Markov chain Monte Carlo sampling

If you are trying to estimate the best values and uncertainties of a many-parameter model, or if you are trying to compare two models with multiple parameters each, fair and thorough sampling of the parameter space becomes challenging. Imagine, for example, that for a given model you have time to calculate the likelihood of the data for a million different parameter combinations. If you have only one parameter, that means that you can look at a million values of that parameter. If you have two parameters, you could do a grid with a thousand values of each of those parameters. If you have six parameters, you could only afford ten values per parameter; if you have twelve, then you could only sample about three values per parameter. At some point, it's clear that a grid sampling such as this cannot probe the posterior probability thoroughly enough to be useful.

It is, of course, possible to use a maximization routine to find peaks in the posterior. But even if the search for the maximum is successful (and it very well might not be!), it isn't enough: almost always, you would like a sense of the *uncertainties* in your model parameters, not just their best values.

This is where Markov chain Monte Carlo sampling (or MCMC for short) comes in. It's not perfect, and there are times when one variant or another is preferred, but overall the development of MCMC techniques has greatly improved the practical utility of Bayesian algorithms, which in turn has meant that it is far easier to be rigorous in statistical analyses of astronomical data sets.

Let's begin by deconstructing the name. A Markov process is a series of steps in which each step only depends on the previous step rather than on a larger set of previous states of the system. In particular, if you have evaluated a certain parameter combination, the next parameter combination you evaluate can be considered as a step from the previous combination. "Monte Carlo" (named after the casino!) is often used to denote randomness in a procedure. Thus MCMC means a random sampling technique in which each step depends only on the previous step.

But of course pure randomness isn't useful. If you just take haphazard stabs at the parameter space, you won't get very far. Instead, you would like to go toward regions of higher posterior, so that you don't waste time doing calculations in regions of parameter space that are a poor match to the data. In fact, what you'd *really* like is a set of parameter combinations such that the probability of getting a parameter combination is proportional to the posterior probability density at that combination. If you can manage that, then you have a fair and representative sampling of the posterior.

The Metropolis-Hastings algorithm has these properties. If you use it to sample from a posterior probability distribution $P(\vec{x})$ (where $\vec{x}$ is the set of parameters in the model), then

the algorithm in sketch form is as follows:

1. Start at some position $\vec{x}_1$ in parameter space. Compute the posterior probability density $P_1 = P(\vec{x}_1)$ at $\vec{x}_1$.
2. Somehow, propose a new parameter combination $\vec{x}_2$ to evaluate. The method of proposal is important for making the algorithm fast, but we'll get into that later.
3. Calculate $P_2 = P(\vec{x}_2)$.
4. Draw a random number y, uniformly between 0 and 1.
5. If $y < P_2/P_1$, then you accept the proposal, and thus your next step is from $\vec{x}_2$ to some $\vec{x}_3$. If $y > P_2/P_1$, then you do *not* accept the proposal, and thus your next step is from $\vec{x}_1$ to some $\vec{x}_3$. **Note**: this assumes that your proposal distribution is symmetric, in the sense that the probability of proposing $\vec{x}_2$ from a current state $\vec{x}_1$ is the same as the probability of proposing $\vec{x}_1$ from a current state $\vec{x}_2$; see https://en.wikipedia.org/wiki/Metropolis–Hastings_algorithm for a more general discussion.
6. Repeat until some convergence criterion is satisfied. After you deem that the sequence has converged, continue for some number of iterations to sample the parameter space.

This satisfies the property of *detailed balance* (and indeed was derived from that principle). You may remember detailed balance from considerations of equilibrium, where in equilibrium all processes and their inverses must be in balance. Here, the equivalent is

$$\pi(\vec{x}_2|\vec{x}_1)P(\vec{x}_1) = \pi(\vec{x}_1|\vec{x}_2)P(\vec{x}_2)\ , \tag{1}$$

which can be read "the probability of accepting a step from $\vec{x}_1$ to $\vec{x}_2$, times the posterior probability density at $\vec{x}_1$, equals the probability of accepting a step from $\vec{x}_2$ to $\vec{x}_1$, times the posterior probability density at $\vec{x}_2$." This also guarantees that if you have a set of samples that are picked according to the posterior probability density P, then if you update each of those samples, you will maintain the posterior probability density P. As a result, if you have a correctly sampled distribution then many samples from that distribution, using an MCMC, will map out your posterior probability.

