Research Notes: Week 2

Maya Watanabe
Christina Catlett

May 25-29

1 Reading

This week, we read the paper "MCMC Techniques for Parameter Estimation of ODE Based Models in Systems Biology". MCMC techniques, such as the Metropolis-Hastings algorithm discussed last week, can be useful tools in parameter estimation for non-linear systems of ordinary differential equations. However, if just using the simple MCMC algorithm on its own, it is difficult to optimize the results; estimations often become stuck at non-global maxima in the posterior distribution because of the limited search space of the Markov chain. More complex algorithms exist to better and more widely sample the posterior distribution, making the selection of a globally maximum parameters more likely. "MCMC Techniques for Parameter Estimation" examined three of these algorithms: Single-chain Adaptive, Parallel Tempering, and Parallel Adaptive. The different algorithms were evaluated based on convergence speed, Estimated Sample Size (ESS), and probability of convergence to true parameters under both informative and uninformative priors. Notes on the specific conclusions of the paper can be found *here*.

From this paper, we were especially interested in understanding the algorithms themselves (Section 2), as well as in the estimation of informative priors from a dataset (Section 3). With this understanding, we moved to try them out for ourselves on a Lotka-Volterra model with observed data (Section 4). This is an ongoing project that we will take into next week.

2 MCMC Algorithms

2.1 Metropolis-Hastings

Original and simplest MCMC algorithm. Used to sequentially sample parameter space for candidate parameters, accepting or rejecting the candidates based on the evaluation of the posterior distribution. Can be univariate or multivariate.

- Requires the definition of priors for parameters, likelihood function (proposal distribution), and initial guess for the parameters, and, for multivariate models, a covariance matrix.
- Algorithm:
    1. Choose initial values for the parameter set θ_{init}, and calculate its posterior probability according to Baye's Theorem.
    2. Independently sample a random candidate value for each parameter in θ_{prop} from the proposal distribution.
    3. Compute the posterior probability of θ_{prop}, again according to Baye's Theorem.
    4. Accept or reject θ_{prop} as the new 'best guess' for the parameters according to the acceptance criteria, replacing θ_{curr}
    5. Repeat a large number of times, eventually reaching convergence

- Acceptance Criteria: θ_{prop} is accepted if it produces a higher posterior probability than θ_{curr}, meaning that it is more likely for the model's parameters to be θ_{prop} than θ_{curr}. Additionally, to avoid converging at non-global maxima, candidate θ_{prop}'s which are not found to be more likely than θ_{curr} can still be accepted with a probability proportional to the difference between the proposed and current parameter sets. Formalized,

$$\rho = \frac{P(\theta_{prop})}{P(\theta_{curr})}$$

  where $P(\theta)$ represents the evaluation of the posterior for parameters θ

  - If $\rho \geq 1$, $\theta_{curr} = \theta_{prop}$
  - If $\rho < 1$, $\theta_{curr} = \theta_{prop}$ with a probability of ρ

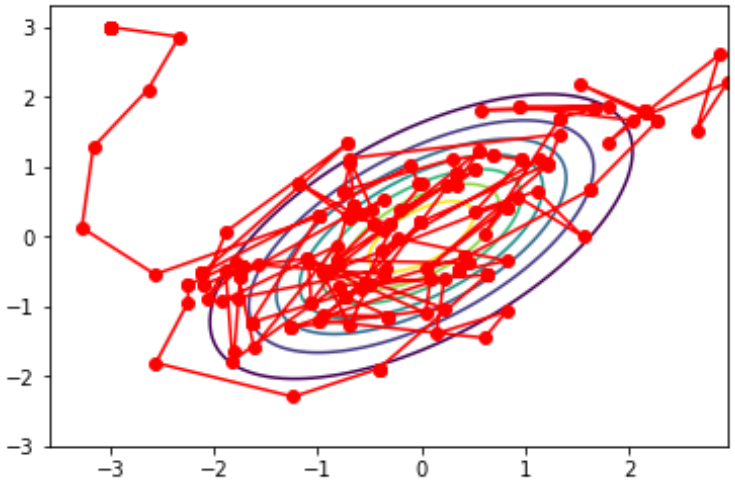

Figure 1: Single-chain Metropolis-Hastings MCMC. It is more likely for this algorithm to sample locally in regions of high probability and not move away.

