Ph 21.4b: Bayesian Data Analysis - Part II: MCMC

-v20130102-

Introduction: Markov Chain Monte Carlo (MCMC)

The previous assignment explored some basic concepts in Bayesian inference. Bayes' theorem was discussed and applied to find posterior probability distribution functions for some simple problems. The computations were done "brute force", i.e. the parameter space was simply gridded and the most probable value of the posterior probability and the width of the probability distribution were found "by inspection", i.e. by plotting the posterior probabilities on a grid of points.

The problem with the brute force approach is that it does not scale well to higher dimensionality. Consider a system with 8 parameters (not uncommon) and for which we wish a resolution in the grid of 2^{-8} for each dimension (also not unreasonable). The total number of points is then 2^{8^8}, i.e. 2^{64}, clearly prohibitive. This lack of scaling slowed the adoption of Bayesian inference techniques for a considerable time, until improvements in both algorithms and computing capability were sufficient to tackle many "real-world" problems.

There is a very large body of literature describing the computational algorithms that have been developed to solve high-dimensionality Bayesian inference problems. The basic problem is one of *sampling*: when brute force grid calculations become prohibitive, how do we sample the prior distribution efficiently in order to maximize our chance of finding the maxima in the posterior probability distribution and to calculate unbiased estimates in the error on the determination of the locations and width of the peaks in the distribution. We cannot do justice to the concept of efficient sampling in the brief space of this assignment. Rather, we will simply indicate relevant literature and the tools available for sampling. A useful reference is chapter 9 of *Data Analysis: A Bayesian Perspective*[3]. A copy is available in the Ph20,21,22 computation lab. In particular, chapter 9 of [3] gives explicit code (in C) for performing "nested sampling" Bayesian inference for the lighthouse problem discussed in the previous assignment. An optional part of this assignment is to implement the nested sampling code in python. Doing so will give you a much better feel for the machinery behind sampling strategies.

So what is MCMC? Markov Chain Monte Carlo is a certain approach to sampling which relies on a "chain" (time series) of samples for which the position of each sample of the chain relies on immediately preceding samples, with the length of the backward dependence being called the "order" of the Markov process. In common usage, the term "Markov chain" often refers simply to a first-order Markov process, i.e. a chain in which each step in the chain depends only on the previous step. In a Markov Chain Monte Carlo, successive samples are typically taken by moving a finite distance in a random direction in parameter space. Given long enough, the entire parameter space will be sampled uniformly (subject to certain rather non-restrictive assumptions about the dimensionality of the parameter space and the distribution of jump distances). Often, performing a "brute force" MCMC can take a tremendous amount of computing, particularly when the parameter space is large. As a consequence, clever algorithms have been developed for dynamically adjusting the jump distances depending on the relative values of the likelihood between the new and the old samples, running several chains with different starting points, etc. Chapter 9 of [3] discusses one such approach in detail, namely "nested sampling". In this assignment, you will use one of the standard packages for performing MCMC, and we will not concern ourselves too much with the exact implementation algorithm. If you want to know more about MCMC, there are several books on the subject, e.g. [1].

Two of the most widely used packages are *emcee* (based on [2]) and *dynesty*. The package *emcee* is the older and more widely used package, while *dynesty* is newer, is easy to use, and has good documentation.

The Assignment - Part I

Look up the web sites describing *emcee* and *dynesty* and choose one as the package you will use to perform the various parts of this assignment. If you are ambitious, you can try both. Redo the coin-tossing problem of the previous assignment, this time using MCMC sampling. Try various chain lengths and numbers of chains and compute how the posterior distributions evolves for several prior distributions: a flat prior and a prior with a gaussian distribution peaked at some 'bias' values. Vary the bias value and the width of the gaussian distribution in order to explore the system.

The Assignment - Part II

Redo the lighthouse problem of the previous assignment using MCMC for the case where neither the location of the lighthouse along the shore (x), nor the distance out to sea is known (y). Plot the 2-D posterior distribution for several choices of length of chain and numbers of chains. Compute the most probable position of the lighthouse separately in x and y as a function of the lengths of the chains for several choices of numbers of chains (i.e. find the chain with the most probable value among all the chains, and plot that). Also try the case of an "interloper", i.e. the case where a lighthouse ship comes nearby the first lighthouse and starts sending out its own light pulses. Will MCMC spot the interloper?

The Assignment - Part III - optional

Consider the case of the periodic system Her X-1 in the very first assignment of Ph22. Use MCMC to search a 3-dimensional space for periodic signals. The 3 dimensions are: amplitude of the periodic signal, period of the signal, and phase of the signal.

The Assignment - Part IV - optional

This part of the assignment is for those students who would like to get a deeper understanding of MCMC. Look up chapter 9 of [3] (the book is available in the Physics computation lab). Implement nested sampling for the lighthouse problem in python. Chapter 9 provides explicit C-code. If you do this optional assignment, you can translate the C-code into python, or get the general feel of the required code and write your own implementation in python.

References

[1] Gilks, W.R. and Richardson, S. and Spiegelhalter, D.J., *Markov chain Monte Carlo in practice* (1996).

[2] Goodman & Weare, Ensemble Samplers With Affine Invariance Comm. App. Math. Comp. Sci., Vol. 5 (2010), No. 1, 6580

[3] D.S. Sivia, *Data Analysis: A Bayesian Tutorial* (2006)

[4] W.H. Press et al., *Numerical Recipes*, various editions beginning in 1988.