The result of your MCMC walk through parameter space is a list of parameter combinations. If you are at step k, with parameter combination $\vec{x}_k$, and you propose a new combination $\vec{x}'$, then if the proposal is accepted you set $\vec{x}_{k+1} = \vec{x}'$. If the proposal is *not* accepted, then you repeat the previous combination: $\vec{x}_{k+1} = \vec{x}_k$. You'll note that if $P(\vec{x}') \geq P(\vec{x}_k)$ then the step will always be accepted, but even if the proposed probability density is *less* than the current probability density there is a chance that the proposed step

will be accepted. A set of parameter combinations from an MCMC run is sometimes called a "chain", and each step is a "link" in that chain.

Strengths and weaknesses of MCMC

The MCMC sampling approach is powerful, flexible, and if there are many parameters it is overwhelmingly faster than a grid search. In the long run, it is guaranteed to sample any probability density correctly.

However, as with any method, there are some caveats. Let's start with the question: at what point do you start counting the parameter combinations on your list, for the purposes of sampling the distribution? Think about it: if you began with some random guess, that's hardly likely to be representative of the distribution! In fact, unless you're amazingly lucky, the probability density at your first guess is almost certainly far, far, below the maximum probability density. This means that you need to have a set of initial steps (often called "burn-in" steps) that you take until you think you've converged. How many steps should that be? In fact, how do you know whether you've converged?

There are various approaches that have been taken. For example, you could specify in advance the number of burn-in steps that you will take before you start the official list of parameter combinations. That has the disadvantage that (1) maybe you didn't take enough burn-in steps, which would mean that you aren't converged yet, or (2) maybe you took too many burn-in steps, in which case your search was inefficient.

Another possibility is that you could keep track of the values of your posterior probability density. If the values have leveled off, according to some criterion, then you could declare that the values have converged. Yet another possibility is a variant on that: you could keep track of the autocorrelation function of your posterior densities, and judge convergence based on that.

There is no *guaranteed* way to determine convergence. To see why, let's picture a particular posterior density as a mountain range with very separate peaks. In your search for high posterior densities, you are climbing those peaks. But if you are near the top of one peak, i.e., if you are converged locally, that does not guarantee that you are at the highest peak. It could theoretically be that if you let your sampling run for a lot longer, there could be a fortuitous step to another, higher, peak.

Another caveat relates to the step size. If you start at a random spot in parameter space, you might want your initial steps to be relatively large, because you have no reason to believe that your initial guess is any good. But if you do that, then after a number of accepted steps you have presumably gone to higher and higher posterior densities. It therefore becomes progressively more difficult to find higher-posterior points in parameter space. As a result,

as time goes on, the probability that a proposal will be accepted becomes lower and lower, and the search rapidly becomes highly inefficient.

Thus if you take this approach you would want to, for example, reduce the step size after some number of consecutive proposals that were not accepted. That's a reasonable approach, but it too runs into a fairly common difficulty. That difficulty arises if two or more of your parameters are correlated in such a way that a contour is long along one direction, and short along another. As a specific example, let us borrow from Lecture 7 the two two-dimensional probability distributions

$$A : P(x, y) = \frac{1}{2\pi} e^{-x^2/2} e^{-y^2/2} \tag{2}$$

and

$$B : P(x, y) = \frac{\sqrt{256/31}}{2\pi} e^{-4(x-y)^2} e^{-4(x+y)^2/31} \ . \tag{3}$$

We reprint the figures of the contours at $e^{-1/2}$ times the maximum probability density, in Figure 1. Imagine a proposal strategy in which we initially take steps of the same size in x and y. Eventually we have some number of consecutive unsuccessful proposals, and thus we reduce the step sizes by the same factor. For the circular distribution A, this is no problem; the region is sampled efficiently and well. But for the narrow and tilted ellipse B, it *is* a problem, because the step sizes need to shrink down to roughly the size of the minor axis. As a result, because the steps can't easily explore the major axis, the sampling is inefficient: it will take many steps to explore the full ellipse. Of course, if we somehow know in advance that the ellipse is tilted and has a high eccentricity, we can adjust the steps accordingly (by, for example, taking long steps in $x + y$ and short steps in $x - y$). But what if we don't have that knowledge?

