Week 3 Summary

Subteam 2

June 2020

1 Introduction

This week we worked with two systems. The first was the Lorenz system from last week. Previously, we used a Chow base code to perform state estimation on this system. This week we used this system to test out the built-in Matlab unscented kalman filter function. The second system was the predator-prey system. We first ran state estimation on this system, and then ran joint state and parameter estimation on it.

2 Lorenz Estimation using the built in Matlab Unscented Kalman Filter Function

2.1 State Estimation of Lorenz Function using Matlab's UKF

Having performed state estimation on the Lorenz Function from the chow paper using built from scratch unscented kalman filter code, I now attempt to run a state estimation of the Lorenz Function from the chow paper using Matlab's built in UKF function. I used the Albers_UKF codde as a template, the generated data as well as some of the functions were taken from the chow paper. By doing this, I hope to see what differences there are in the performances and if the built in unscented kalman filter is a workable option for future projects. One of the major differences between the chow ukf function and the matlab one is that the built-in matlab ukf does not have a cholesky factorization in the covariance propagation step. The assumption was that this would lead to the model breaking at a certain point.

2.1.1 Model Set Up

The matlab unscented kalman filter function has many parameters. I filled them in as follows:
Measurement Function: I got this function from the chow paper
Transition Function: I got this function directly from the chow paper (the runge kutta approximation of the function)

Alpha, beta, and kappa parameters: I got these parameters from the chow code.
Additive vs Non-additive noise: I made both the process and measurement noise additive. That is because in the lorenz system, the noise is additive
Noise covaraince matrices: I got both of these from the chow paper
Initial state guesses: I also got these from the chow paper. The unscented Kalman filter object has two other attributes, State and StateCovariance that represent the state estimate and the covariance matrix. These are updated using the functions predict and correct.

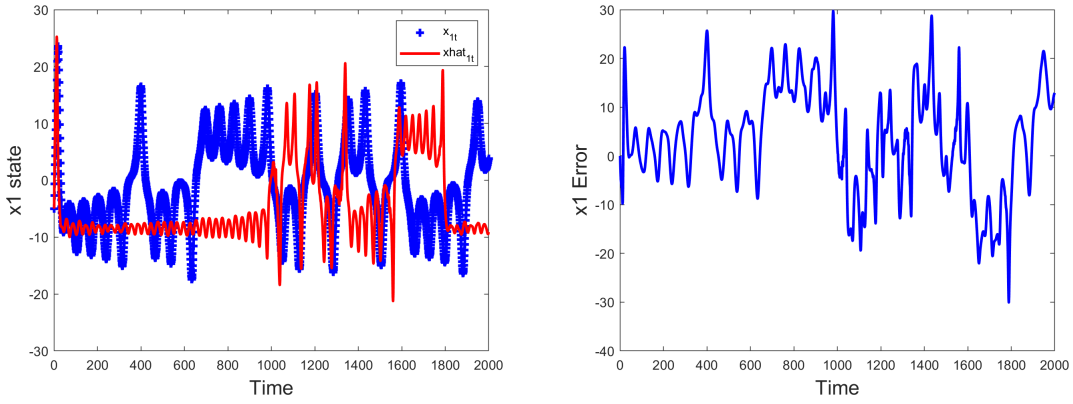
2.1.2 Model Execution

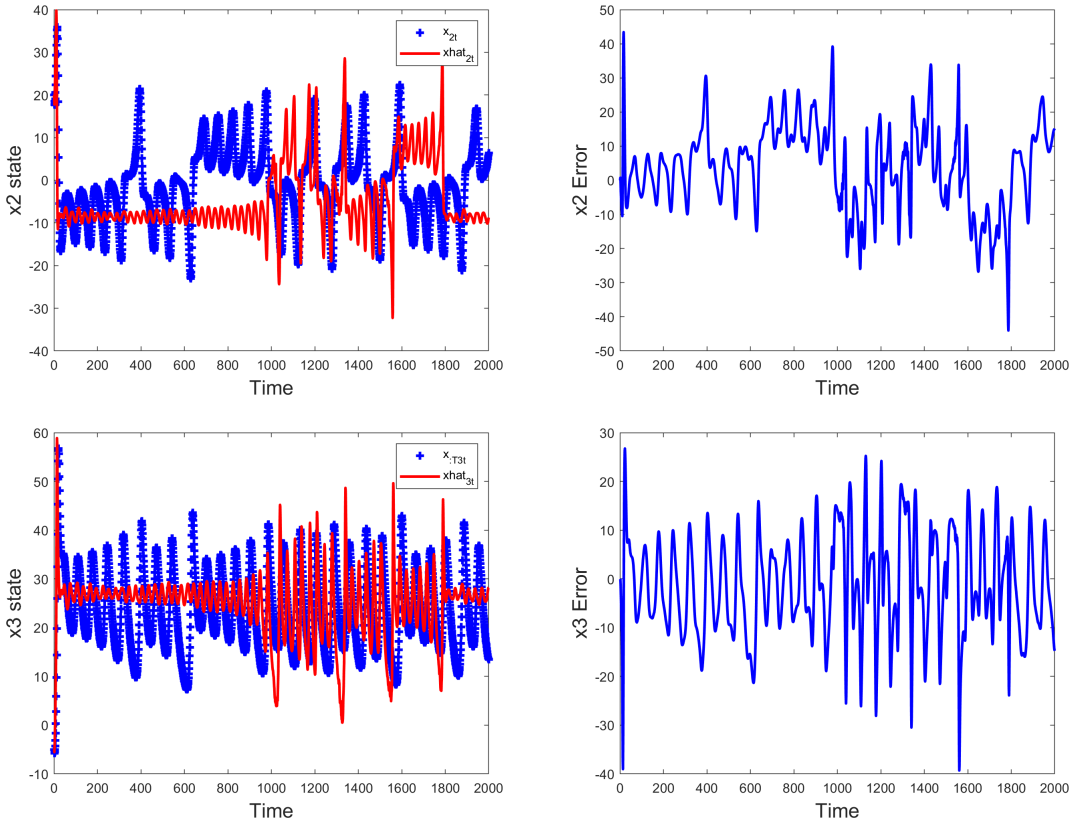
Once it is created and all its attributes are set, the unscented Kalman filter object has two main functions: predict and correct. The function *predict* takes the current state, what we would know as $\hat{x}_{k-1}$ and returns a prior estimate, or $\hat{x}_k^-$. The function *correct* takes in the current estimate $\hat{x}_k^-$ and the current measurement y_k and returns the posterior estimate $\hat{x_k}$. Then, the cycle can begin again.

