An Introduction to Markov-Chain Monte-Carlo

Markov-Chain Monte-Carlo (MCMC) refers to a suite of processes for simulating a posterior distribution based on a random (ie. monte-carlo) process. In other words, when fitting a model to some data, MCMC helps you determine the best fit as well as the uncertainty on that best fit.

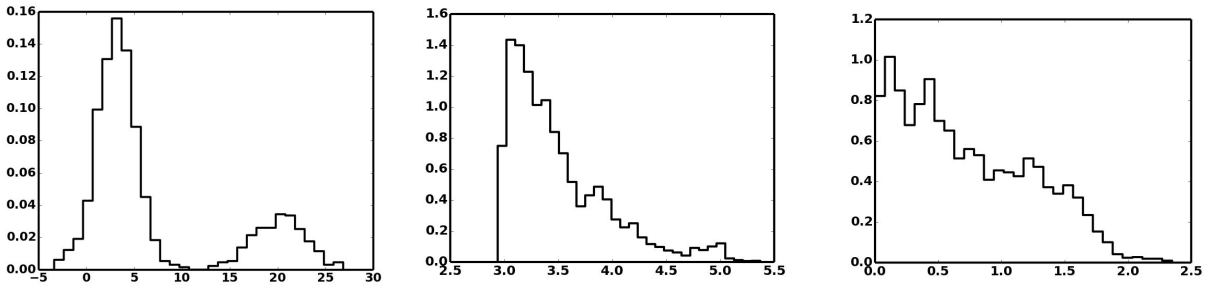
This article serves as a broad introduction to MCMC, covering the basic conceptual ideas while leaving aside much of the mathematical framework on which MCMC is built. More detailed descriptions of this framework, plus implementations in the literature, are included in the appendix.

What is a posterior distribution function?

First, we need to understand what exactly a posterior distribution function (PDF) is, and why we need to go to such lengths to measure it.

A PDF is a measure of the relative probability of different values of a model parameter. For example, suppose you are fitting a reddened blackbody to some photometry of a star and you find that the best fit effective temperature for the blackbody is $T_{eff} = 5000 \pm 200$ K. In the language of a PDF, what you are really saying is: 'The PDF of T_{eff} can be well approximated by a Gaussian shape with a mean of 5000 and a standard deviation of 200'.

While this shorthand is useful, it may not be reflective of what is actually happening. In other words, the PDF doesn't necessarily have to be Gaussian. It can have any of a wide range of possible shapes.



Examples of non-trivial posterior distributions. (Left) Two solutions, one less likely that the other, (Middle) A PDF with a long tail towards high values, (Right) A upper limit. In all three cases a simple estimate of the best fit, plus a standard deviation, would not be an accurate description of the acceptable models.

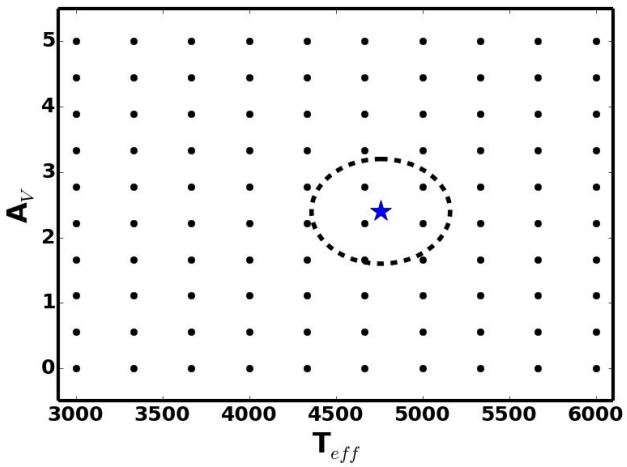
When examining the PDF for a single parameter, such as the examples shown in the figure above, you are looking at the marginalized PDF, because you have marginalized over the other parameters in the model. This may hide additional information if e.g. there is a strong degeneracy between two parameters. In this case a two dimensional PDF is necessary to completely convey information about the model.

In the end, regardless of its exact shape, it is important to understand, and report when necessary, the PDF for each parameter in your model. It is this motivation that in large part drives the use of MCMC.

The Process

If the goal is the determine the PDF for each model parameter, then we need a method that returns the PDF with minimal model evaluations (which can be expensive in terms of time) and with minimal fine-tuning. MCMC works to perform this process.

Consider again our example above of fitting photometry from a star with a reddened blackbody. Our model has two parameters T_{eff} and A_V. The simplest way to determine the best fit model is to lay down a grid over the parameter space.