2.2 Adaptive

The basis of this Adaptive MCMC algorithm is the Metropolis-Hastings algorithm but the proposal distribution is "tuned" (or updated) along the search according to the covariance calculated from a fixed number of previous states).

- Initiation of this algorithm requires the same definitions required by the MH MCMC algorithm: parameter priors, likelihood functions, a starting parameter set, and a covariance matrix (for multivariate distributions.
- We can also think of the adaptive algorithm as an MH algorithm where the proposald distribution depends on time
- Algorithm:
  1. Let q be the initial proposal distribution and assume that parameter sets $X_1, ..., X_k$ have been sampled
  2. Candidate set Y is sampled from the proposal distribution $q_k s$
     - Y has been sampled from a distribution that depends on the history of the chain
  3. Accept Y with a probability $\alpha(X_k, Y) = min(1, \frac{\pi(Y)}{\pi(X_k)})$

- Where $\pi(\cdot)$ is the unscaled probability density of the target distribution (posterior)

- Acceptance Criteria: Y is accepted with probability 1 if the posterior distribution for Y is greater than the posterior for the current parameter set X_k: $\alpha(X_k, Y) = min(1, \frac{\pi(Y)}{\pi(X_k)})$. It is important to remember that this posterior (π) is continuously updated based on the covariance matrix computed from the previous samples.

- How does the algorithm update and how does the proposal distribution depend on the "history" of sampling?
  - Starting at time 0, we sample using the MH algorithm
  - Some time later at time t, we have sampled at least H sets $\{X_1, ..., X_{t-H+1}, ..., X_t\}$
  - We then define the proposal distribution at time t, q_t for sampling proposal state Y as: $q_t(\cdot|X_1, ..., X_t) \sim N(X_t, c_d^2 R_t)$
    - R_t: $d \times d$ covariance matrix determined by sets $X_{t-H+1}, ..., X_t$ (i.e. all the sets before Y)
    - c_d: a scaling factor
  - The update occurs when the new covariance matrix is calculated using the previously sampled data

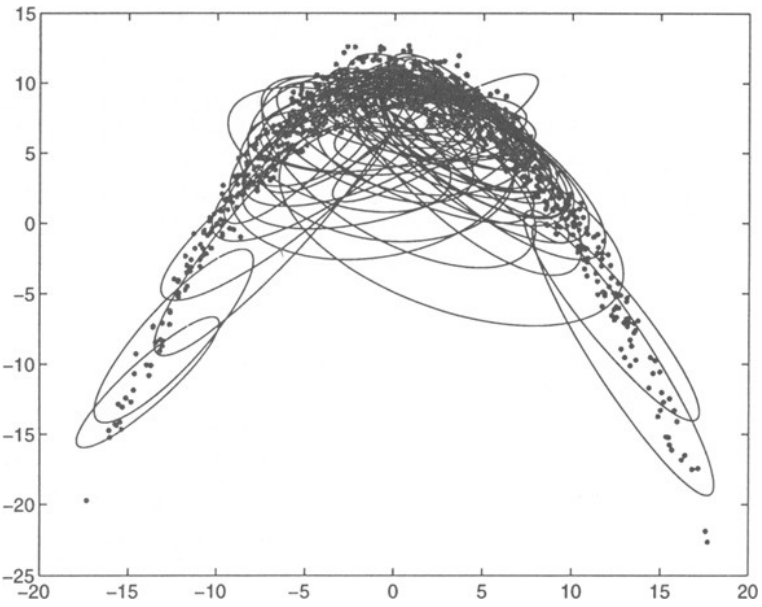

Figure 2: Single-chain adaptive MCMC. This algorithm samples the parameter space based on what has already been sampled. The likelihood function and subsequently the acceptance ratio is updated after specified time of sampling. This allows the chain to "move" around in the parameter space and sample different regions.

