1. MCMC Overview

In data analysis, as in many other aspects of (an admittedly sad) life, we are often faced with having to do high-dimensional integrals over likelihood functions. An example would be to calculate the expectation of a model parameter $\langle p_i \rangle = \int p_i \mathcal{L}(p) \mathrm{d}^n p / \int \mathcal{L}(p) \mathrm{d}^n p$. Similarly, we can calculate uncertainties on parameters by calculating their variances since $\mathrm{Var}(p_i) = < p_i^2 > - < p_i >^2$. If the likelihood is reasonably well described by a Gaussian, a Taylor expansion around the peak (found via something like Levenberg-Marquardt) can work very well.

For broad classes of problems, though, this can fail, and fail spectacularly. A classic example is a χ^2 surface with multiple minima. LM is unlikely to notice the multiple minima, and its reported error bars will be derived from the curvature around the minimum it found. If you blindly use those error bars, you can underestimate your uncertainties by orders of magnitude. Evaluating the integrals on a grid of points rapidly becomes intractable, and so we will need another technique.

A very robust (and easy to code) tool that can handle complicated likelihood surfaces is Markov Chain Monte Carlo (MCMC). The downside is that it's hundreds of times slower than Newton's method or LM, and that's on a good day. The inspiration for MCMC comes from thermodynamics. I'll go through the motivation, but don't worry if you don't follow all the details. You'll still be able to happily and usefully use MCMC.

2. MCMC Derivation

We'll start with the seemingly unrelated problem of a two-state system with populations n_1 and n_2, and transition probabilities p_{12} for a particle in state 1 to transition to state 2, and p_{21} for the reverse. The time evolution of the system is then

$$\frac{\mathrm{d}n_1}{\mathrm{d}t} = -n_1 p_{12} + n_2 p_{21}$$

and similar for state 2. If we let the system come into equilibrium, then the derivative must be equal to zero, so we have

$$\frac{n_1}{n_2} = \frac{p_{21}}{p_{12}} \tag{1}$$

That is, the ratio of the population of the two states is equal to the ratio of their transition probabilities. We could carry out a simulation of this with a single particle. At every time step, we roll a die, and based on our roll and on the transition probabilities, we decide if the particle jumps states or not. If we wait long enough, we know that the fraction of the time the particle spends in state 1 divided by the fraction of the time it spends in state 2 is just the ratio of p_{21} to p_{12}. A fact that will become crucial is that the absolute magnitudes of the transition proabilities don't matter - it is just the ratio. If the probabilities are very low, we will have to wait a very long time for the particle's behavior to approach the average, but if we wait long enough, it will.

How does this extend to higher numbers of states? Well, we can convert our two state system into a matrix equation:

$$\begin{pmatrix} \dot{n_1} \\ \dot{n_2} \end{pmatrix} = \begin{pmatrix} -p_{12} & p_{21} \\ p_{12} & -p_{21} \end{pmatrix} \begin{pmatrix} n_1 \\ n_2 \end{pmatrix} \tag{2}$$

In steady state, this must equal zero. We can now build up a many-state transition matrix by adding together all the two-state matrices. As long as the phase space densities are correct (*i.e.* we set $n_1, n_2, n_3...$ correctly), then every two-state matrix will give zero when we multiply by the phase space density, and so *every* possible sum of two-state matrices will give zero. As a more concrete example, let's take a three state system and explicitly write down all the transition matrices:

$$\left[\alpha_{12} \begin{pmatrix} -p_{12} & p_{21} & 0 \\ p_{12} & -p_{21} & 0 \\ 0 & 0 & 0 \end{pmatrix} + \alpha_{13} \begin{pmatrix} -p_{13} & 0 & p_{31} \\ 0 & 0 & 0 \\ p_{13} & 0 & -p_{31} \end{pmatrix} + \alpha_{23} \begin{pmatrix} 0 & 0 & 0 \\ 0 & -p_{23} & p_{32} \\ 0 & p_{23} & -p_{32} \end{pmatrix} \right] \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} \tag{3}$$

As long as Equation 1 holds for each pair of states, then this must be equal to zero for *any* values of α_{ij}. You can think of the α's as mixing rates between the states. Large values of α mean that particles are likely to bounce back and forth between those two states quickly, while small values of α mean particles will only slowly transition between the two states. Note also that there is nothing special about the number of states. In particular, we can reproduce continuous distributions by taking the limit as the number of states approaches infinity.

