Research Notes: Week 3

Maya Watanabe
Christina Catlett

June 1-5

The Lotka-Volterra model is a well-known model of predator-prey dynamics. The system of two nonlinear ordinary differential equations, where H denotes the prey population density, and L the predator,

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

considers the intrinsic growth rates of each species (α and β, respectively) as well as interactions between the two populations: γ being the predation rate, and δ being the reproduction rate of predators per prey eaten.

This week, our goal was to use our general knowledge of Markov chain Monte-Carlo (MCMC) algorithms to parameterize a Lotka-Volterra system from a dataset. The data we were given comes from the Hudson Bay Company's yearly census of snowshoe hare and Canadian Lynx populations from the years 1845-1935. This is a fundamental data set used to model predator-prey relationships: To attempt to fit parameters α,β,γ,

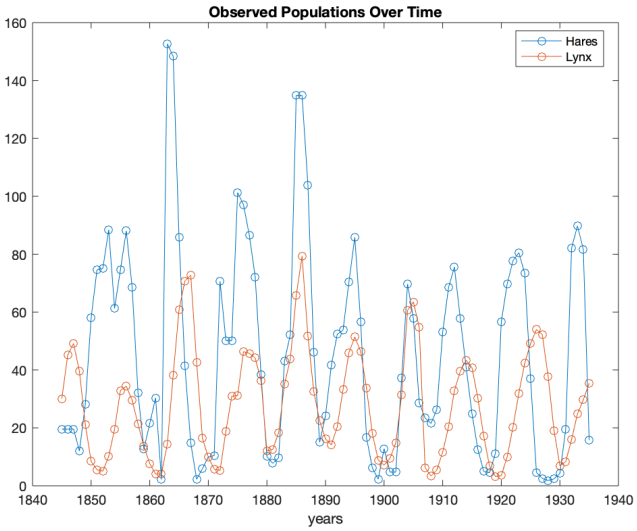

Figure 1: Observed data for snowshoe hare and Canadian Lynx populations from 1845-1935

and δ to this dataset, we implemented three different versions of MCMC using two different algorithms:

1. Metropolis-Hastings
   (a) MATLAB's *mhsample*
   (b) Code adapted from Chapter 8 of Ralph C. Smith's 'Uncertainty Quantification', 2014

2. Delayed Rejection Adaptive Metropolis (DRAM)

   (a) Code adapted from an example by Marko Laine, 2018

1 Fitting with Metropolis Hastings

The Metropolis-Hastings (MH) Algorithm has been well-documented in previous notes. See Week 1 and Week 2 summaries for an explanation.

1.1 Matlab *mhsample*

The full documentation for the built-in MATLAB function *mhsample* can be found **here**. Briefly, *mhsample* requires the user to determine the same initial set-up as we implemented in our tutorial i.e. an initial guess for parameter values, number of samples in the chain and burn-in. In addition, the user must implement their own unique proposal, likelihood, and prior functions, either via other built-in MATLAB functions or their own explicit definitions (we used a combination of both).

In order to get an understanding of how to define our distributions, we looked at an example **here**. Before our explanation of running *mhsample* for the Lotka-Volterra model we note some potential concerns:

1. Our implementation is adapted from an example that is *not* parameterizing an ODE model. Despite our efforts to modify the seed code, we recognize that there may not be a perfect fit in the results, especially as we did not alter the proposal distribution.
2. There do not appear to be many papers that use *mhsample* to parameterize ODE models for a biological system. This tells us that *mhsample* may not be the most powerful MCMC method for an ODE model.