2.1.3 Results

This would always run, however the results would be highly inaccurate. The graphs barely matched at all and the error remained high. There was also a high degree of inconsistency between how each run of the code would look.

An example of the results for this transition function are below. This code was run with an alpha value of 10e-4. We have the generated data overlaid with the predictive states in each case as well as the error graphs for each state.





Obviously, the results here are not great, especially as compared to the built from scratch unscented kalman filter showcased last week. As a result, that is the avenue we will pursue in the future. git

3 Predator-Prey Model Introduction

Having built an understanding of how the Unscented Kalman Filter (UKF) can be applied to do state estimation for the Lorenz system, our goal is now to apply the filter to a new system in order to study the UKF in a new context as well as obtain a better general understanding of the code we have been using. Thus, we now move to using the UKF to perform state estimation for the Lotka-Volterra Predator Prey model. The data that is used in this section is the Hares-Lynx data set from 1845 to 1935. To begin, let us set up the Predator Prey model. The system is described by the following two differential equations:

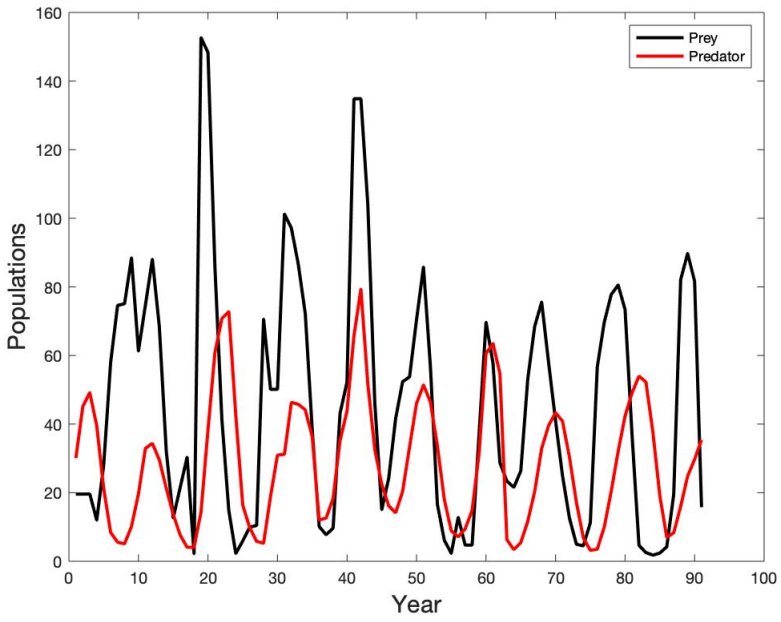
$$\dot{x}_{1t} = \alpha x_{1t} - \beta x_{1t} x_{2t}$$

$$\dot{x}_{2t} = -\gamma x_{2t} + \delta x_{1t} x_{2t}$$

Where $\alpha, \beta.\gamma, \delta$ are the model parameters.

3.1 The Data

The raw data that was used for this process was the 1845-1935 Hares-Lynx dataset, where Hares are the prey and Lynx the predator. A figure of the raw data is below:



Looking at the raw data, it is evident that there is a substantial amount of noise. Although we do see the general shape of the predator-prey relationship, there are parts of the dataset which show some strange behavior. In doing our analysis, we will thus use three different portions of the data and compare results:

- The whole dataset: this will be what will be of most interest
- Years 1850-1935: the early behavior of the prey population is quite strange. Since UKF's tend to overtrain on early portions of the data, we are interested in what our results will be on this subportion of the data

- Years 1908-1935: This is the most "regular" portion of the data, and is often the subset chosen in literature for understanding a classical predator prey model. Thus, we will see how our results look for this portion alone

3.2 Our Process

The process of understanding the capabilities of the UKF on this predator prey model will unfold in a few steps. First, we need to modify our code and make the necessary adjustments so that it will function correctly. The details of the code modifications will not be discussed here. Next, the model will be parametrized. Recall that when using the UKF for state estimation, parameter values are held constant. Thus, for each of the subportions of data described above, we will need to get appropriate true parameter values. Now that our code and model is ready, we will first test on fake data. This data will be generated using the *ode*45 solver in MATLAB with a large quantity of noise. Then, we will use our UKF code to attempt to do state estimation on this fake data to benchmark how well our code does in this scenario. Finally, we will test our implementation on the three sub-portions of data above and analyze the results. We will now go into more detail for each of these sections below.