With these results in hand, we are now ready to make the key leap. We can simulate a system by starting in a state, picking a new state the system *might* jump to, and then deciding if the system actually makes the jump or not. By convention, if the phase space density is higher at the trial point, we always take the jump. If it is lower, we *sometimes* take the jump, with probability $\frac{n_i}{n_j}$. If we wait long enough, then the the path our system took in our simulation (called a Markov chain) will have its observed phase space density converge to the underlying average (ergodicity). We can then measure any quantity we desire by averaging over the path of the particle.

How we pick trial state j when starting at state i is called the trial distribution. Crucially, the distribution of states in the Markov chain doesn't rely on the trial distribution. A better trial distribution means our Markov chain converges *faster*, but it will still converge even with a poor trial distribution. The only formal constraint is that the probability for state i to attempt to transition to state j has to be the same for state j to transition to state i (and that the chance to transition is non-zero - obviously, we can't wall off sections of parameter space).

How do Markov chains relate to parameter fitting? Let's say we want to calculate the expectation of a model parameter. If we know how to calculate the likelihood of a given set of model parameters, we can write down the expectation of model parameter m_i:

$$\langle m_i \rangle = \int m_i \mathcal{L}(m) \mathrm{d}^n m$$

We could in principle just calculate this integral, but these sorts of integrals can rapidly become intractable. Let's say we have 10 parameters, and want to do a numerical integration going from -3σ to 3σ in steps of 0.2σ. In that case, each parameter can have one of 31 possible values, so to brute force this integral we would need to evaluate the likelihood at 31^{10} different points in parameer space. That works out to over 800 trillion points. If each likelihood took 1 microsecond,

we would need 25 years just to do this integral! Now let's say we had a Markov chain - we could take the expectation of m_i just by averaging the values of m_i in the chain. If the Markov chain converges at the level we care about in many fewer steps, then we can calculate parameter expectations, or indeed *any* statistical quantity, very quickly by averaging over samples in the chain.

This then is the power of using Markov chains to carry out Monte Carlo integrations over complicated likelihood surfaces (hence the name of this technique: Markov Chain Monte Carlo or MCMC). With a good trial distribution, MCMC will converge faster, but even with a poor trial distribution, it will eventually converge. The number of chain samples required to converge tends to grow only slowly with increasing numbers of parameters, and so MCMC is useful for high dimensional problems. They are particularly when the likelihood surface is non-Gaussian, since methods based on Taylor expansions can give wildly misleading results.

We can now write down the steps to carry out MCMC:
Step -1: Decide on a trial distribution. It must be symmetric, but otherwise can be (nearly) anything.
Step 0: Pick a random point in parameter space and calculate its likelihood. This is your current location.
Step 1: Take a trial step, drawn from the trial distribution.
Step 2: Evaluate the likelihood at the trial location.
Step 3: If the new likelihood is larger, move to the new location with probability 1. If the new likelihood is smaller, move to the new location with probability $\mathcal{L}_{new}/\mathcal{L}_{old}$ (you'll need to generate a random number for this).
Step 4: Save your current location, and go back to step 0.
Keep going until you're converged.

3. Convergence

We have repeatedly mentioned that Markov chains must be "long enough", or "converged". What does that mean, and how will we know when we are converged? The qualitative answer is that we are converged when the difference we expect between any quantitty we care about and the result from an infinitely long Markov chain is "small enough". Convergence criteria can vary wildly depending on which wuantities the user cares about and what errors they are willing to tolerate. Because a particle spends only a tiny fraction of its time in regions of very low likelihood, measuring the 5σ error on a parameter can take drastically longer than measuring its mean.

An important concept in convergence is the idea of an *independent sample*. If I know the state of my system at time t, I know that the state at time $t + dt$ must not have changed much if dt is small. For sufficiently large dt, knowing the state at time t tells you nothing about the state at $t + dt$. Loosely speaking, there is a minimum value of dt for which samples are independent, and convergence often comes down to knowing how many independent samples we have. Take the case of measuring the mean of a parameter to 0.1σ. The variance of each individual independent sample would be σ^2, and so we could get to 0.1σ uncertainty with 100 independent samples. In contrast, a Gaussian-distributed particle would spend about 10^{-6} of its time at 5σ or greater, so brute-force

we would need millions of independent samples to get enough 5σ entries in the Markov chain to say anything useful about the distribution there.