The initialization process for running *mhsample* is as follows:

1. Load data
2. Initialize constants
   - Number of parameters: n = 4
   - Initial parameter values: thetaInit = [0.7 0.1 0.7 0.1]
   - Number of samples in chain: nsamp = 20,000
   - Burn-in length: K = 1,000
   - (Optional) Thinning interval: M = 10 (collect every 10^{th} sample)
3. Implement distributions
   - Proposal distribution: logproppdf = log-normal
     - proprnd: A random number generator that draws from the proposal distribution is also defined here
   - Likelihood function: log-normal
   - Prior distribution: uniform
     - logpdf: the likelihood and prior functions are added (due to log) to create the prior pdf

Once all the above constants and distributions have been defined, it is straightforward to plug them into the *mhsample* function, run, and plot the results. We produced the following results (Figures 2, 3).

```
% ===================================================================
% MH-sampling routine
% ===================================================================
[result,accept] = mhsample(thetaInit,nsamp,'logpdf',logpdf,...
    'logproppdf',logproppdf,'proprnd',proprnd,'burnin',K,'thin',M);
```

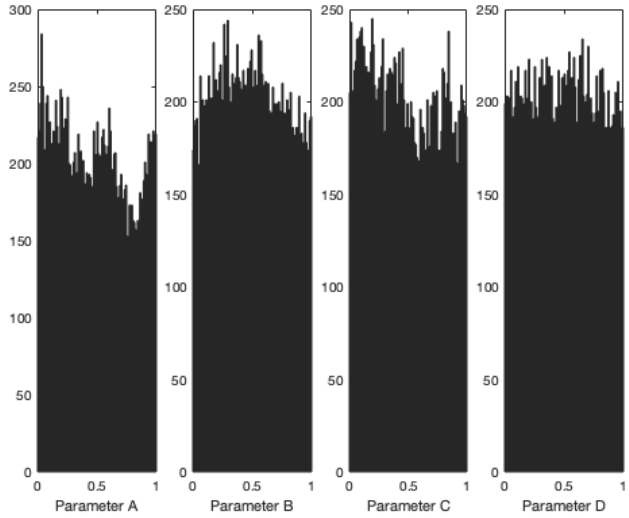

Figure 2: Parameter distributions. Histogram of sampled parameter values *mhsample* algorithm.

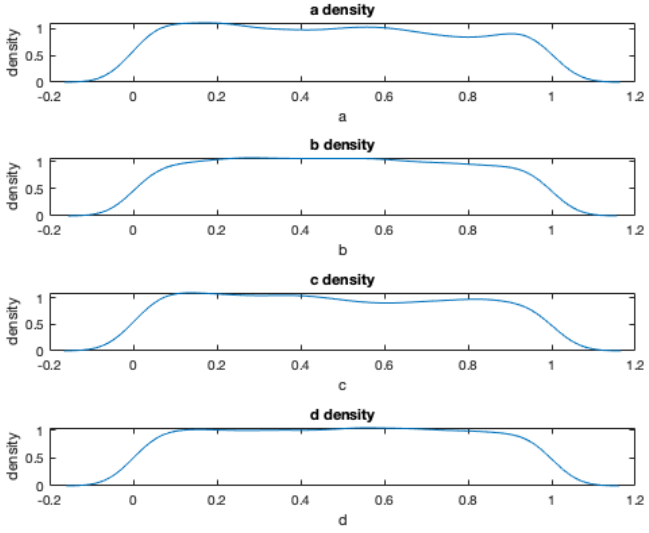

Figure 3: Parameter posterior PDFs. Density functions estimated after *mhsample* algorithm sampling.

Full access to our code can be found **here**.

We recognize that the outcome of the data is not great. There are not clear and recognizable distribution functions for any of the parameters. Also note that this function was run on a subset of the data (the years 1905-1935). Although this is not a great technique, as the ultimate goal is to be able to run our algorithms on the whole data set, when we did try to do this, we received ODE warning messages (integration errors). Only using data from the final 30 years (when the data was more regular) at least allowed us to bypass those warnings and produce some posterior PDF functions.

1.1.1 Adapted Code from Smith, 2014

Smith's Chapter 8 uses a 'sum of squares' approach to MH, less explicitly evaluating the likelihood and prior functions as was done in our self-authored code for fitting the linear model (Week 1). The fit of the parameters is determined by reducing the sum of squares residue between the model outputs and the observed data. Algorithmically, this functions in a close-to-identical fashion to previous descriptions of MH. Pseudocode from Smith is included below:

Algorithm 8.5 (Random Walk Metropolis with Noninformative Prior).