2.3 Parallel Tempering

In its essence, parallel tempering consists of running many MH chains synchronously, allowing for swaps between the chains. This is beneficial because it becomes more likely that a larger portion of the potential parameter space is explored in a reasonable amount of time, in comparison to MH. This is especially beneficial in a distribution where areas of high probability are seperated by areas of low probability, making it unlikely that a traditional MH chain would explore all high-probability areas.

The idea of "tempering" comes from the idea that a "temperature" parameter can be used to flatten a distribution. As the temperatures rises, more parameter space is explored and the distribution flattens,

making the likely random walk performed by MH span more of the parameter space. The idea of parallelism enters when a number of MCMC chains, k, are run simultaneously at different temperatures (degrees of flattening) given by $p(x_k)$ where x_k represents the values of the chain. The chains of different temperatures are allowed to "mix" by swapping entries with a swap probability, leading to wider exploration, and potentially a more likely final parameter estimate.

- Because parallel tempering uses the MH algorithm to run its replica chains (chains at different temperatures), it relies on having the same initial information as MH. It additionally requires a set of temperatures of length M, t_k, and a defined number of sweeps, N_{sweep}. Each "sweep" of the model consists of running the chains for N_{iter} iterations and then making swaps.

- Algorithm:

  1. For each sweep, $i = 1, ..., N_{sweep}$
     (a) For each temperature, compute N_iter iterations of an MH MCMC chain, adjusting the likelihood function, $q(x_{prop}|x_{curr})$ to consider the temperature:
     $$q(x_{prop}|x_{curr}) = x_{curr} + N(0, I/t_k$$
     (b) For each chain $k = 1, ..., M-1$, swap the elements x_k^i and x_{k+1}^i according to the swap criteria; that is, switch the parallel elements in the chain at the current temperature and the temperature directly above according to the swap criteria.

- Acceptance Criteria: Same as MH algorithm (Section 2.1)

- Swap criteria: The algorithm intends to keep the "warmest" temperature chain after each sweep. This means that if performing the swap would increase the temperature, the swap will be performed. Likewise, the swap would be unlikely to occur if the temperature is lowered. This is formalized by accepting the swap with a probability α_k:

  $$\alpha = \min(1, \frac{p_k(x_{k+1})p_{k+1}(x_k)}{p_k(x_k)p_{k+1}(x_{k+1})}$$

  In the above expression, the numerator represents the swap: evaluating x_k in the temperature function for the chain $k+1$, and vice versa. The denominator represents the current temperature: evaluating x_k in p_k and x_{k+1} in p_{k+1}. If the newly proposed temperature is warmer, the fraction will have a value > 1, and the swap will occur with a probability of 1. If not, the swap will be performed with a probability of α_k.

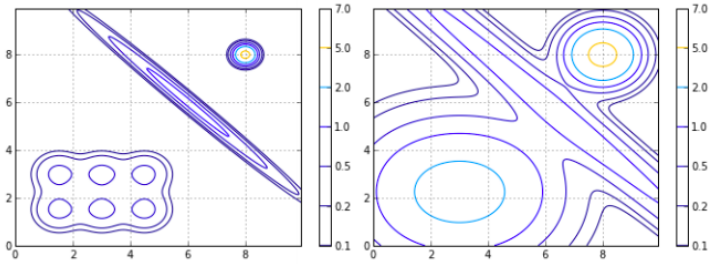

Figure 3: Parallel Tempering MCMC. Because each chain is sampling under a different temperature and then swapping chains of samples with other chains, the MCMC is able to explore areas of lower probability. Thus, it is less likely for chains to get stuck in local maximas. (a) shows an example distribution with 3 local maxima that would be difficult to sample with only a single MH chain. (b) shows the larger sampling space of running an MH chain at a different temperature. Running multiple chains simultaneously simply adds swapping between chains of different temperatures.