With that in mind, let us now think about some improvements.

Improvements to the MCMC algorithm

We'll begin with the issue of multiple peaks. There is a straightforward way to get a better idea of the peak structure, and a more complicated but ultimately more thorough way to do it.

Trying multiple starting points.—For the straightforward way, we presented the metaphor of the mountain range as representative of the posterior probability density. If you were dropped randomly somewhere in the range, and could only feel around blindly, you'd basically head up the nearest mountain. In the same way, if you start at a random point in parameter space, and take steps, you're likely to go up a particular peak if there are many. By trying multiple starting points, you increase your chances of exploring multiple peaks. If those peaks all have comparable probability densities, then they all will contribute. If

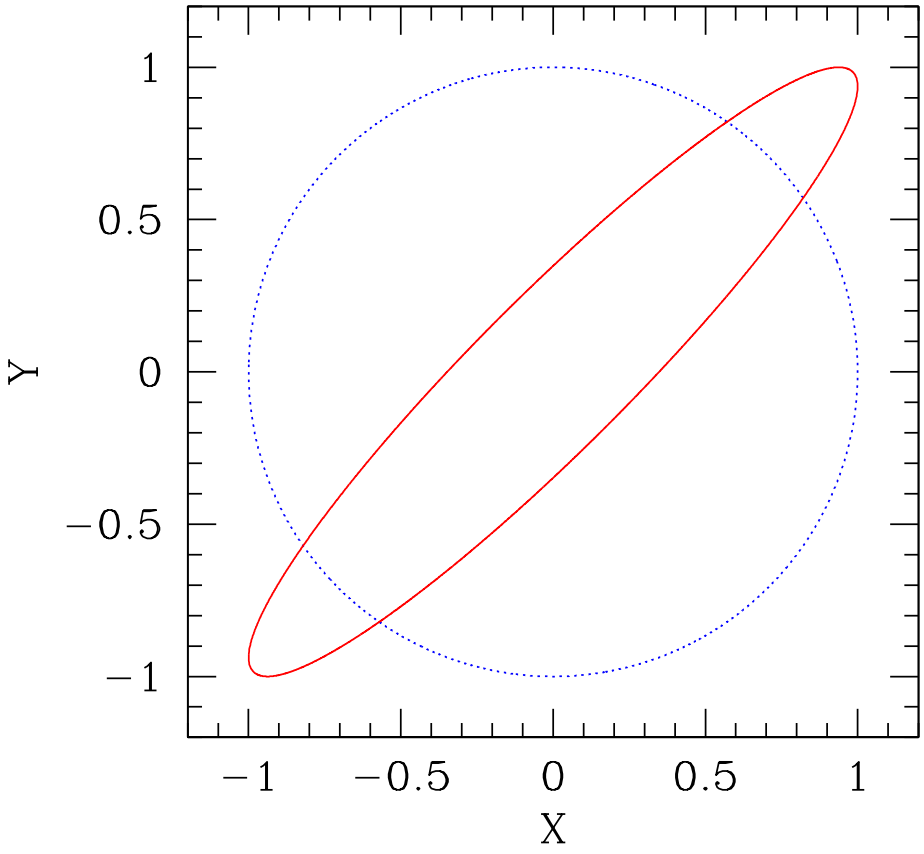

Fig. 1.— Contours for $P(x, y) = e^{-1/2}$ times the maximum probability density, for the two distributions given in the text (this is taken from Lecture 7). Here our focus is on implications for sampling. If our initial step sizes in x and y were initially the same, and if we reduce each by the same factor when there are some number of consecutive unsuccessful proposals, then distribution A (the circular distribution) will be sampled just fine, but distribution B (the slanted elliptical distribution) will be inefficiently sampled, because only small steps will be taken due to the small minor axis of the ellipse, but there is a lot of room to be explored along the major axis.

instead there is one peak that is much higher than the others, then only it will play an important role.

