Research Notes: Week 1

Maya Watanabe
Christina Catlett

May 18-22

This week, we followed C.A.L Bailer Jones's *Astrostats: Bayesian parameter estimation and model comparison* in order to parameterize a simple linear model with unknown noise using Markov Chain Monte Carlo Methods (MCMC).

1 Bayesian Parameter Estimation

Bayesian parameter estimation consists of inferring the posterior probability density function (PDF) over a model's possible parameters given the data, D, for some model, M. This PDF can be evaluated for a set of parameters, giving their probability. It is defined by Baye's Theorem as follows:

$$P(\theta|D,M) = \frac{P(D|\theta,M)P(\theta|M)}{P(D|M)}$$

where θ denotes a set of parameters, and M,D are defined as above. $P(D|\theta,M)$ is referred to as the *likelihood* and denotes the probability of the data given the current parameters and model. $P(\theta|M)$, or the *prior*, gives the probability of a set of parameters for a given model, regardless of the data, D. Finally, the denominator $P(D|M)$ is known as the *evidence*. As the evidence does not depend on the parameters, θ, it is simply a normalization constant for the posterior probability.

The first step in parameter estimation is to establish the model, or to select a likelihood function and prior that describe the event. The quality of the estimation relies on the choice of likelihood and prior, so most frequently, these choices are informed by *a posteriori* knowledge about the event. In general, however, the choice of the prior and likelihood often cause the resultant posterior ditribution to be of non-standard form, i.e. unable to be analytically modeled by a known distribution. Additionally, a large number of parameters within a model leads to a high-dimensional parameter space, and thus a high dimesional PDF. The best fit of parameters for the model would be located at the peak of this posterior PDF, as these are, by definition the parameters located where the probability is the highest.

This maximum likely cannot be found analytically, and searching for it directly poses a challenge. While sufficiently sampling a one-dimensional PDF requires N samples, sufficiently sampling a two-dimensional PDF requires building a grid of N^2 samples, and a J-dimensional PDF consequently requires the choice of N^j samples. This is known as the *curse of dimensionality*. Testing the posterior probability each of these N^j samples is computationally infeasible, and moreover, the probability at the majority of the these coordinates is minsicule, making a large part of the computatation time wasted. To overcome the curse of dimensionality, we instead look to *sample* the posterior PDF directly in a way that the distibution of samples is equivalent to that of the PDF after a large number of draws. Essentially, these samples simulate the distribution itself, allowing for information, such as the expected values of parameters, to be extracted from the simulation. One class of methods for selecting and evaluating these samples is known as Markov Chain Monte Carlo Methods, or MCMC.

2 Metropolis Markov Chain Monte Carlo Algorithm

The idea behind Metropolis MCMC sampling is to draw samples (of parameters) from an arbitrary probability density using random numbers drawn from a simpler distribution. MCMC uses a Markov Chain to perform

rejection sampling - that is, given a proposal distribution, we draw samples and penalize with an acceptance criteria (a ratio). MCMC establishes a random walk over the parameter space such that we can preferentially sample regions of high probability. Furthermore, each 'step' along this walk is independent from the last.

The Metropolis-Hastings MCMC is one of many kinds of MCMC algorithms. In order to implement this method we need:

- A proposal distribution (likelihood) to draw new proposed parameters
  - This is often assumed to be a multivariate Gaussian distribution
  - The best proposal distribution is one that gives a higher probability of picking nearby points rather than those far away
- The proposed stationary distribution of the Markov chain (prior)
  - Sometimes little is know about this distribution, and so we implement a prior that does not impact the posterior distribution very much (Jeffrey's prior)
- An initial guess of our parameters (θ_{init})
  - Time until convergence can depend on the quality of θ_{init}. To lessen the impact of a poorly guessed θ_{init} on the result of the MCMC, there are often a number of samples known as 'burn-in' that are run before true data is recorded. This allows the Markov Chain to reach reasonable values before data is collected.