4 Model Parameters

Before we can do state estimation, we need to get true parameter values that will remain constant throughout the UKF process. In order to do so, we will use a Particle Swarm Optimization approach to fit a Least Squares set of parameters to the data. This code is not our own and has been borrowed from INSERT for the purpose of this project. The parameters we obtain are as follows:

For the full dataset:

- $\alpha = .7436$
- $\beta = .0507$
- $\gamma = .7669$
- $\delta = .0259$

For years 1908-1935:

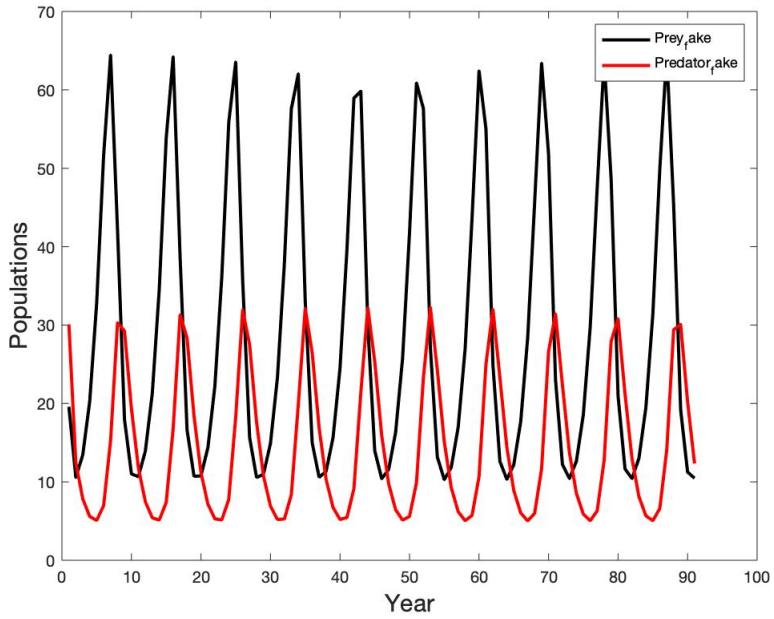
- $\alpha = 0.5969$
- $\beta = 0.0413$
- $\gamma = 0.6967$
- $\delta = 0.0202$

For years 1850-1935:

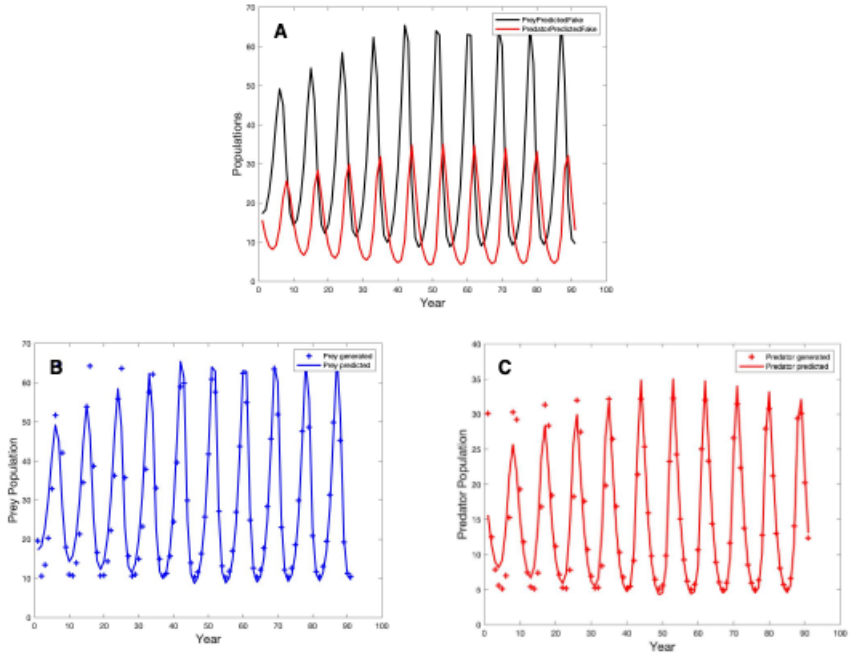
- $\alpha = 0.6259$
- $\beta = 0.1067$
- $\gamma = 0.7229$
- $\delta = 0.0158$

5 Benchmarking with Fake Data

Fake data was generated only for the time frame of the full dataset. Thus, the parameters that were used were those from the section above related to the full dataset, and artificial, additive, white, Gaussian noise was added in order to make the scenario more realistic. The Lotka-Volterra system of ODE's was solved using MATLAB's *ode*45 and then 91 time points were taken out, one for each year in the dataset. This process produced the following data:



Now, the UKF was used to try and perform state estimation on this fake data. To visualize our success, we plot both the generated predator and prey together (A), as well as a graph of the overlayed generated data vs predicted data for both predator (C) and prey (B) separately:

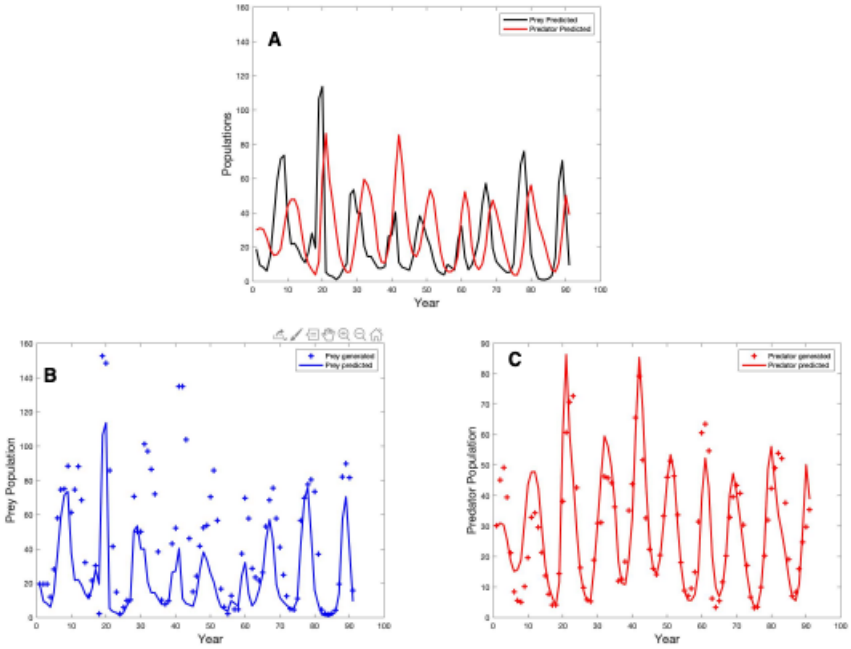


We can see that, overall, our code is very successful at performing state estimation in this context. Even with a substantial amount of artificial noise, we are able to do a good job of fitting to the data. The only issue we can see is that some of the peaks are not fully reached in the dataset. However, the overall results confirm to us that our code is functional, at least in the context of this "nice" data. However, to test the data more thoroughly, we now turn to the real dataset.

6 The Real Data

6.1 The Entire Dataset

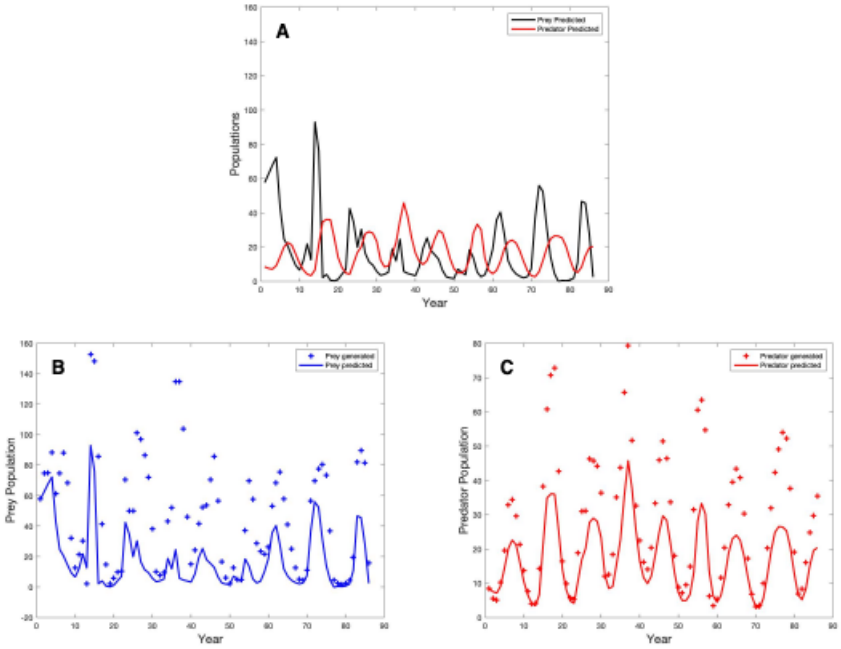
Similar to the figure we used to understand our results for doing state estimation on the fake data, we can create a similar figure for the entire dataset, where (A) shows the state estimates for the predator prey relationship, (B) overlays the prey predictions with the raw data, and (C) overlays the predator predictions with the raw data:



Looking first at **A**, the general predator prey realationship can be seen in the results. However, there is a consistent error, particularly in the middle of the data, where the predator population is higher than that of the prey. Comparing to the raw data, we can see that the prey population should be the one with higher values. Now, when turning first to **C**, it is clear that we are doing a very good job of predicting predator populations. Thus, it is not that the predator populations are too high, but is rather that the prey populations are too low. This belief is confirmed when looking at **B**. Although the "up and down" motion is present, the values are far too low, and this is clearly where most of our error is coming from. It will be interesting to see if this trend continues in other subportions of the data.