1. Set number of chain elements M and design parameters n_s, σ_s
2. Determine $q^0 = \arg\min_q \sum_{i=1}^{n} [v_i - f_i(q)]^2$
3. Set $SS_{q^0} = \sum_{i=1}^{n} [v_i - f_i(q^0)]^2$
4. Compute initial variance estimate: $s_0^2 = \frac{SS_{q^0}}{n-p}$
5. Construct covariance estimate $V = s_0^2 [\mathcal{X}^T(q^0)\mathcal{X}(q^0)]^{-1}$ and $R = \text{chol}(V)$
6. For $k = 1, \ldots, M$
   (a) Sample $z_k \sim N(0, I_p)$
   (b) Construct candidate $q^* = q^{k-1} + Rz_k$
   (c) Sample $u_\alpha \sim \mathcal{U}(0,1)$
   (d) Compute $SS_{q^*} = \sum_{i=1}^{n} [v_i - f_i(q^*)]^2$
   (e) Compute
   $$\alpha(q^* | q^{k-1}) = \min\left(1, e^{-[SS_{q^*} - SS_{q^{k-1}}]/2s_{k-1}^2}\right)$$
   (f) If $u_\alpha < \alpha$,
   Set $q^k = q^*$, $SS_{q^k} = SS_{q^*}$
   else
   Set $q^k = q^{k-1}$, $SS_{q^k} = SS_{q^{k-1}}$
   endif
   (g) Update $s_k^2 \sim$ Inv-gamma(a_{val}, b_{val}), where
   $a_{val} = 0.5(n_s + n)$, $b_{val} = 0.5(n_s \sigma_s^2 + SS_{q^k})$

Figure 4: Pseudocode for MH where q denotes a choice of parameters

This main place where this differs from code we had written in the past is in its construction of the covariance matrix, V, and its Cholesky decomposition R. In linear models, we were able to assume a predetermined diagonal matrix for V, however this does not hold for nonlinear models like Lotka-Volterra.

In step 5 of the pseudocode, an initial estimate of the covariance matrix is composed using $\chi(q_0)$, the sensitivity matrix. The sensitivity matrix quantifies the sensitivity of each parameter at each data point, creating an n-by-p matrix (where n is the number of datapoints, and p is the number of parameters). Each entry in the matrix, $\chi_{ik}(q_0)$ is defined by

$$\chi_{ik}(q_0) = \frac{\partial f_i(q)}{\partial q_k}$$

representing the partial derivatives of the modelling function taken with respect to each of the parameters. Implemented in file *senseq.m*, we were able to compute the sensitivity matrix given parameters, a time series,

and initial populations. This was then used to find the covariance to be used in selecting new candidate parameters. Following steps 3-5 in the pseudocode,

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solve ODE for initial params
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
model_sltn = model_sol(q_init,t,pops_init)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construct the sensitivity matrix chi, covariance matrix V.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  chi = senseq(q_init, t, pops_init);
  V = inv(chi'*chi)*var;

  % Ensure sensitivity matrix is positive-definite
  try
    R = chol(V);
  catch
    R = chol(nearestSPD(V));
  end