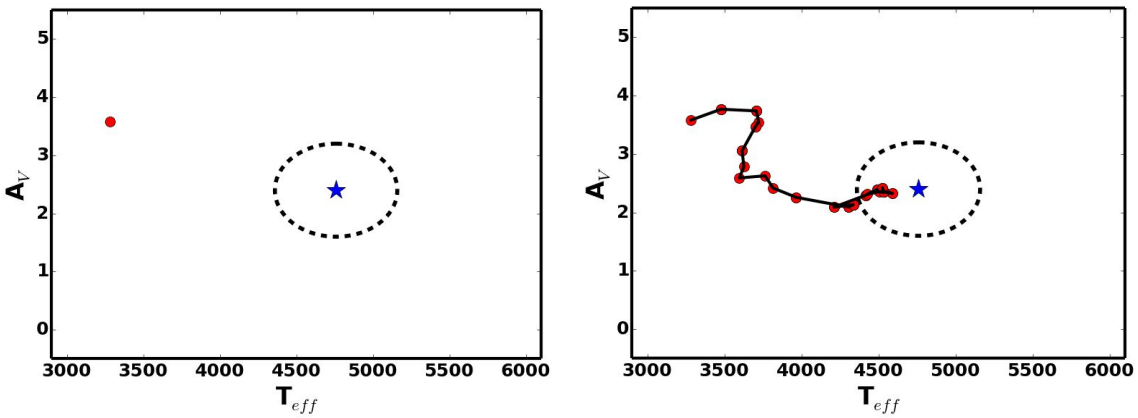


A grid search is conceptually simple, but becomes computationally expensive when the optimal grid spacing and a rough estimate of the best fit are unknown. The coarse spacing shown here will come close to the best fit (blue star) but will not be effective at sampling the 1σ range in the PDF (marked with a dashed circle). A finer grid could be used, but would also result in extraneous model calculations far from the best fit.

The challenge with this is that there is some fine-tuning required to figure out the appropriate grid spacing for each parameter. If your grid is too coarse, you will miss the best solution. If it is too fine your grid search will take a very long time to finish as it generates models far from the best fit. Also, remember that our goal is not simply to find the best fit but to determine the PDF (in the above figure you can think of the PDF as defining boundaries, of unknown shape, tracing the 1,2,3,..σ regions around the best fit). A very fine grid will help you determine the exact shape, although at great computational expense.

MCMC codes are a clever (and statistically valid) way to avoid these complications. Consider our parameter space again. Let us start by randomly selecting a point in parameter space and evaluating the likelihood of this model. Now propose a nearby point, and evaluate the likelihood at this new position. If this new position is a better fit to the data (ie. it has a higher likelihood) then the chain moves to that new position. If it is worse then the chain sometimes moves to the new position (the reason why this is ‘sometimes’ and not ‘never’ will be discussed

below). This process of proposing and moving (or not) is repeated over many, many iterations. This process causes the chain to move towards the best fit on a random walk, thus allowing us to find the best fit with very little prior knowledge, and without spending too much extra time evaluating models that are nowhere near the best fit.



MCMC, instead of covering a uniform grid in parameter space, starts with an initial guess (left) and 'random walks' its way down towards the best fit (right). It does this with a series of proposed new positions, defined relative to the current position, followed by acceptance or rejection of the new position based on the likelihood of the new model compared to the old model.

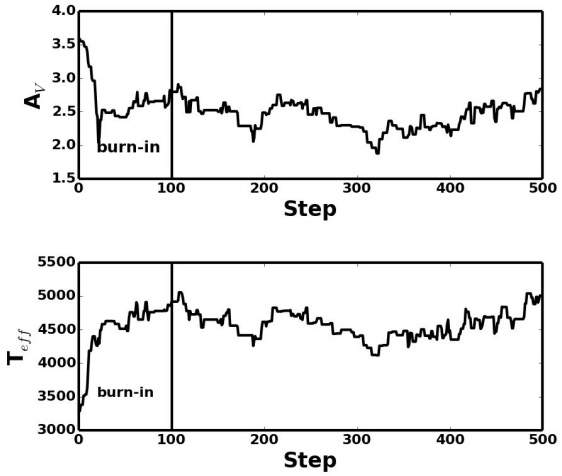
Now, why does the chain 'sometimes' move to a worse model instead of 'never'? Well, consider what would happen if the chain never moved towards a worse model. In this case the walker would quickly move toward the best fit, and park itself there. This is fine if we just want the best fit model, but remember we want the PDF, which means we want the relative probability of the models at those off-best positions. MCMC handles this by making the probability of moving to one of those off-best positions proportional to the relative likelihood of that position. So if we take the ratio of the likelihoods for the new and old position (L(new)/L(old))[1] then this gives the fraction of the time in which the walker will move to the new (worse) position. In practice this is done by drawing a random number from a uniform distribution between 0 and 1 and if this random number is less than the ratio of the likelihoods then the walker moves to the new position (this also works if the new position is better than the old one because in that case L(new)/L(old)>1, and our random number will always be less than 1, by definition).