Note that a procedure that is sometimes used is to apply a maximization routine, and thus to find the single highest peak that the routine can manage, and then to start your parameter search on that peak. This is *not* a good idea in general. It will work fine if that peak is by far the highest in the entire parameter range, but if there are comparable-height peaks elsewhere the result will be that you will produce a drastic underestimate of the uncertainties in your parameters because you are focused on only one peak.

Flattening of the posterior.—The more complicated but more thorough approach can be used in different ways. The reason that the mountain range is difficult to explore is that the peaks are very high compared with the valleys between them. That is, if you happen to be near the top of one mountain, there is a negligible chance that you will find your way down the valley and to the top of the next mountain.

One way to deal with this is to "flatten" the posterior landscape. For example, suppose that you have a posterior $P(\vec{x})$, which we represent as $P(\vec{x}) = e^{\ln[P(\vec{x})]}$. Instead of sampling from P, we could sample from

$$P_T(\vec{x}) \equiv e^{\frac{1}{T}\ln[P(\vec{x})]} \ , \tag{4}$$

where $T > 1$ is sometimes called the "temperature" in analogy to some thermodynamic concepts. The larger T is, the smaller are the differences between the peaks and valleys. For sufficiently large T, then, your proposals will be accepted all over the parameter space.

Ultimately, of course, you'd like to sample correctly from your original posterior, which effectively has $T = 1$. You could do this in a stepwise procedure: do a search with a large T, tighten the limits on your parameters as a result, do a search with a smaller T, tighten the limits further, and so on, until you eventually have a constrained enough parameter space that the $T = 1$ search is efficient. This can work well in many cases. But a more popular choice is *parallel tempering*, which we will now describe.

Parallel tempering.—The idea here is that, rather than running a single search on your original posterior, or rather than running a single search with a single value of T, you run several in parallel and have them communicate in a way that maintains detailed balance. The parallel runs are called "rungs" (on a temperature ladder), and each have a different temperature T. For this method, you actually keep the prior the same for each rung; you only flatten the likelihood. A popular, and good, choice is to have the temperatures arranged in an exponential ladder: for example, the lowest rung would have $T = 1$ (i.e., the original likelihood; this is the only one that you read out), the next rung might have $T = 2$, the next would have $T = 4$, the next $T = 8$, and so on. The idea is that the high-temperature rungs survey the parameter space and communicate good locations to the lower-temperature rungs, and the $T = 1$ rung provides the most precise exploration.

There are two types of steps in this approach: proposed steps *within* a rung, and proposed *exchanges* between parameter combinations on different rungs.

The steps within a rung are just like they were before, but in each rung we are now sampling from $e^{\frac{1}{T}\ln\mathcal{L}}q$, where $\mathcal{L}$ is the likelihood and q is the prior.

The new element is the proposed exchanges between rungs. Let the proposed exchange be between a parameter combination $\vec{x}$ on rung i (with temperature T_i), and a parameter combination $\vec{y}$ on rung j (with temperature T_j). By *exchange* we mean that if the proposal is successful, then after the exchange $\vec{x}$ will be on rung j (which still has temperature T_j), and $\vec{y}$ will be on rung i (which still has temperature T_i). If, for any parameter combination $\vec{z}$ we indicate the posterior probability density on rung k as $P_k(\vec{z})$ (where $P_k = e^{\frac{1}{T_k}\ln\mathcal{L}}q$), then the acceptance probability that maintains detailed balance is

$$\text{prob(exchange)} = \min\left(1, \frac{P_j(\vec{x})P_i(\vec{y})}{P_i(\vec{x})P_j(\vec{y})}\right) \ . \tag{5}$$

That is, the probability of acceptance is the product of the posterior probability densities at the two rungs when the parameter combinations are switched, divided by the original product.