  Hres = popsdens(:,1) - model_sltn(:,1);
  Lres = popsdens(:,2) - model_sltn(:,2);
  Hres_T = Hres';
  Lres_T = Lres';
  % Initial variance estimate
  SS_old = Hres_T*Hres + Lres_T*Lres;
```

The rest of the steps in the code (running the MCMC chain itself) are typical and consistent with our previous work and explanations. This estimation of the covariance matrix allowed for more accurate proposals of candidate parameters according to step 6*b* of the pseudocode, resulting in chains that converged to reasonable mean parameter sets, and providing better, more informed outcomes than hard-coding a guess for a diagonal covariance matrix.

While the algorithm did not converge exactly to the true parameters, $q_{true} = [\alpha, \beta, \gamma, \delta] = [.7436.0507.7669.0259]$, the algorithm was close: $q_{pred} = [.6731.0671.6961.0376]$. These discrepancies can possibly be explained by the relatively low number of iterations on the chain (1e4), as well as by the elementary nature of the algorithm. MH on its own is rarely used for fitting models such as these, but it was a proof-of-concept for it to work as well as it did. The resultant chains are shown below:

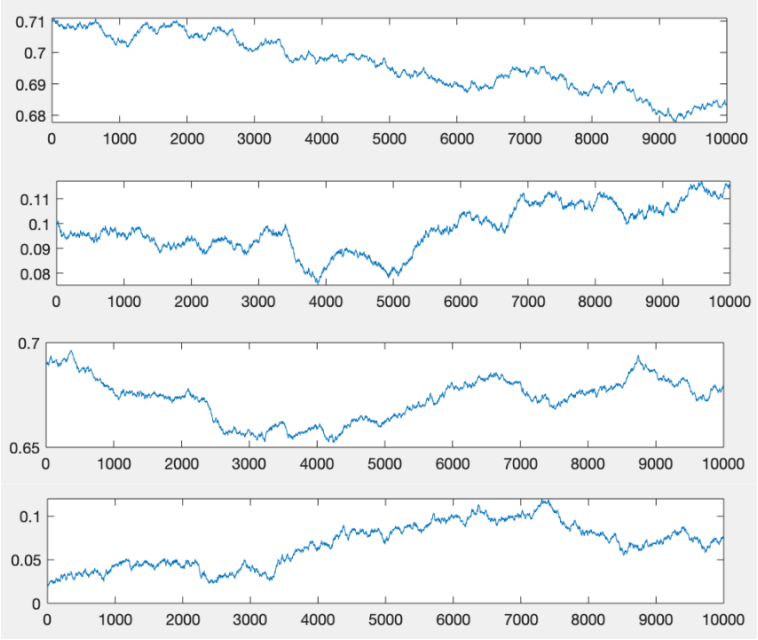

Figure 5: MCMC chains using adapted Smith code (top to bottom: $\alpha, \beta, \gamma, \delta$)

2 Fitting with Delayed Rejection Adaptive Metropolis (DRAM)

After (semi-)successfully implementing 2 versions of the Metropolis-Hastings algorithm, our next goal was to implement a *Delayed Rejection Adaptive Metropolis* (DRAM) algorithm. We read Chapter 8 from Ralph C. Smith (2014) to learn about this method. DRAM builds on techniques of previous MCMC algorithms that we have discussed, MH and adaptive Metropolis (AM), and thus requires similar initialization of constants and distributions (proposal, likelihood, and prior). In previously examined algorithms, the proposal distribution is assumed given some information about parameter values and variability (type of dist, covariance matrix). In adaptive methods, information learned about the posterior distribution is incorporated into how proposed parameters are accepted.

2.1 Adaptive Metropolis

DRAM uses the principle of the *adaptive Metropolis* (AM) method:

1. Sample parameter sets for a period of length k_0 where values are computed using an initial covariance matrix V_0
    - k_0 is chosen to balance mixing with sufficient diversity of points to ensure a non-singular covariance matrix
    - Shorter adaptation intervals tend to produce better mixing and high acceptance ratios; thus the chain is more responsive to new data
    - In practice, k_0 is often 100