2.4 Parallel Adaptive

The basis of parallel adaptive MCMC (aka inter-chain adaptation (INCA)) is to use the algorithm of the adaptive single-chain MCMC method and apply it to multiple chains. The concept of parallel chains is used to detect different regions of parameter space with high probability under the posterior distribution.

- The very foundation of parallel adaptive methods is run using the MH method. Thus, these methods require the same initial conditions.
- Parallel Adaptive Algorithm:
  1. Run K different chains in parallel
     - Each chain is started independently from the same over-dispersed starting distribution (this is the initial guess for the likelihood function but will change in the future)
  2. After a burn-in period, the K kernels are simultaneously adapted using *all the samples* provided by the K chains so fair
     - This is where the adaptive algorithm is borrowed from the single-chain method above as the sampling distribution is updated based on the samples already taken
     - If our MCMC method is random walk Metropolis with Gaussian proposals, updating kernels are equivalent to setting the proposal covariance matrix to the sample covariance matrix of all the available samples
     - Note that within this step, the step of accepting and rejecting the parameter set also takes place
  3. Stop the swapping of information between chains when the chains no longer each contribute different information about the target distribution
     - Do this by using the *potential scale reduction* R (Brooks-Gelman-Rubin R), i.e. a convergence factor
- Acceptance Criteria: Same as Adaptive algorithm (Section 2.2)
- Swap Criteria: Same as Parallel Tempering (Section 2.3)
- Adaptive Parallel Tempered Algorithm: The idea is to apply adaptive and tempering methods to multiple chains.

  Start with $T = t_{max}$ and at each temperature and at each temperature:

  1. For $T = t_j$ perform parallel adaptive algorithm for a target density until convergence
  2. Keep the simulation parameters obtained (after rejection/acceptance that occurs within parallel adaptive algorithm) and repeat 1) with the next colder temperature $T = t_{j-1}$. Stop after $T = 1$

  - We assume that the kernel (or covariance matrix) adapted/updated at temperature t_j is a reasonable starting choice for the kernel used at temperature t_{j-1}
  - The goal of adaptive parallel tempering is to produce a reasonable starting proposal in a high dimensional problem (something usually difficult to do)
  - Adaptive parallel tempering significantly reduces the burn-in period (compared to the parallel adaptive method).
- Acceptance Criteria: Same as Adaptive algorithm (Section 1.2)
- Swap Criteria: Same as Parallel Tempering (Section 2.1)

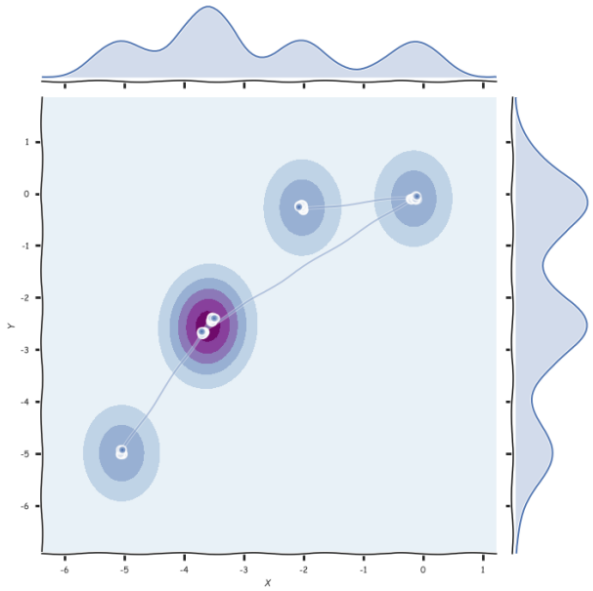

Figure 4: Adaptive Parallel Tempering MCMC. Because each chain is sampling under a different temperature as well as adjusting the likelihood and acceptance ratio via the adaptive MCMC algorithm and then swapping chains of samples with other chains, the MCMC is able to explore areas of lower probability. Thus, it is less likely for chains to get stuck in local maximas.

