that was written.

[56]Foreman-Mackey *et al.* (2013)

[57]One that we are interested in is *Kombine* (Farr & Farr, in preparation).

[58]For example, we find empirically that it is better to initialize the ensemble in a ball that is smaller than the full posterior pdf width, not larger. For another example, the ensemble produces a set of chains that can be used to compute the Gelman–Rubin diagnostic (Gelman & Rubin, 1992), but this is not a conservative test if the initialization was made in a small ball.

[59]For example, *JAGS*; find it on the internets.

[60]The point is that sometimes there are some parameters that only affect a very small number of data points, or parameters that change the model or the likelihood function and prior values in a very simple-to-compute way. If such simple parameters exist, it makes sense to write your likelihood and prior code to capitalize on these simplicities. This is not always trivial, because it may involve caching parts of the calculation and so on. Details would be beyond our scope.

[61]For example, *JAGS* automatically parallelizes Gibbs problems.

[62]The idea is that for (almost) any function you can write, a robot can write the function that is the derivative of that function, using simple differentiation rules and the magic of the chain rule. Differentiation is a typographic operation on computer functions! The idea is *not* to take numerical (small

difference) derivatives of the code; there is no way that finite-difference differentiation could help your sampling in the long run.

[63]For example, we use importance sampling to convert individual-planet exoplanet inferences (all performed with dumb priors) into a full-population consistent hierarchical model in Hogg *et al.* (2010b).

[64]DFM: Cite us and Murray here.

[65]For example, see Skilling (2006) and Brewer *et al.* (2011)

[66]Our attempt to sample this very problem is Brewer *et al.* (2013).

[67]If you aren't unit testing, you are probably making some very big mistakes.

[68]It is easy to think you have flat priors when in practice your sampler has flat priors in the logarithms of the parameters or inverses of the parameters. Also, it is easy to have improper priors when you think they are proper, because limits are not working, or you were wrong about the convergence of some integral. Sometimes the prior pdf in practice has different edges than you think because there is some censoring happening that you haven't considered. And then, of course, it is possible for your sampler itself to be buggy.

[69]Such as *emcee* (Foreman-Mackey *et al.*, 2013); one nice thing about the ensemble sampler is that it gives you great diagnostic information automatically.

[70]Note how much we like to cite Foreman-Mackey *et al.* (2013)!

[71]The real quotation is a parenthetical sentence "Remember that all models are wrong; the practical question is how wrong do they have to be to not be useful." (Box & Draper 1987, p. 74). It is amazing how rarely this is ever quoted correctly; or maybe there is another source for this quotation?

[72]Physicists sometimes forget this, and believe that various physical theories, such as quantum electrodynamics or the cold-dark-matter cosmological model, are strictly true. First of all, there is no way to know that for sure, and second of all, even if they *are* true, they don't precisely explain any observation, which is also affected by various kinds of auxilliary effects and noises for which there is (and can be, for deep reasons, to be discussed in

other *Chapter*s in this series) no extremely precise model. So in detail, any model of any specific observation must be wrong, even in the (exceedingly unlikely) event that the fundamental model is correct.

[73]Sometimes Bayesians like to complain that the chi-squared statistic is a measure of the size of your data! That's true when your model is wrong (that is, always).

[74]Cross-validation is an incredibly valuable set of techniques that all astronomers should know. it is not clear what to cite here, but one possibility is Geisser (1993).

[75]The posterior predictive check is one of the Bayesian equivalents of a goodness-of-fit test (Gelman *et al.*, 1996a).

[76]And be sure not to forget that there are (at least) as many assumptions encoded in your likelihood function as there are in your prior pdfs.

# References

Box, G. E. P. & Draper, N. R. 1987, *Empirical Model Building and Response Surfaces*, John Wiley & Sons, New York, NY ISBN:978-0471810339.

Brewer, B. J., Pártay, L. B. & Csányi, G., 2011, Diffusive nested sampling, *Stat. Comput.* **21** 649 DOI:10.1007/s11222-010-9198-8.