One way of checking convergence is through the Gelman-Rubin test. It relies on the fact that the scatter in the mean of a distribution is smaller than the scatter of individual realizations of that distribution. For a single Markov chain, the error in the mean of a paramter is roughly the per-sample error divided by the square root of the number of independent samples. This doesn't help us much if we only have one Markov chain, but take the case of having multiple, independent Markov chains. We can get an estimate of the error in the mean by comparing the means of the different chains. The number of independent samples is then something like the mean of the variances within chains divided by the variance of the differences between chains of their means. One usually runs several different chains (often in the range of 5-10) starting from different, random points, and stops when the scatter of the means across chains is smaller than some factor (often $\sim$ 0.01-0.02) times the average of the within-chain scatters. Gelman-Rubin says to periodically check this ratio and stop when it is small enough. It is generally wise to pick starting points for the chains with more scatter than the chains themselves have, to reduce the risk of believing you are converged when you're not.

The Gelman-Rubin test works well in the world where you have many chains. The question naturally arises: "Can we estimate convergence from a single chain?" Happily, the answer remains yes. If we have uncorrelated samples, then if we took the Fourier transform of the parameter values in those samples, we would see white noise (since the Fourier transform of white noise is white noise). If the samples are correlated, we would see that there is more noise on large scales (where the samples bounce around) than on small scales (since nearby samples are closely related). When we Fourier transform the parameters, we can square and smooth, and typically see a bend in the smoothed transform (referred to as a *power spectrum*). If you take the index of the power spectrum where the slope breaks, that indicates roughly how many independent samples you have (in fact it's slightly pessimistic, because there is *some* information in the correlated region, but you won't be too far off taking the break position). Examples of unconverged and converge Markov chains and their corresponding power spectra are showin in Figures 1 and 2.

4. Importance Sampling

For large problems, generating well-converged Markov chains can often be a pretty large computational task. Very often, you may want to make small changes to the likelihood (say from including a bit of extra data, or including prior knowledge of parameters). Starting from scratch to include the new information would be painful, so one might naturally ask if there's a way to handle this case without having to re-run a full chain? Happily, the answer is yes! If I have a likelihood function $\mathcal{L}(p)$ and want to adjust the likelihood to be some new function $\mathcal{L}'(p)$, then I know the ratio of the new phase-space density to the old one is just $\mathcal{L}'(p)/\mathcal{L}(p)$. If our likelihood has taken the form of χ^2, this is just $\exp(-\frac{1}{2}\delta\chi^2$. If I have a well-converged chain for
$mathcal L(p)$, then I can just adjust the phase-space density in that chain by $\exp(-\frac{1}{2}\delta\chi^2$, and so I can average over the new distribution simply by taking a weighted average over the old distribution. One needs to be careful when doing this, as if the parameters shift by too much (typically