3 How to estimate a prior distribution

In MCMC, the prior distribution provides information that helps drive the sampled posteriors toward a final distribution. Thus, one important part of setting up to run an MCMC algorithm is determining your *a priori* prior distribution.
Recall Bayes' Theorem: $P(\theta|D,M) = \frac{P(D|\theta,M)P(\theta|M)}{P(D|M)}$

- θ: parameter set
- D: observed data
- M: the model
- $P(D|\theta,M$: likelihood function
- $P(\theta|M)$: prior distribution
- $P(\theta|D,M)$: posterior distribution

We can think about the *likelihood function* (it is also called the proposal distribution) as the way sets of parameters are related to each other and thus the way in which we want to sample candidate parameter sets. Gaussian noise informs us of the way different solutions of our model are distributed and thus informs the way different parameter values are distributed (in relation to one another). The *prior distribution* is the initial, general "guess" for the distribution that we think our parameter values fall under. As we sample during each step of MCMC, we accept or reject a candidate set of parameter values and thus "carve" away until we reach the true distribution of our parameters.

When we know very little about our parameter space, we will choose an *uninformative* prior which is a uniformly distributed prior. If we know more about our parameter space (via existing data), we can determine a more specific, *informative* prior that will affect the posterior probabilities calculated in MCMC. The paper we read (Section 1) provides their R code for determining an informative prior by extracting data from the BRENDA database (an enormous enzyme repository). The following is the steps that they took to compute an informative prior:

1. Load and format the database from which your data is coming from

- BRENDA is a huge text file that they must turn into a matrix in order to extract any data

2. Filter and collect relevant data
    - BRENDA has certain key terms that can be used to filter the data
    - They also sort the relevant parameters that they are looking at
3. Determine a range for the parameters k
4. Once all the data is collected, formatted, and filtered, they apply the R function *fitdistr* which, using the data points, is a maximum-likelihood fitting of univariate distributions (they are fitting the parameter distributions one-by-one)
    - Here they are choosing to fit a normal distribution and then taking the log of it after
    - *fitdistr* outputs a list containing the parameter estimates, the estimated standard errors, the estimated variance-covariance matrix, and the log-likelihood
    - An equivalent function exists in MATLAB: *fitdist* (although it looks like we would have to figure out how to extract the log-likelihood from this function)
    - fitdist documentation: `https://www.mathworks.com/help/stats/fitdist.html`