6.2 1850-1935

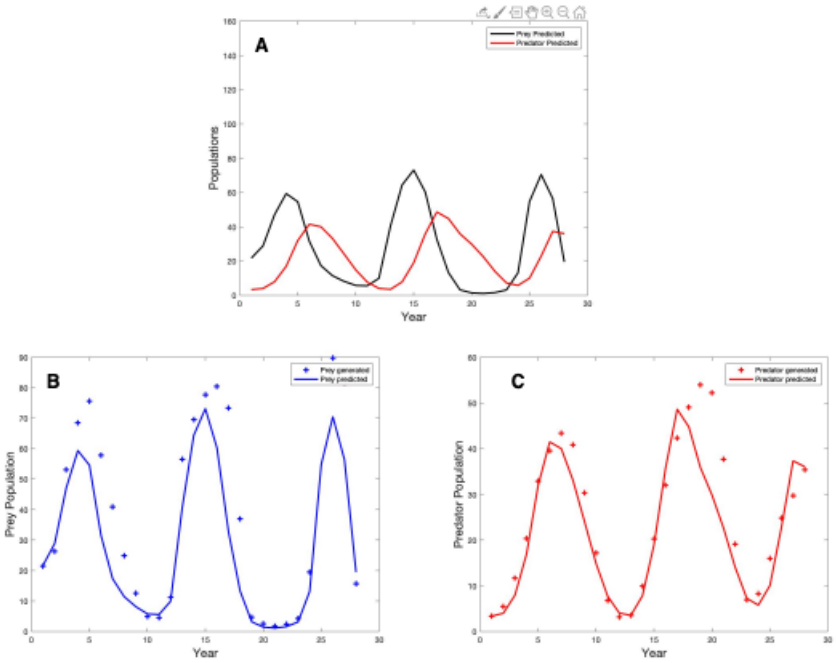
In this portion of the dataset, we are removing the early "weird" behavior to see if our results improve because of this. Once again, we include a figure where (A) shows the state estimates for the predator prey relationship, (B) overlays the prey predictions with the raw data, and (C) overlays the predator predictions with the raw data:



Surprisingly, results appear to get worse under these new conditions. Now, we can see that both predator and prey are being consistenly underestimated. It is important to note that there is the possibility that this is due to less accurate true parameter values. Since we have fewer data points here, it is possible that the quality of the parameter fit from the Particle Swarm algorithm is not as good as it was when using the full dataset. Nonetheless, this confirms that more data helps the UKF and that we should, in general, use as much good data as possible. As a final test, we turn to the "nicest" portion of the data, 1908-1935

6.3 1908-1935

In this last section, the data we are basing our UKF state estimation off is the nicest of all of the subsections. At this point, we have very regular behavior that we expect to be able to get a good fit to. Similarly to before, we present a figure where (A) shows the state estimates for the predator prey relationship, (B) overlays the prey predictions with the raw data, and (C) overlays the predator predictions with the raw data:



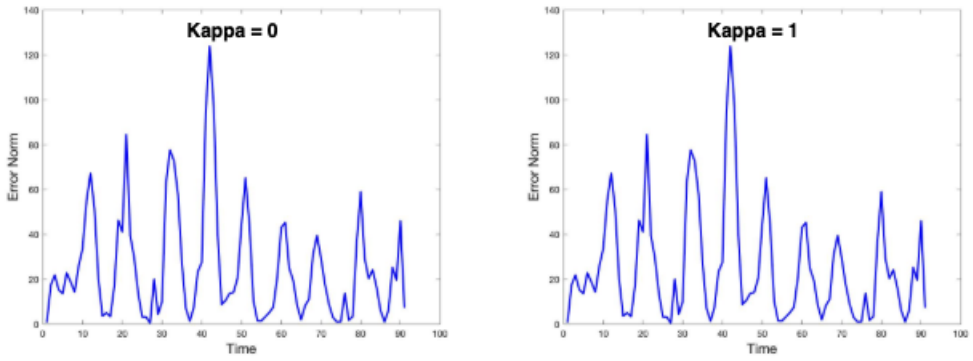
It is evident that, although the dataset is small, we are able to get a very good fit to the data. All around, from the predator to the prey predictions, the state estimations are consistently good estimates for the real data. This, as mentioned, is unsurprising. However, this scenario is also incredibly unrealistic. In reality, it is not viable to selectively pick portions of the data to use that are nice and completely ignore the rest. Thus, it is assuring that we also had decent results on the full portion of data, where the biggest improvement to be made is in improving prey estimates to climb to higher values at its peaks.

7 UKF Parameters and Sensitivity Analysis

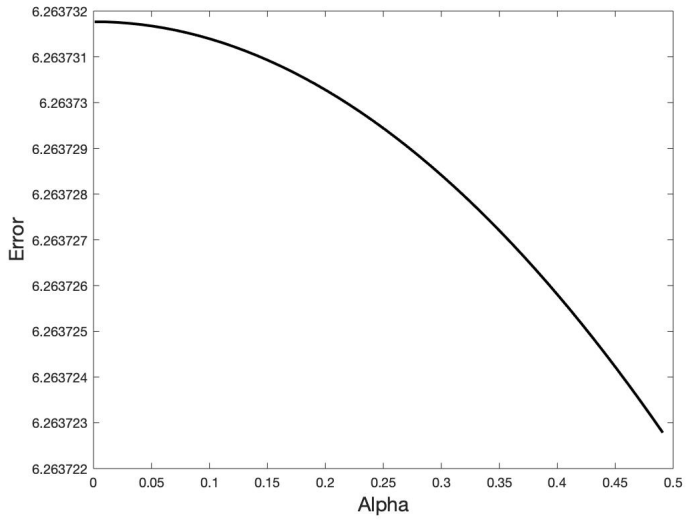
Apart from the model parameters, the UKF itself has parameters that need to be set. Namely, these are α and κ, to control the spread of the sigma points chosen, and β to control our prior knowledge of the distribution of the states. In our scenario, β is kept constant at 2 which, based on the literature we have explored, represents assumptions of a Gaussian distribution.