That's it! The process of proposing and accepting/rejecting moves in parameter space is continued until the PDF is sufficiently well sampled. The first few steps when the chain is making its way towards the center of the PDF, are called 'burn-in' and are removed since they do not represent the final PDF. After burn-in the chain happily bounces around, randomly sampling the entire PDF. A simple histogram of the chain steps, normalized so that the area under the curve is equal to one, gives the marginalized PDF, while multi-dimensional PDFs can be generated based on the relative position for each parameter at steps along the chain.

---

[1] One of the simplest definitions of the likelihood is based on the chisq, $L(\text{model}) \propto \exp(-\chi^2/2)$ where $\chi^2 = \sum((\text{data}-\text{model})^2/\text{unc}^2)$

Determining exactly how to propose a new step is tricky. The simplest method, known as the Metropolis-Hastings algorithm, is to draw a new position from a Gaussian distribution centered at the old position. This does require some fine-tuning of the width of the Gaussian, which is not always simple. One prominent criteria for evaluating the efficiency of the jumping criteria is the acceptance fraction. Ideally your chain should accept new positions 20-50% of the time. If this is not the case then adjust the size of the proposal region and create a new chain. The only restriction on how long the chain is run is based on burn-in; the chain has to spend enough time in a post burn-in state to sufficiently sample your PDF. There is no simple tool to estimate how long this will take, outside of examining the movement along the chain.



During the initial steps of the chain the parameters are converging toward the acceptable area of parameter space. These initial steps are referred to as burn-in and are thrown out. Typically burn-in is extended beyond that shown in the figure to ensure that none of these initial moves are biasing the final distribution. Here the acceptance fraction, after burn-in, is ~30% which is within the acceptable range.

IMPORTANT: MCMC will find the best fit given the model that you have chosen, but whether or not the best fit parameters reflect reality depends on the quality of the model. In our example above, if a reddened blackbody is not an accurate model (e.g. the photometry is from an optically thin H_2 region, or there are absorption lines that lead to strong deviations from a blackbody) then our derived T_{eff} will not be reflective of the temperature of the system. MCMC, and Bayesian analysis in general, do not provide information on the overall quality of the model, only the relative quality of different model parameters. MCMC is happy to move along toward the best fit, regardless of whether or not that best fit (and the model in general) is meaningful. This can cause problems in interpreting your results, and should be a source of caution.

Priors:

Often we have additional information that can contribute to the constraint on the model parameters. These are called priors and can be easily included in the MCMC. All that is required is a simple modification of the likelihood function. Multiplying the likelihood by the relatively probability given by the prior ensures that this information is taken into account in the movement

of the chain. If the chain proposes a new position that runs contrary to the prior, then the likelihood of that new model will be downgraded appropriately, limiting the probability of the chain moving to this new position.

Two simple examples of priors are a flat prior, and a gaussian prior. For example, in our reddened blackbody model, we can impose a flat prior on extinction by requiring that it be positive (since negative extinction is not physical). We could also impose a gaussian prior on T_{eff} if, for example, we have a spectrum that allows us to independently constrain the effective temperature.

To accommodate a flat prior, such as a restriction to positive A_V, you can multiply the derived likelihood by 1 if A_V is positive and 0 if A_V is negative (since there is a 0 likelihood that A_V is negative). This will ensure that the chain never explores the undesired region of parameter space. A Gaussian prior can be accommodated by multiplying the likelihood for a particular model by $\exp(-(T_{eff}-T_0)^2/(2\sigma^2))$, where the prior restricts T_{eff} to $T_0 \pm \sigma$.

IMPORTANT: Be very careful when adding priors. Restrictive priors can artificially distort the PDF in a way that significantly biases the final results. In general it is best to start with very mild priors, and add additional information when necessary.

Other Resources:

http://pareto.uab.es/mcreel/IDEA2015/MCMC/mcmc.pdf : Very extensive and thorough lecture slides on MCMC.

http://nitro.biosci.arizona.edu/courses/EEB519A-2007/pdfs/Gibbs.pdf : Lecutre notes on MCMC

Ford 2005 ApJ, 129, 1706: Introduction to MCMC in the context of fitting RV curves of planets. Contains a useful list of steps for executing a MCMC program

Foreman-Mackey et al. 2013, PASP, 125, 306: Introduces a powerful MCMC modeling code, written entirely in Python. Useful if you need something stronger and more robust than the Metropolis-Hastings algorithm described here.