2. At the kth step of the chain, adaptation commences: the initial V_0 is updated to the covariance matrix, V_k

$$V_k = s_p cov(q^0, q^0, ..., q^{k-1}) + \epsilon I_p$$

    - s_p is a design parameter that depends o the dimension (p) of the parameter space; a common choice is $s_p = 2.38^2/p$

- The term ϵI_p refers to an ID matrix of dimension p. It simply exists to ensure that V_k is positive definite. Often, $\epsilon = 0$

2.2 Delayed Rejection

In the standard Metropolis algorithm, when a candidate parameter set q^* is rejected (with its prescribed acceptance ratio), the current step parameter set, q^{k-1}, is retained and q^* is discarded before choosing the next sample. In the *delayed rejection* (DR) algorithm, when q^* is rejected, alternative candidates q^{*j} are constructed and considered instead of immediately reverting back to q^{k-1}. For example, q^{*2}, a second-stage candidate, is chosen using proposal function

$$J_2(q^{*2}|q^{k-1}, q^*) = N(q^{k-1}, \gamma_2^2 V_k)$$

- $V_k = R_k R_k^T$ is the covariance matrix produced in the adaptive algorithm
- $J_2(q^{*2}|q^{k-1}, q^*)$ indicates that we propose q^{*2} having started at q^{k-1} and rejected q^*
- $\gamma_2 < 1$ narrows the second-stage function and increases mixing

The acceptance criteria for the second-stage candidate is

$$\alpha_2(q^{*2}|q^{k-1}, q^*) = min(1, \frac{\pi(q^{*2}|v)J(q^*|q^{*2})[1-\alpha(q^*|q^{*2})]}{\pi(q^{k-1}|v)J(q^*|q^{k-1})[1-\alpha(q^*|q^{k-1})]})$$

The DRAM algorithm works as follows: Starting with parameter set q^{k-1}, we choose candidate sample q^*. If q^* is rejected, we propose candidate q^{*2}. If q^{*2} is rejected, we impose candidate q^{*3} and so on until we accept candidate q^{*j}. We begin sampling again with $q^{k-1} = q^{*j}$ as our current parameter set. After taking k samples, we update our covariance matrix (and thus our proposal distribution and acceptance ratio).

In the DRAM algorithm, the adaptive Metropolis and delayed rejection work together: AM updates the proposal via previously accepted candidates and the covariance matrix DR alters the proposal function to improve mixing.

2.3 Algae Example: Code from Laine, 2018

We looked at code from Laine 2018 which uses DRAM to parameterize an algae predator-prey ODE model. The model consists of 3 differential equations representing phytoplankton (A) and zooplanktaon (Z) populations, as well as phosphorus (P) density (food for the algae). This model included 20 parameters (whose relationships were combined into 6 parameters) representing the interactions between the 3 equations. Full details of the model can be found **here**. We successfully ran the code for the algae model, but our next goal was adapt the code for the Lotka-Volterra model.