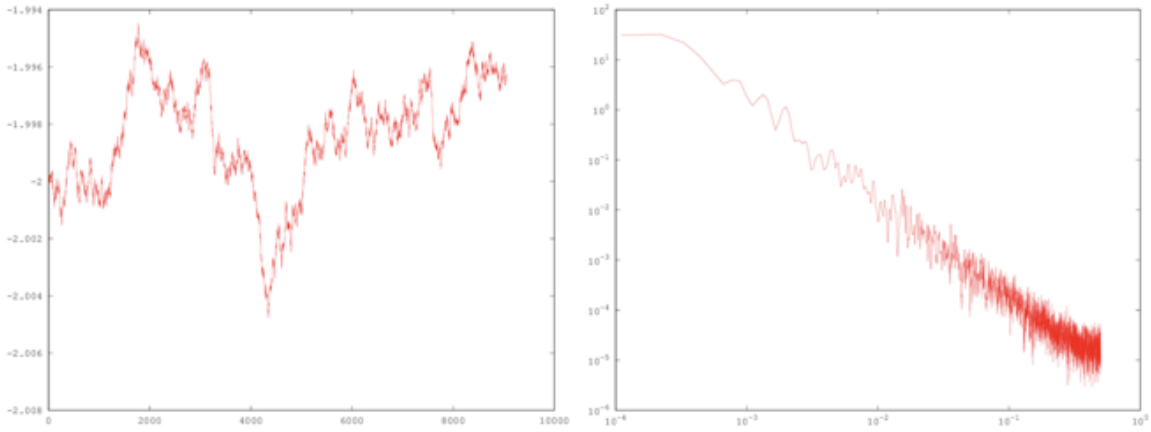

Fig. 1.— Left: An unconverged Markov chain. If someone bet you your life you knew what the long-term average of this was going to end up being, would you take that bet? Right: The power spectrum (smoothed Fourier transform) of the Markov chain in the left panel. Note that power keeps going up as you go to larger scales (move towards the left). There's no indication that the spectrum is levelling off.

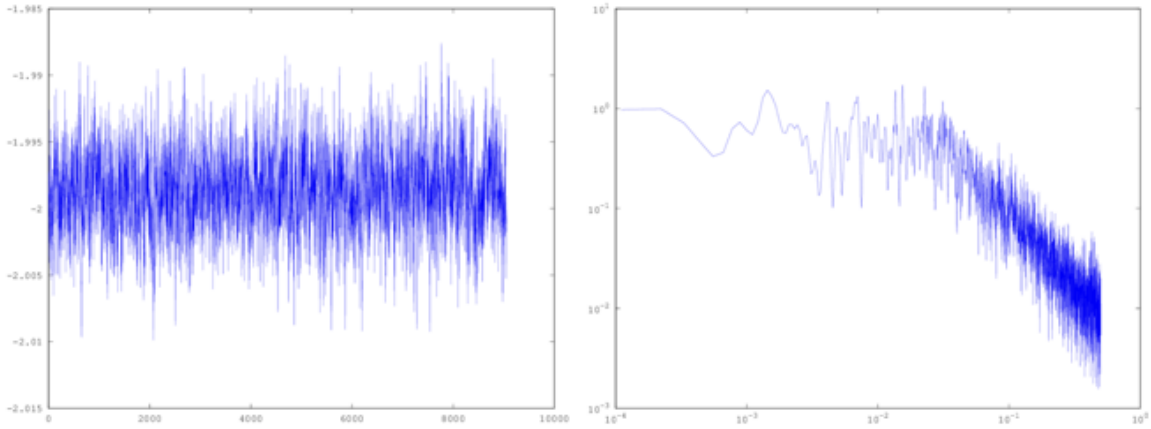

Fig. 2.— Left: An converged Markov chain. If someone bet you your life about the mean of this distribution, would you take it this time? Right: the power spectrum of this Markov chain. The power flattens off in the left half of the spectrum, indicating convergence. The x-axis has been scaled to go to 1, and the knee is at about 0.03, so we get an independent sample roughly every 30 entries in the chain. Since the chain is 9000 samples long (from the x-axis of the left panel), we have about $0.03 \times 9000 \sim 300$ independent samples. We should have good estimates of the $1-\sigma$ errors, reasonable estiamtes of the $2-\sigma$ errors, and nothing to say about $5-\sigma$ errors from this chain.

$1-\sigma$ or so), then the old chain will only poorly sample some regions that are important for $\mathcal{L}'(p)$. This technique is called *importance sampling*, and is very handy for quickly getting new parameter estimates/errors when only slightly changing a problem. Note that to do this, you'll need to keep track of the likelihood of each chain sample. You should be doing this anyways.

Notice that when we importance sample, the way it can break down is if the new parameter space has important regions that fall outside the old parameter space. If the new parameter space is a subset of the old space, then we are much less likely to have issues. We still may want longer chains, since we are in effect throwing away samples, but we aren't going to miss anything major. With this in mind, we can use importance sampling to get to those elusive $5-\sigma$ errors. Say we divide χ^2 by 5 and run a Markov chain. Regions that used to be pretty much disallowed are now going to be well sampled by this new chain. Since thermodynamically, dividing χ^2 by a factor is equivalent to increasing the temperature by the same factor, these chains are usually referred to as having high temperatures. After running a high-temperature chain, we can now importance sample it down to the actual temperature we want (for numerical overflow/underflow purposes, you might want to pick a new/old reference χ^2 and importance sample based on scaling the difference between the sample χ^2's and the reference value. While you'll probably want to run a longer-than-usual chain initially, this technique will get to large-σ error bars far more efficiently than trying to brute-force an unheated chain.