For κ, it is expected to be set to either 0 or $3 - L$, where L is the number of states in our system, which is currently 2. Thus, the two values of κ we must check are 0 and 1. First, it is important to note that all these comparisons will

be done on the full dataset, as that is what is of most interest here. To do the comparison, we will plot the norm of the error vector $x_t - \hat{x}_t$ at each time point for κ equal to 0 and 1:



We can see that these plots are nearly identical, revealing that kappa does not seem to play a large role in the quality of UKF estimates in this system.
Values for α, unlike those for κ, have a wider range of possibilities. In particular, they can range for $10e-4$ to 1. In our scenario, we found that problems arose in the ability for the system to make predictions at all if α rose too large, so we have chosen to limit our range from $10e-4$ to 0.5, going in increments of .01. In this case, we use the final error value (ie at time 91) to represent the success of a particular α value and plot this relationship:



Here, it is evident that larger alpha values are helpful, however the extent to which they are is very slight (the scale on the y axis is miniscule). In essence, it appears that UKF parameters do not play a significant role in this particular system.

8 Joint Estimation

Having now explored using the UKF to perform state estimation on the predator-prey system, we move onto our next task: doing Joint Estimation on the Predator Prey system. In joint estimation, both our unknown states (x's) and our parameters (in this case α, β, γ, and δ, are approximated. If we have states x_k, parameters w_k, transition function F, state process noise v_k, parameter process noise r_k, observables y_k, exogenous input u_k and measurement noise n_k, then our joint UKF system is set up as follows:

$$\begin{bmatrix} x_{k+1} \\ w_{k+1} \end{bmatrix} = \begin{bmatrix} F(x_k, u_k, w_k) \\ Iw_k \end{bmatrix} + \begin{bmatrix} v_k \\ r_k \end{bmatrix} \quad (1)$$

$$y_k = [1\ 0] \begin{bmatrix} x_k \\ w_k \end{bmatrix} + n_k \quad (2)$$

Equation is saying that the states, as before, are moved through a nonlinear function F to get to time $k+1$ along with some noise. However, we now additionally have the parameters w_k, with stay the same from time k to $k+1$, apart from some noise r_k. Then, equation 2, which describes the observables, is essentially the same as it was before since the matrix multiplication picks out the observables x_k along with some measurement noise n_k, while ignoring the parameters since those are unobservable.

It is important to note that, from now on, when we refer to a **state vector** we are referring to the vector containing **both** the latent states x_k and the parameters w_k.

8.1 Implementation Strategies

Our goal is now to use MATLAB code in order to perform joint estimation for the Lotka-Volterra predator prey model. We have two starting implementations that we have worked with. First, code originally from Albers to do joint estimation on a T2D model, as well as the Chow code which we have worked with throughout and for which the state estimation was done on the predator prey. These two code bases are both meant to accomplish the same thing (joint state and parameter estimation), however they are written in slightly different ways. We hope to see if one approach is better than the other. We will now discuss the results of these two approaches.

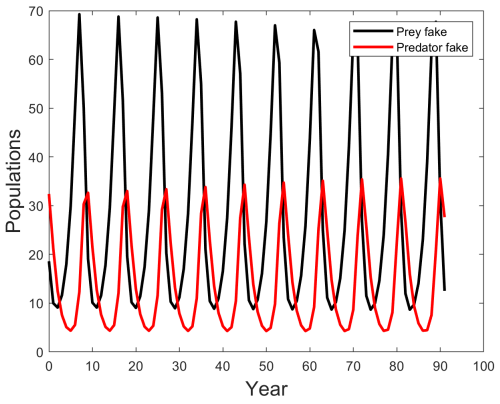
9 The Albers Implementation

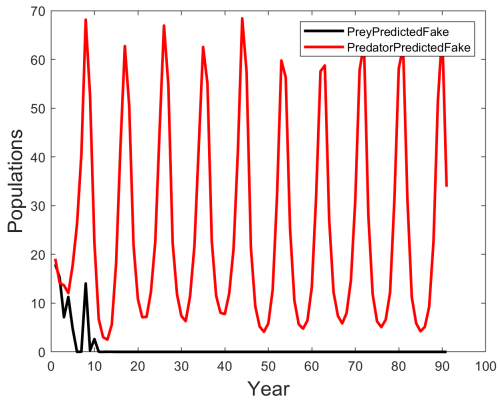
This implementation was based on the Albers code which paramaterized their T2 Diabetes ODE using a joint unscented Kalman filter method. Their code was modified to parameterize the Lotka-Volterra Predator-Prey Model described above.
Some specific modifications beyond replacing the system set up are as follows:

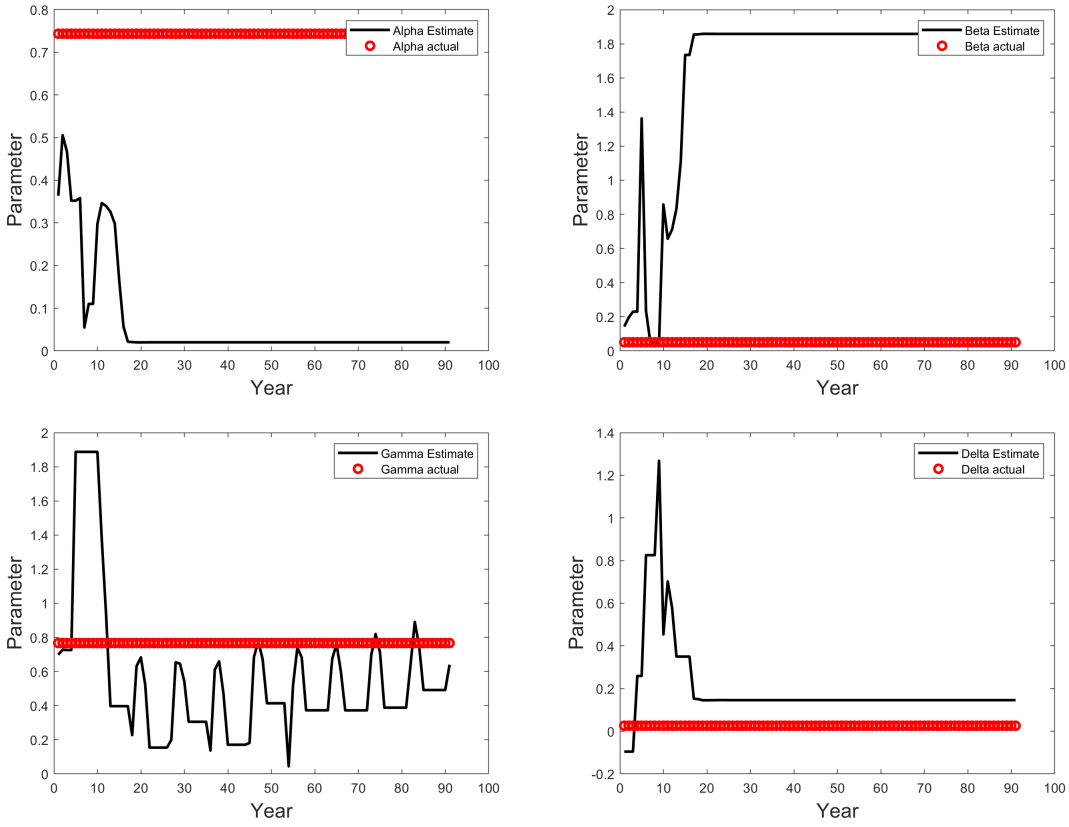
- Constraints were added so that no parameter or state could go below zero and no parameter value could go above 2. This fixed the issue of the code crashing when state values went negative.
- Measurement noise was increased. This allows the filter to better anticipate irregularities in the data.

9.1 Results - Fake Data

Disclaimer: This is very much still in progress data. The filter still needs a lot of work, and right now all of its predictions are pretty messed up in various ways. Below are the generated data, the state estimate, and the parameter estimates.







9.2 Results - Real Data

Currently, I have not been able to get the code to run on the real data. When I try to do so I receive an error that says the covariance matrix was not positive definite. Future work involves trying to find the source of and fix this error. One possibility is to make sure that during the propagation of sigma points, none go below zero.

10 The Chow Implementation

After performing the necessary modifications to the Chow code which performed solely state estimation to get it to run in the Joint case, two main problems emerged (Note: the details of these modifications will not be discussed here but the full code can be accessed on Github (https://github.com/shtyllab/2020-HMC-REU-Codes/tree/master/subteam2/PredatorPreyJointEstimation/UsingChowBase):

1. The filter was prone to crashing when parameter values shot up too high

or went negative

2. The filter performed drastically better on the predator populations but struggled significantly on the prey

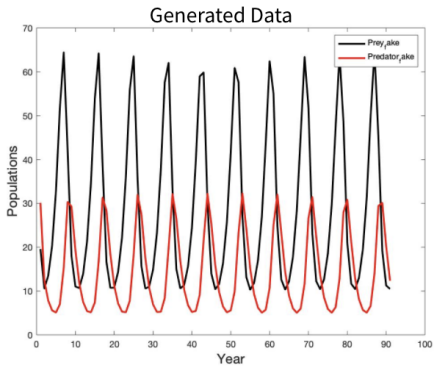
Currently, the following two solutions are in place in order to help the filter in avoiding these issues:

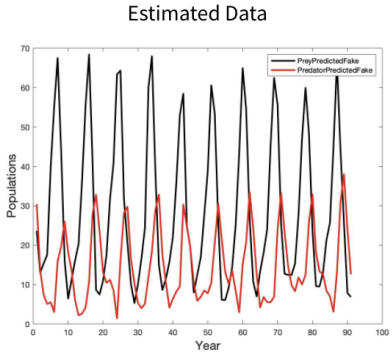
1. Constraints are added on parameter values between 0 and 2 to ensure that they stay within this range. This restriction, of course, requires prior knowledge of acceptable ranges for the parameters and would ideally be removed.

2. Noise covariance matrix values are increased in order to inform the filter that the real data is quite noisy. In particular, the noise values associated with the prey populations are increased. "Good" noise values were found through trial and error but would ideally be estimated in a systematic fashion.