2.3.1 Adapting to Lotka Volterra

In structure, the Algae example and the Lotka Volterra model are very similar: both are systems of nonlinear ODE to be fit to a datset. This meant that the alteration of the code was minimal to adapt Laine's example to the Hare-Lynx data. The code, as it relies on the library 'mcmcstat' to perform the DRAM fitting in a kind of 'black box', only required altering the inputs and their structures.

- The structure of the likelihood function was left unchanged, as a normal likelihood was appropriate for both models (though, of course, the ODE solved within the likelihood function was changed to match the Lotka-Volterra system)
- The number of parameters and their values were changed; instead of 20 parameters, we now use just four. For biological reasons, these parameters were restricted to be positive ≤ 1, and assumed to be uniformly distributed within that interval.
- Noise within the system was assumed to follow an inverse Chi-squared distribution (in keeping with the algae example). Initial variance and DOF were calculated from data.

With all required aspects defined, 1000 burn-in iterations are performed. After these 1000, the final accepted parameter set, q, is used as the initial guess in a second, longer chain of 10000 iterations. As discussed in previous week's work, allowing for burn-in lessens the impact of the initial parameter guess on the final posterior distribution, as the data collected as the chain 'navigates' to the neighborhood of the true parameters is thrown out. Ideally, this means only 'significant' data will be used to calculate the posterior distributions, leading them to be unimodal and easily analyzed for mean or MAP (maximum a posteriori) parameters.

This code also provides a function through which we can check that our chain has converged via Geweke's diagnostic. Geweke's diagnostic compares the mean of the first 10% of the chain's samples to the last 50% of the chain's samples. Perfect convergence is indicated when these two means are equal - that is Geweke's diagnostic is 0. However in practice, a number very close to 0 can still indicate convergence. In our code, the Geweke value represents a *ratio* of the means of the first 10% and last 50% of the chain, thus a value of 1 indicates that the two means are equal. The Geweke value, shown in Figure 6, for each of the parameters indicates that the chain has converged. In addition, Figure 6 shows the mean values of each of our parameters (for this particular run) and the amount of error produced while running the chain.

	mean	std	MC_err	tau	geweke
alpha	0.71731	0.018554	0.0037791	819.6	0.96892