The advantage of this method is that it can explore a multimodal posterior (i.e., one that has many separated peaks). The disadvantage is that it requires more evaluations of the posterior in each step. You'll need to decide what works best for your particular problem.

Now we have to address the question: what if the natural contours of the posterior are thin ridges, particularly if those ridges are not aligned with the parameters?

Affine-invariant MCMC.—In 2010, Goodman and Weare (Comm. App. Math. and Comp. Sci., vol. 5, no. 1, pages 65-80) presented a powerful and flexible approach to MCMC sampling that has now been incorporated widely in astronomical analyses. It is possible that the method was suggested earlier, but I'm not well-versed in its history.

Goodman and Weare begin with the following thought experiment, which we'll phrase in terms of posterior probability densities although the method is more general. Suppose that your posterior is

$$P(x_1, x_2) \propto \exp\left(-\frac{(x_1 - x_2)^2}{2\epsilon} - \frac{(x_1 + x_2)^2}{2}\right) \ , \tag{6}$$

and that $\epsilon \ll 1$. You'll recognize this from some of our previous examples. If the axes that you use to do your sampling are x_1 and x_2, then you run into the problems that we discussed previously: you end up taking tiny steps, which makes your sampling of the space very inefficient.

In contrast, if you make the affine transformation (which is a slight generalization of a

linear transformation)

$$y_1 = \frac{x_1 - x_2}{\sqrt{\epsilon}}, \qquad y_2 = x_1 + x_2 \ , \tag{7}$$

then your posterior becomes

$$P(y_1, y_2) \propto e^{-(y_1^2+y_2^2)/2} \ , \tag{8}$$

which is an easily-sampled symmetric Gaussian. After you do your thorough sampling in (y_1, y_2) space, it is simple to transform back to (x_1, x_2).

In most circumstances you won't know the structure of the posterior, so you can't do this kind of transformation by hand, but Goodman and Weare came up with a very clever approach that does this automatically.

Their core idea is that rather than just following one chain of links through parameter space, you follow many *walkers* simultaneously. The proposals for new steps for each walker then depend on the relative locations of the *other* walkers. This allows the group of walkers to evolve like an amoeba through parameter space, stretching and shrinking in ways that follow the posterior. Thus, for example, the thin ellipse in the previous example would be matched by the eventual distribution of the walkers. As a result, very long, thin linear structures are followed easily in this approach. As you will see below, even nonlinear structures (e.g., curves) can be handled surprisingly well using this method.

With that, let's actually go through the Goodman and Weare algorithm.

The Goodman and Weare algorithm.—In this algorithm, we start with a set of parameter combinations (aka walkers). The method of selection of the set is up to the user; what I've found is that it is most convenient to begin with a fiducial parameter combination and then perturb the parameter values only slightly to form the rest of the initial walkers. This has the advantage that because (as we'll see below) proposed updates to walkers involve steps in the direction of other walkers, those steps have a good chance of reaching high-posterior portions of the parameter space. The potential drawback is that if your walkers get caught near a local maximum, it may not be easy to get them out if all are in roughly the same place. The other extreme is to start with all of the walkers in different random places in the parameter space. The problem there is that the walkers can easily end up standing on different peaks, which means that steps between them are almost certain to end up on valleys, which implies in turn that only a vanishing fraction of proposals are accepted.

In any case, following Goodman and Weare, let us suppose that we have L walkers in our set. Now, rather than proposing and evaluating only one update per step, we need to propose and evaluate updates for each of the L walkers in a single step. There are various ways that the proposed updates can happen, but the one that Goodman and Weare recommend, and that I have found to be good in practice, is what they call a *stretch move*, where the proposal

is:

$$X_k(t) \to Y = X_j(t) + Z(X_k(t) - X_j(t)) \ , \tag{9}$$

where $j \neq k$ is a different walker than k, and Z is a scaling variable. They find that a good probability distribution from which to select Z is

$$g(Z) \propto \begin{array}{ll} \frac{1}{\sqrt{Z}} & \text{if } Z \in \left[\frac{1}{a}, a\right] \ , \\ 0 & \text{otherwise} \ . \end{array} \tag{10}$$

Here the parameter $a > 1$; they recommend, and I find, that $a = 2$ is a good choice.