10.1 Results - Fake Data

To begin, we generated artificial data to run the joint UKF on. This data was generated with noise, with more noise being placed in the prey values than the predator. The generated data and results are below:



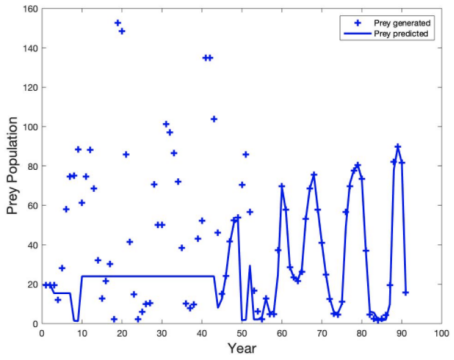


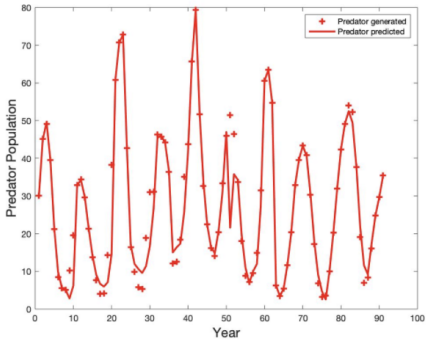
We can see that, even with some noise, the joint UKF does a pretty good job of doing the estimation for the fake data. This gives us some level of confidence in our implementation, but the true test will of course will be running the UKF on the real data.

10.2 Results - Real Data

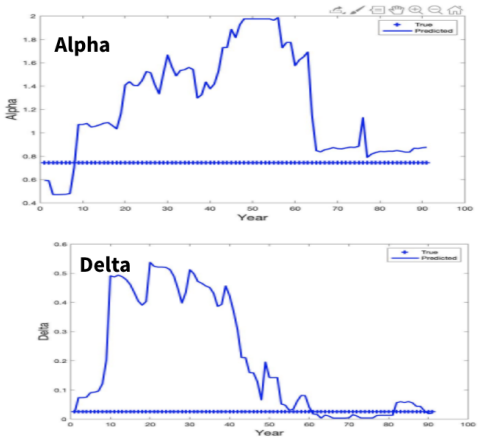
In understanding our results, we are going to look at both the quality of the estimates of the states as well as of the parameters. To begin, let's look at the

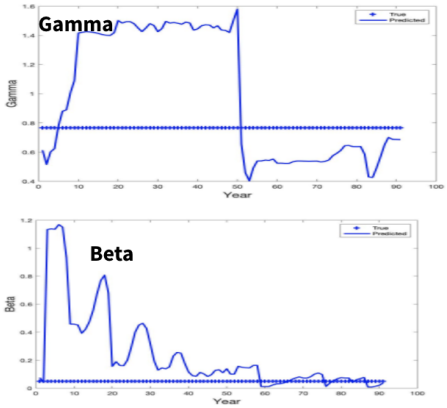
states. Below are overlays of both the real prey data and the predictions (left) as well as the real predator data and the predictions (right):





Let's first look at the predator data. We can see that the filter is able to learn the predator behavior very quickly and does a very good job of doing state estimates. On the other hand, the prey predictions struggle in the beginning. In particular, they go flat until around data point 45, after which the filter is able to learn the behavior and becomes accurate in doing predictions. To understand this discrepancy, it is helpful to look at the behavior of the parameter values. To do that, parameter values, along with their "true" values (constant) are plotted below:





It is evident that, after enough data has been passed into the filter, we are able to do a decent job of converging to the constant values. However, **Delta** and **Beta**, which happen to be the parameters most associated with the predator

populations, converge much more effectively and quickly. This sheds light as to why our predator estimates perform better than those of the prey. Looking at **Alpha** and **Gamma**, which are more associated with the prey, they do eventually get close to the correct values, however do so not as effectively. However, once they do find the correct values, predictions improve dramatically, as we saw earlier.

10.3 UKF Parameters

Joint estimation with the UKF appears to be more sensitive to the UKF parameter values. In particular, running with $\kappa = 1$ resulted in significantly **worse** performance than with $\kappa = 0$. Similarly, finding a value for alpha meant in needed to not be too low or too high. The lowest possible (based on literature), 10e-4, proved to be low, and 10e-2 was too high. By too low" or "too high", we mean that the algorithm would no longer function correctly, specifically because of singularity of matrices. Thus, $\alpha = 10e-3$ is what was chosen upon and what is used throughout these results. The final UKF parameter, β, was left as $\beta = 2$. This, based on literature, resembles us assuming the distribution of the states and parameters to be Gaussian.

11 Next Steps

First off, noise appears to be playing a very large role here. When noise values are changed even slightly, performance of the filter changes quite a bit. Right now, noise values are being determined in an ad hoc fashion. However, it would be more effective, and likely help performance, to develop a systematic way in which to calculate the amount of noise in the data before running the UKF. Additionally, the filter currently relies on prior knowledge of parameter ranges in order to get the UKF to run without crashing early. It would be ideal to modify the method so that these conditions are no longer necessary. Lastly, after improvements on this joint UKF are made to a point where we are ready to move on, our next steps will be to implement a **Dual** UKF approach. Here, two separate UKF's are run, one for the latent states and another for the parameters, simultaneously. This is expected to be a better method for doing these estimates and is the one we would like to eventually apply to the T1D model.