beta	0.024935	0.0011731	0.00015603	220.44	0.98647
gamma	0.6882	0.072796	0.016175	575.98	0.85994
delta	0.065511	0.0053789	0.00083638	353.51	0.93472

Figure 6: Results of the function *chainstats*. *mean* indicates mean values of the parameters after sampling; *std* indicates the standard deviation per parameter; *MC-err* indicates the amount of error introduced during sampling; *tau* refers to the integrated autocorrelation time; *Geweke* indicates the ratio between the means of the first 10 and last 50 percent of the chains.

We find that the mean values of each of our parameters fall relatively close to our expected mean values $[\alpha\ \beta\ \gamma\ \delta] = [.7436\ .0507\ .7669\ .0259]$.

This code, unfortunately, lacks consistency when it is run (note the large envelope of reasonable parameter choices in Figure 9). Though it does produce results that appear valid according to Geweke's diagnostic and are in keeping with the known true parameters of the system, it frequently returns very different looking distributions. Additionally, though the chains do often converge to reasonable parameters, we question why they do not fall even closer to the true parameters. It is not yet clear as to whether this is related to the limitations and randomness involved in DRAM, or errors we introduced in coding the model. Despite this, helpful visuals are produced than demonstrate the relationships between model parameters, the predicted posterior distributions, as well as a visualization of the solved ODE overplotted with the datapoints.

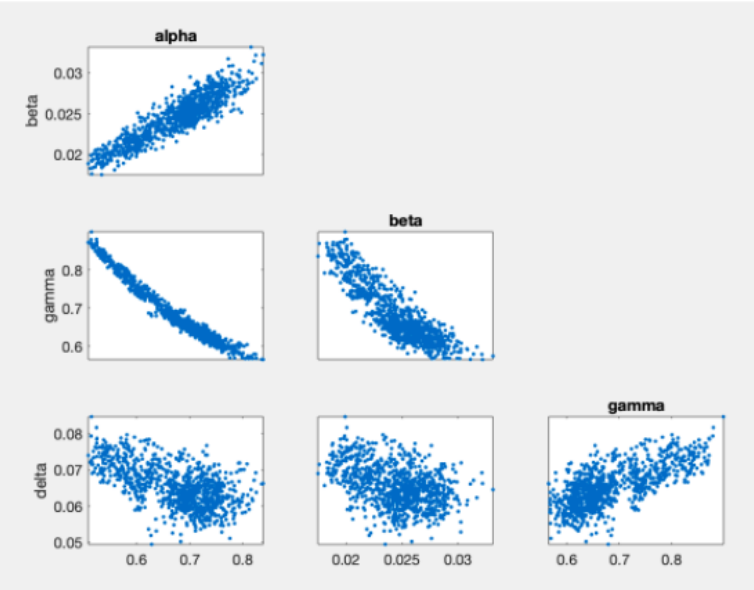

Figure 7: Relationships between the DRAM sampled values of parameters α, β, γ, and δ

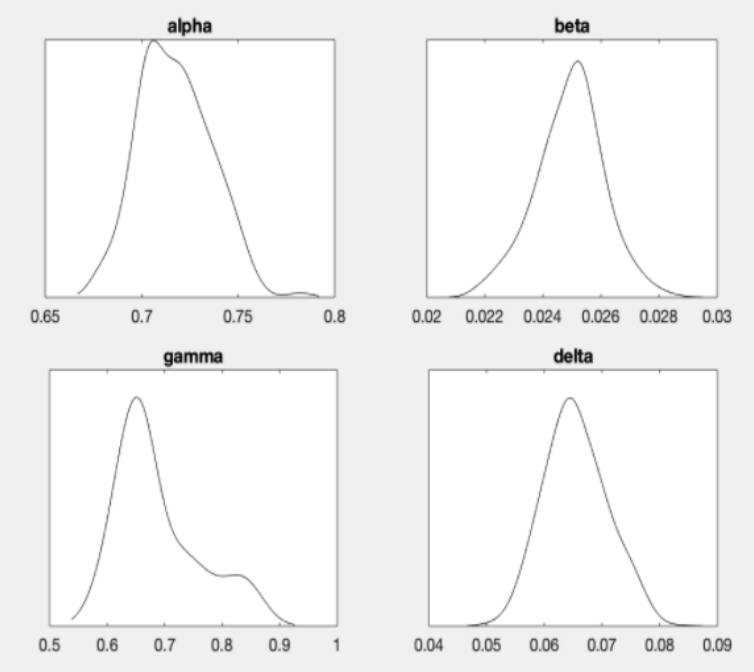

Figure 8: Posterior probability density distributions for each parameter.

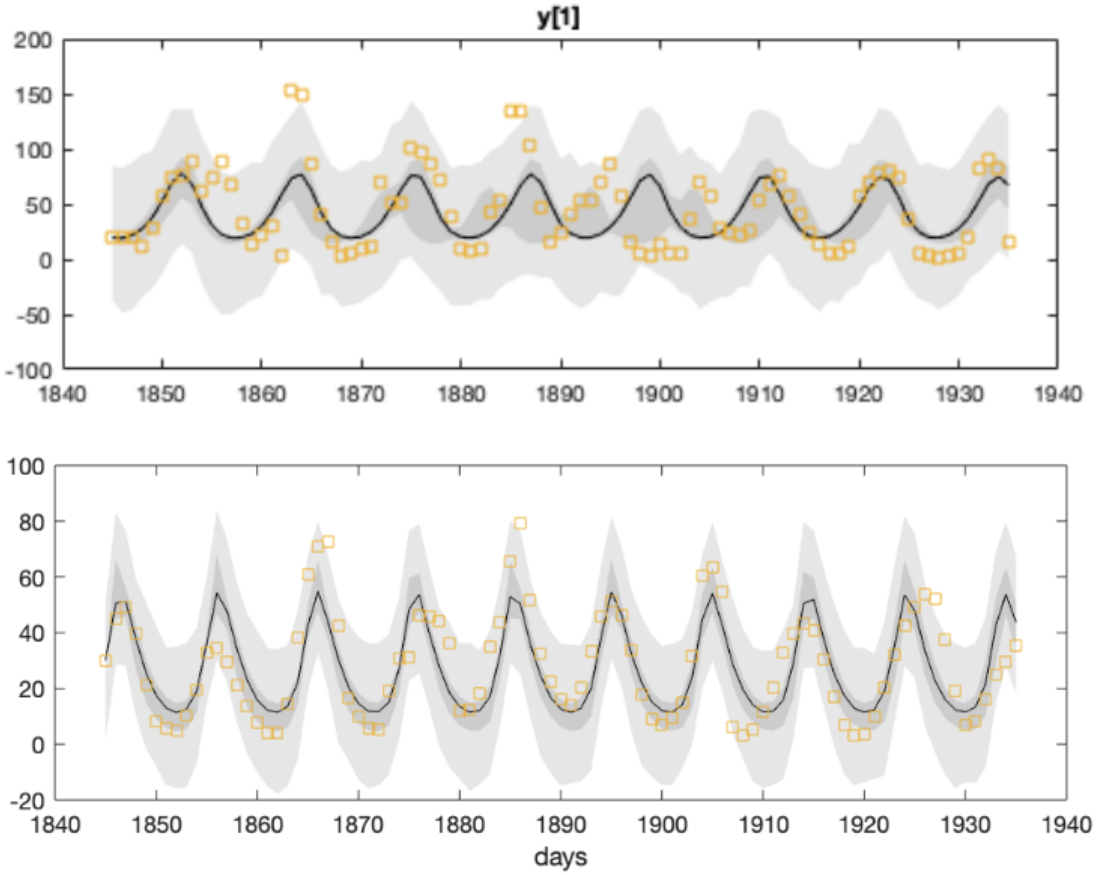

Figure 9: We sampled 500 realizations from both the burn-in and fully-sampled chains and plotted the predictive calculations (*mcmcpred* function). Overlaid in yellow are the data points from the original data set. (top: hare, bottom: lynx)

3 Moving Forward

Clearly, the parameterized system plotted above does not align very well with the observed data. Not only do we plan on debugging this code and ensuring that the proper assumptions were made, but we also hope to implement measures to validate the model formally by segementing the dataset into calibration and validation data, ensuring that model behavior is predictive of observed data across all subsets of data. This should help negate the effect that potential outliers have on our choice of parameters. Especially in very noisy data, such as the hare population, this would become beneficial.