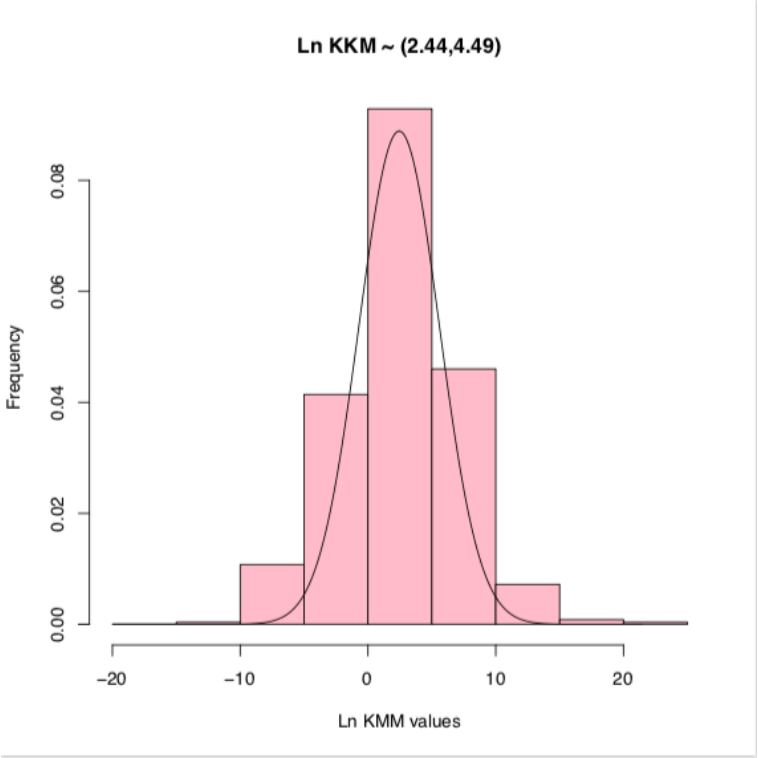

Figure 5: Example of the output of the R code to compute informative priors using the BRENDA database. The pink histogram illustrates the frequencies of the data that they extracted. The black curve indicates the density function of the data.

4 Fitting a Lotka-Volterra Model

In previous work, we had successfully used the MH algorithm to parameterize a linear model with added noise. Fitting an ODE model follows similar logic, however the aspect of time must be considered. As our

end-of-summer goal is, of course, working on the T1D ODE model, we began working with a simpler system as a means of testing out the algorithms discussed in our reading.

The model we are using is a simple Lotka-Volterra model, which is a system of two non-linear ODEs that describe the related growth of predator and prey populations:

$$\frac{dH}{dt} = \alpha H - \beta HL,\ \frac{dL}{dt} = -\gamma L + \delta HL$$

where $L(t)$ gives the density of the predator population at time t, and $H(t)$ gives the density of the prey population at time t. α is the intrinsic rate of prey population increase, β the rate of predation, γ the natural mortality rate of the predator population, and δ the reproduction rate of predator per prey.

Given a sample data set of lynx (predator) and hare (prey) populations over the years 1845-1935, we can use MCMC methods to determine likely values for parameters $\alpha, \beta, \gamma, \delta$. We start by fitting the data with basic MH MCMC:

1. As is detailed in Section 2, the MH algorithm requires the definition of an initial parameter set, priors for the parameters, θ_{init}, a likelihood function, and a covariance matrix.
   - The value of $\theta_{init} = [\alpha, \beta, \gamma, \delta]$ was given: $[0.7, 0.1, 0.7, 0.1]$
   - The covariance matrix was defined using the makeCovMatrix.m file written during the Astrostats tutorial (see Week 1's notes).
   - The likelihood is assumed to be Gaussian, consistent with the logic in the Astrostats tutorial:

     $$P(D|\theta) = \frac{1}{\sigma\sqrt{2\pi}} exp[-\frac{[y_r - f(x_r;\alpha,\beta,\gamma,\delta)]^2}{2\sigma^2}]$$

     where D represents the observed data.
2. According to MH, candidate parameters were chosen based on the likelihood function and uninformative priors: $P \sim Uniform(10^{-5}, 1)$
   - This range was selected because values for α, β, γ, and δ must be positive and < 1.
3. To evaluate the posterior distribution and determine the acceptance or rejection of the candidate parameters, the system of ODEs is solved at the same discrete time steps as are given in the dataset using a built-in MATLAB ODE solver.
4. The ODE solution produced is evaluated in comparison to observed data according to the MH algorithm, being accepted or rejected
5. These steps were repeated 10000 times for 10000 iterations in the Markov chain, resulting in estimates for both the mean parameter set and the Maximum A Posteriori (highest probability) parameter set discovered, as well as estimated distributions for α, β, γ, δ.

While the program we have written runs, we are still working to debug its outputs, so the figures produced are not yet accurate. Once we complete our program to run the MH algorithm, we plan on working through the same example using the other three algorithms in Section 2.

5 Moving Forward

After implementing adaptive and parallel MCMC methods for the Lotka-Volterra model, we will look at how the uncertainty of these models can be evaluated and quantified by reading 'Bayesian calibration, validation and uncertainty quantification for predictive modelling of tumour growth: a tutorial' by Joe Collis, 2017. Finally, in moving towards using MCMC to parameterize the T1D model, we will examine a new MCMC model known as *delayed rejection*.