The final step is the determination of the probability of acceptance of a proposal. For this, suppose that your parameter space has n parameters. Then if the posterior probability density at some parameter combination X is $P(X)$, the correct acceptance probability is

$$\text{prob} = \min\left\{1, Z^{n-1}\frac{P(Y)}{P(X_k(t))}\right\} \ . \tag{11}$$

To essentially copy from Goodman and Weare, the pseudocode involved is therefore just: "The operation $\vec{X}(t) \to \vec{X}(t+1)$ using one stretch move per walker is given by:

for k = 1,. . ., L
  choose $j \neq k$ at random
  generate $Y = X_j(t) + Z(X_k(t) - X_j(t))$, all Z choices independent
  accept, set $X_k(t+1) = Y$, with probability (11)
  otherwise reject, set $X_k(t+1) = X_k(t)$"

That's it! This is fundamentally a very simple procedure. The last note is that what you print out in your file of parameter combinations is the *whole* set of walkers, every time. For example, if you have 50 walkers, you print out the parameter combinations (and log posteriors) of all 50 in every step; you do not just print out the accepted steps. This means that you might reprint the same parameter combination many times, if the proposed updates for some walker are not accepted for several consecutive steps.

As an example of how this works, we show in Figure 2 several snapshots of the locations of four independent sets of walkers, which are sampling the Rosenbrock density:

$$P_{\text{rosen}}(x_1, x_2) \propto \exp\left(-\frac{100(x_2 - x_1^2)^2 + (1 - x_1)^2}{20}\right) \ . \tag{12}$$

Because this density is very narrow, and strongly curved, it presents a significant challenge to the (fundamentally linear) affine-invariant sampler, but the sampler is up to the challenge. This is far better than a standard MCMC would do!

Using MCMC sampling for parameter estimation.—So you've sampled your space using some variant of the MCMC technique. Your sampling has converged and now you have