Brewer, B. J., Foreman-Mackey, D., & Hogg, D. W., 2013, Probabilistic catalogs for crowded stellar fields, *Astron. J.* **146** 7 DOI:10.1088/0004-6256/146/1/7.

Brooks, S., Gelman A., Jones G., & Meng, X.-L., eds. 2011, *Handbook of Markov Chain Monte Carlo*, Chapman and Hall / CRC Press ISBN:978-1-4200-7941-8.

Foreman-Mackey, D., Hogg, D. W., Lang, D., & Goodman, J. 2013, *emcee*: The MCMC Hammer, *Pubs. Astron. Soc. Pac.* **125** 306–312 DOI:10.1086/670067 (also *arXiv*:1202.3665).

Foreman-Mackey, D., Hogg, D. W., & Morton, T. D., 2014, Exoplanet population inference and the abundance of earth analogs from noisy, incomplete catalogs, *Astrophys. J.* **795** 64 DOI:10.1088/0004-637X/795/1/64.

Geisser, S., 1993, *Predictive Inference: An Introduction*, Chapman & Hall ISBN:978-0412034718.

Gelman, A., & Rubin, D. B., 1992, Inference from iterative simulation using multiple sequences, *Statist. Sci.* **4** 457–472.

Gelman, A., Meng, X.-L., & Stern, H., 1996a, Posterior predictive assessment of model fitness via realized discrepancies, *Statistica Sinica*, **6**(4) 733–760.

Gelman, A., Roberts, G. O., & Gilks, W. R., 1996b, Efficient Metropolis jumping rules, in Bernardo, J. M, Berger, J. O., Dawid, A. P., & Smith, A. F. M, eds., *Bayesian Statistics* **5**, Oxford University Press 599–607.

Geyer, C. J., 2011, Introduction to Markov Chain Monte Carlo, in Brooks *et al.* (2011), 3–48.

Goodman, J. & Weare, J., 2010, Ensemble samplers with affine invariance, *Comm.Åppl. Math. Comp. Sci.*, **5**(1), 65-80 DOI:10.2140/camcos.2010.5.65.

Hogg, D. W., Bovy, J., & Lang, D., 2010a, Data analysis recipes: Fitting a model to data, *arXiv*:1008.4686v1.

Hogg, D. W., Myers, A. D., & Bovy, J., 2010b, Inferring the eccentricity distribution, *Astrophys. J.* **725** 2166–2175.

Hou, F., Goodman, J., & Hogg, D. W., 2014, The probabilities of orbital-companion models for stellar radial velocity data, *arXiv*:1401.6128.

Lang, D. & Hogg, D. W., 2012, Searching for comets on the World Wide Web: The orbit of 17P/Holmes from the behavior of photographers, *Astron. J.* **144** 46 DOI:10.1088/0004-6256/144/2/46.

MacKay, D. J. C., 2003, *Inference, Information Theory, and Learning Algorithms*, Cambridge University Press ISBN:9780521642989.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E., 1953, Equation of state calculations by fast computing machines, *J. Chem. Phys.*, **21** 1087–1092.

Neal, R. M., 2011, MCMC Using Hamiltonian Dynamics, in Brooks *et al.* (2011), 113–162.

Pasarica, C. & Gelman, A., 2010 Adaptively scaling the Metropolis algorithm using expected squared jumped distance, *Statistica Sinica*, **20**(1) 343–364.

Skilling, J., 2006, Nested sampling for general bayesian computation, *Bayes. Anal.* **1**(4) 833-860 DOI:10.1214/06-BA127.

van der Walt, S., Colbert, S. C., & Varoquaux, G., 2011, The NumPy Array: A structure for efficient numerical computation, *Computing in Sci. & Eng.* **13**, 22–30 DOI:10.1109/MCSE.2011.37.

Wolpert, D. H. & Macready, W. G., 1997, No Free Lunch Theorems for Optimization, *IEEE Trans. Evolutionary Computation* **1** 67.