Metropolis-Hastings MCMC algorithm:[1]

1. Choose initial values for each parameter, θ_{init}, and calculate its posterior probability
2. For each iteration of the Markov chain
   (a) Sample a random candidate of parameter values, θ_{prop}, from the proposed distribution (Gaussian likelihood) and calculate its posterior probability
   (b) Decide to accept or reject θ_{prop} based on a ratio of posterior probabilities
   - $\rho = \frac{posterior\ probability\ of\ \theta_{prop}}{posterior\ probability\ of\ \theta_{cur}}$
   - This is the ratio between the posterior probabilities of the proposed parameters and the parameters from the current step in the Markov chain
   - If $\rho \geq 1$ then we accept the transition (and θ_{prop} becomes our θ_{cur} for the next iteration)
   - If $\rho < 1$, then we only accept the transition with a probability of ρ. (Note: although moving to an area of lower probability is less likely, it can still be beneficial to our algorithm because the MCMC chain avoids getting stuck at local maxima.)
3. Repeat this process many times, i.e. take enough samples for your Markov chain to reach a steady state (in our example, we sample $10,000$ times)

Below is our MATLAB translated code for the Metropolis-Hastings MCMC algorithm:

MATLAB file: *metroMCMC.m*

```
% Authors: R Code by C.A.L. Bailer-Jones, translated to MATLAB by Christina Catlett & Maya Watanabe
% Date: 5/20/2020
% Description: R Code from Appendix C of 'Astrostats: Bayesian parameter estimation and
%              model comparison' translated to MATLAB

% Metropolis (MCMC) algorithm
```

[1]This algorithm was adapted from R Code from Appendix C of 'Astrostats: Bayesian parameter estimation and model comparison' by C.A.L. Bailer-Jones.

```
function [funcSamp] = metroMCMC(thetaInit, Nburnin, Nsamp, verbose, sampleCov, obsdata, Ndat)

    Ntheta=length(thetaInit); % number of parameters
    thetaCur=thetaInit; % initializing a start to the Markov chain using an initial theta
    funcCur=logpostlinearmodel(thetaCur,obsdata, Ndat); % returns the posterior probability
    %   of the current parameters (unnormalized)

    funcSamp=zeros(Nsamp,2+Ntheta); % initialize an empty matrix to store posterior pdfs and
    %   parameters later on
    nAccept=0; % # of accepted parameters
    acceptRate=0; % percentage of accepted thetas
    nb=Nburnin+Nsamp; % number of times to sample; burn-in is 0 in this tutorial so that we
    %   can see the Markov chain reach steady state

    % For each iteration:
    for n=1:nb
        % Metropolis alg. There is no Hastings factor for symmetric sampling
        %   distributions

        % Randomly sample a new set of proposed parameters
        a0Prop = mvnrnd(thetaCur, sampleCov);
        aProp = mvnrnd(thetaCur,sampleCov);
        sigProp = mvnrnd(thetaCur,sampleCov);
        thetaProp = [a0Prop(1) aProp(2) sigProp(3)]; % proposed parameters

        % Calculate the posterior probability of the proposed theta
        funcProp = logpostlinearmodel(thetaProp, obsdata, Ndat); % scalar

        % Calculate the rejection criteria difference (log10(rho))
        logMR = sum(funcProp) - sum(funcCur); % the ratio betw the posterior
        %   probabilities of new parameters vs. old parameters

        % Deciding to accept/reject proposed theta based on rho
        % if posterior probability of prop params > current params: prop params more likely than curren
        if logMR >= 0 || logMR > log10(unifrnd(0,1,1))

        % logMR>= 0: accept if the thetaProp is more likely/posterior probability is
        %   greater than the thetaCur (old theta)
        % OR
        % logMR<0: draw from a uniform probability and accept with
        %   probability logMR

            thetaCur=thetaProp; % proposed theta becomes the new theta
            funcCur=funcProp; % proposed posterior becomes the new posterior
        end

        % Store the posterior probabilities and parameter values for each sampling
        if n>Nburnin
           funcSamp(n-Nburnin, 1:2)=funcCur; % fills in the empty matrix with the
           %   posteriors from each sample
           funcSamp(n-Nburnin, 3:(2+Ntheta))=thetaCur; % filling in the 'found'
           %   parameters for each sample
        end
```

```
    end
end
```

Our MATLAB code containing the function 'logpostlinearmodel' (and the functions there in) can be found at `https://github.com/shtyllab/2020-HMC-REU-Codes/tree/master/subteam1`.