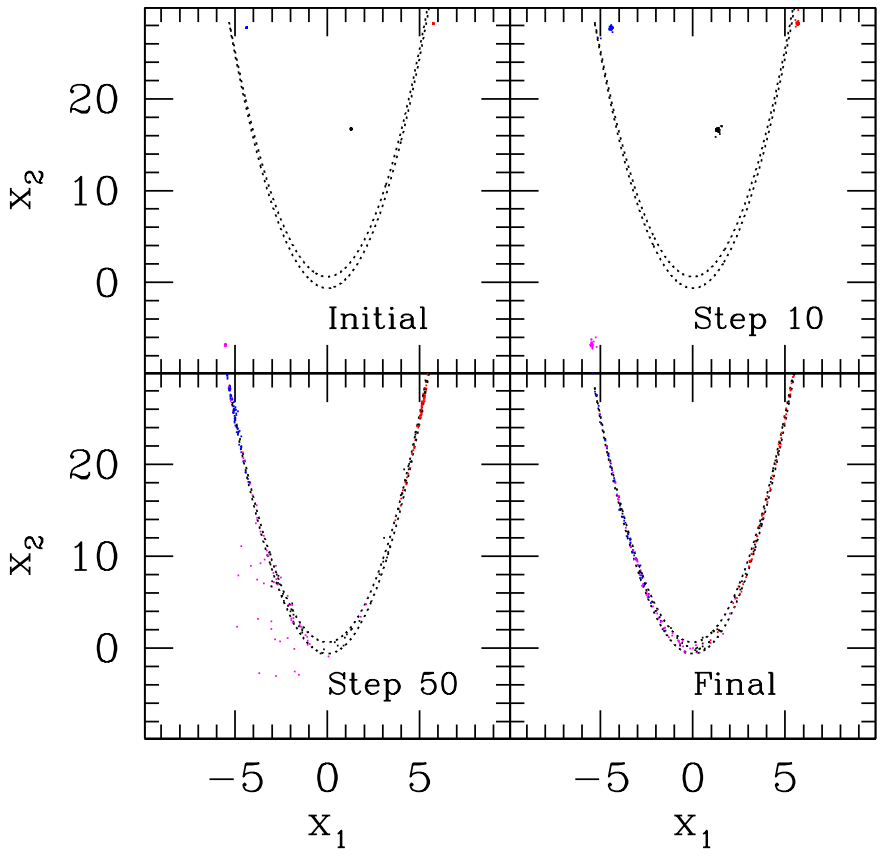

Fig. 2.— Several snapshots of the locations of four independent sets of 60 walkers each, exploring the Rosenbrock density (equation 12). The dotted black smiley curve shows the location in (x_1, x_2) space of the points with $P_{\rm rosen}(x_1, x_2) = e^{-2}$ (the maximum value is $P_{\rm rosen,max}(0, 0) = 1$). Each set of walkers has a single color: black, red, blue, or magenta. The top left shows the initial (x_1, x_2) locations; here we have made the recommended choice of picking a random fiducial initial location for the first walker in a set, and then varying the initial coordinates of the others in that set only slightly from the fiducial values. The top right shows the locations of each set of walkers after 10 steps; the bottom left after 50 steps; and the bottom right after the algorithm plus my own choices indicates that each set of walkers is finished. You can see that, although the affine-invariant method is really optimized for linear isoprobability contours, the independent sets of walkers do an excellent job of sampling this difficult distribution.

a list of parameter combinations, post-convergence, which you believe fairly represents the posterior probability distribution. But now you'd actually like to use your list to do things like estimate parameters. How do you do it?

The good news is that for estimates of the posterior probability distributions of each of your parameters separately, it's very easy! That is, if what you want is your full posterior probability distribution, marginalized over all the parameters except one (recall that this means that you want to *integrate* the full posterior over all the parameters except your chosen one), it's simple: all you need to do is to take the full list of parameter combinations and sort them in increasing order of the value of the parameter you care about, and that gives you the distribution!

To be more explicit, suppose that you have ended up with 10,000 parameter combinations that you believe represent a converged, fair sampling of the posterior. Say that you are interested in parameter 3 in your list. You sort your 10,000 parameter combinations in increasing order of their value of parameter 3. After sorting, then (for example) the middle 68% of your properly marginalized posterior parameter density for parameter 3 alone goes from element 1,600 (i.e., 16% of the way through your list of 10,000) to element 8,400 (i.e., 84% of the way through your list of 10,000).

Of course, you'll want to perform some sanity checks. If you have taken at least a few independent samples, do your samples converge to the same values of the posterior? Even if they do, are the parameter value sets from independent samples consistent with each other? You can see from Figure 2 that the four independent samples don't each, individually, stretch all the way around the smile. But over time, with many steps (and recall that each snapshot shows only one step), the cumulative distribution of an individual set of walkers would.

Getting multi-parameter credible regions from MCMC sampling is also possible, but it's more complex and the guidelines for doing so aren't as clear as they are for one parameter. One point of entry is to note that if you have a measure for the probability density in your converged sample (e.g., the number of points per parameter volume), you can use that by going to the densest region first, then to less and less dense regions, until you get to some desired fraction of the total number of points (which thus is that fraction of the total probability). As we've emphasized before, there is not a unique definition of the credible region for any number of parameters, except that it has to contain the specified fraction of the total probability, but this is as good as any.

Now it's your turn: as suggested in the accompanying computational exercise page, please write an affine-invariant MCMC sampler and apply it to the listed distributions, including the baryonic mass versus rotational speed data we explored before. What trends do you notice? What choices of the parameter a, or of termination conditions, give you the best results?

In summary, MCMC methods are extremely useful in sampling multidimensional parameter spaces. The recent improvements, such as affine-invariant samplers, have dramatically improved the speed and accuracy of the methods. There are codes such as EMCEE that implement these methods well, but in my opinion it is highly useful for you to do your own initial coding, so that you can understand the issues. In addition, any specific task will have its own optimizations and requirements, so if you have a code framework in hand there may be multiple uses that you find for it. Good luck!