For N samples and p parameters, this algorithm produces a N-by-$p+2$ matrix which stores the accepted parameter values for each sample and the corresponding evaluated likelihoods. This data can then be used to find the probability density distributions of each individual parameters (we use a kernel smoothing function estimate for multivariate data).

3 Tutorial: Fitting a linear model with unknown noise

In this tutorial, we want to fit a model, in this case a linear model, to some generated two-dimensional data. In other words, we want to find the line of best fit for the data.

We have a set of data $D = [x_r, y_r]$ with R data points ($D_r = (x_r, y_r)$ is one data point). Next we set up our linear model M which predicts the values of y as being:

$$y = f(x) + \epsilon$$
$$f(x) = a_0 + a_1 x$$

Where a_0 is the intercept, a_1 is the slope of our model, and ϵ is an assumed Gaussian random variable of noise with a mean of 0 and a standard deviation of σ. Our parameters, θ are (a_0, a_1, σ). Because ϵ is $N(0, \sigma)$, the residuals (the observed value of y − the predicted value of y) is modeled as $\epsilon = y - f = N(0, \sigma$, our likelihood function becomes:

$$P(D_r|\theta, M) = \frac{1}{\sigma\sqrt{2\pi}} exp[-\frac{[y_r - f(x_r; a_0, a_1)]^2}{2\sigma^2}]$$

Since ϵ is assumed to be normal and given that $\epsilon = y - f$, we can write distribution of ϵ as above and thus obtain the Gaussian likelihood function. *Note: In order to make calculations easier, we take the log of many of the functions and values so we can deal with sums rather than products.*

The goal now is to infer the posterior pdf of the parameters:

1. Define a prior pdf over the parameters (this is the distribution we believe our data to be drawn from)
2. Define a covariance matrix of the proposal distribution (Gaussian)
3. Define initial parameters as the starting point of the MCMC (θ_{init})
4. Run the MCMC to obtain the posterior pdfs over the parameters
5. Compute the estimated density function of each parameter using the posterior pdfs
6. Calculate the mean and MAP (Maximum a posteriori: parameter values at the maximum value of the posterior pdf) of the density functions as a "solution" to our line of best fit

Figures 1 and 2 show the iterations of the MCMC algorithm and the computed densities of each parameter.

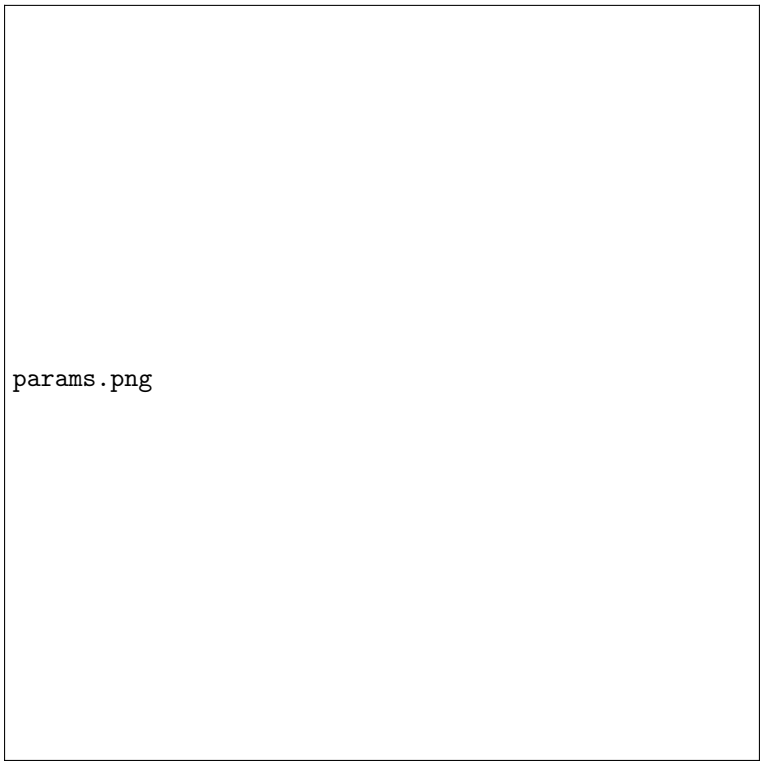

Figure 1: The Markov chains for the 3 parameters

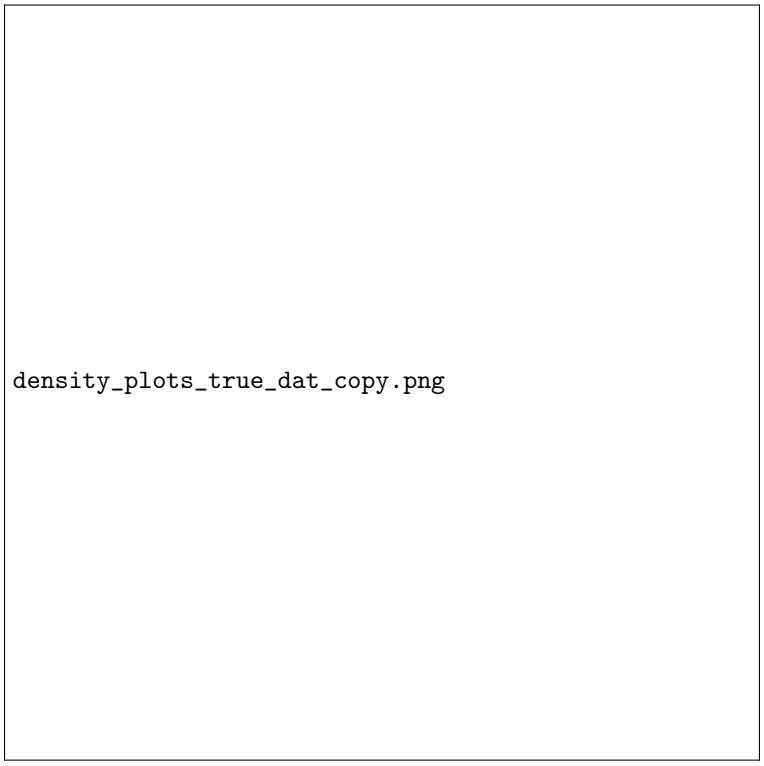

Figure 2: The pdfs generated from the MCMC samples via kernel density estimation for each parameter. Vertical lines indicate the true parameter values

4 Next Steps

Running our current main file, *astrostats.m*, only runs the MCMC algorithm once, determining a single set of ideal parameters for the linear model. To encapsulate more fully the behavior of the algorithm and produce a more sound prediction of the ideal parameters, we hope to adjust out current algorithm to run the MCMC calculations a large number of times, generating a reasonable range of lines of best fit to plot. This will better visualize the spread of possible outcomes for the MCMC chain, and allow for a mean line of best fit to be calculated.

The Monte Carlo simulation produced by the MCMC algorithm can also be used to predict the probability of new data added to the model. Given the posterior PDF and a new value of the independent variable, x_{new}, candidate values of y, y_{cand}, can be tested in the model by integrating the likelihood over the posterior. This is denoted as $PDF(y_{cand}|x_{new}, obsdata)$. This could be used both to evaluate the probability of seeing an xy-pair according to the PDF, or to predict the most likely y-value for an x.

Finally, as using MCMCs for Bayesian parameter estimation is a universal approach for fitting, we plan to read more about the use of MCMCs for models beyond the simple linear model. This summer's overarching task is to work towards parameterizing the Type 1 diabetes model, so next week we will begin looking at the use of MCMCs in parameterizing nonlinear ODE models of tumor growth in *Collins, 2017*. As an important expansion on what was covered in the Astrostats tutorial, Collins considers ways not only to use MCMC to parameterize, but also to validate the model and quantify uncertainty.