Week 4 Summary

Subteam 2

June 2020

1 Finalization of Joint Estimation Algorithm

At the conclusion of last week, I had a form of the joint estimation for the Lotka Volterra system that was somewhat functional. At that point, predator estimation was working well, prey struggled in the beginning but then performed well, and parameter estimation was decent but not excellent. On June 8th, a couple of changes were made to the UKF system set up that greatly improved performance both in the state and parameter estimation.

1.1 The Changes

The following two changes were made: initialization of the initial covariance matrix P_0 to a 6 by 6 matrix of ones, and reducution of the process noise covariance matrix.

1.2 Initialization of P_0

Up until this point, P_0 had been initialized to the identity matrix, meaning that there were one's along the diagonal and zero's everywhere else. This symbolized that the covariance between all of the states (2 of which are the populations and the other 4 the parameters) are all 0. However, when thinking about the predator-prey system, it is evident that parameters and states are closely related. For example, the predator growth rate parameter is directly dependent on what the predator population is. Thus, by assuming that no variance between the two exists, we miss out on a lot of information. By instead initializing P_0 as a matrix full of ones, we now allow for these variances to be taken into account in the algorithm, greatly improving performance. However, this alone is not enough.

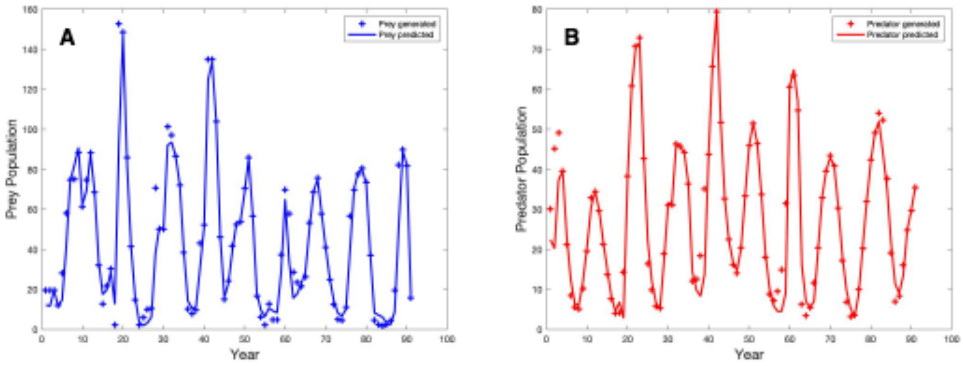
1.3 Reduction of Process Noise

In the UKF, the matrix Q represents the covariance of the process nosie. In general, larger values in Q mean that more noise is expected in the system. Up to this point, our Q matrix was a diagonal matrix with 10e-4 across the diagonal. This had been assumed to be a small enough value. However, this parameter turns out to be very sensitive. A change to using 10e-5 drastically improved performance. We believe that at 10e-4, the algorithm was trusting the predator data very much but because of this was ignoring the prey values (causing the flat lining in the beginning). By turning it to 10e-5, we lost a slight amount of performance on the predator estimation, but this tradeoff was worth it for the drastic improvements in prey state estimation and parameter estimation.

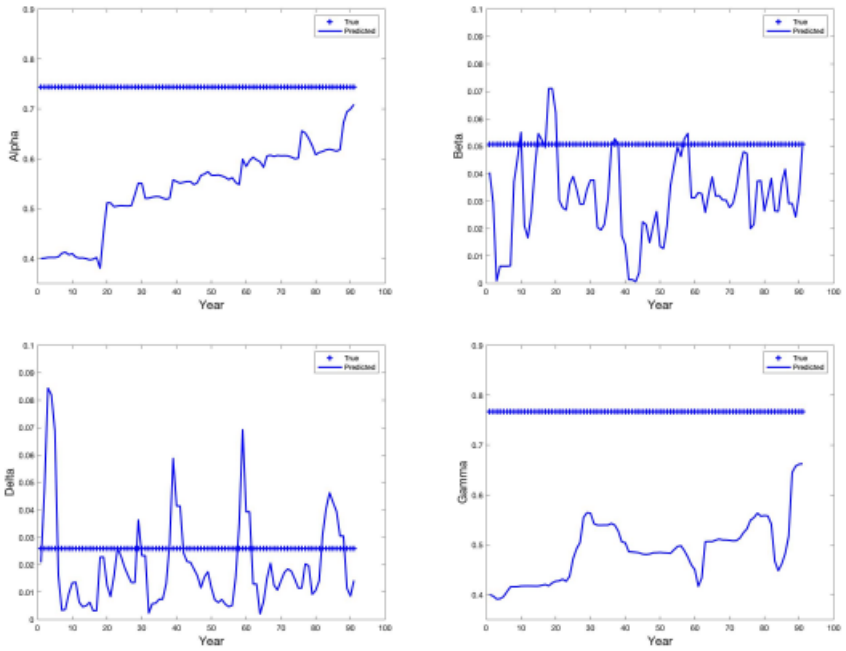
This alteration, along with the one related to P_0 are **both** necessary to create the following results. It is important to note that when one of these alterations is make instead of both the results are not accurate whatsoever.

1.4 Results

Let us look at our results when using the full Hares-Lynx dataset. To test the joint estimation algorithm, we have chosen to initialize our parameter values to roughly half of their true value. First, let's look at our state estimates where **A** is the prey estimates overlayed with the true data and **B** is the predator estimates overlayed with the true data:



It is evident that **both** for the predator and prey, our UKF now does a very good job of estimation the states. Next, lets look at our parameter estimates:



Beginning first with Alpha and Gamma, we see that both are making steady climbs towards their true values. Now, it is true that Gamma never reaches its true value, but we are only currently working with 91 data points, so it is likely that, given more data, Gamma would have enough time to reach its true value. For Beta and Delta, the parameter values appear to oscillate within a small range. Although we would like to see some sort of convergence for these values, the range in which they oscillate, as can be seen by the y axes, is quite small. Thus, it is assuring to see that these values do not spike/drop too high/low. Moreover, the values that they conclude at, which is most important, is very close to the true value.

If one looks back at our previous summaries, they will notice that we had introduced checks on our parameter values to ensure that they did not leave a certain predetermined range. This had previously been something that was necessary for the filter to work. However, we are now able to remove all checks except for non-negativity checks for Delta and Beta. Due to their low initialization and true values, this check is reasonable to include and will likely need to remain in the algorithm for this system.

2 The Dual UKF

We have now been successful at using the Joint UKF to perform state and parameter estimation for the Predator Prey model. We will now explore using the Dual UKF to perform the same task. The Dual UKF runs two filter simultaneously, one for the unknown states, which in this case are the predator and prey populations, and another for the unknown parameters, which are α, β, γ, and δ. The general process is as follows:

1. Initialize necessary covariance matrices **both** for the states and parameters
2. Run the parameter filter using output from state filter (or initial condition if first iteration)
3. Run state filter using output from parameter filter
4. Run steps 2) and 3) for t = 1 to T.

Let us more closely examine each of the state and parameter filters.

2.1 The State Filter

The state filter works exactly as it did when we were performing **only** state estimation for the Predator - Prey model. The only difference is that instead of running all iterations of the state filter at once, we are now running the parameter filter in between each iteration. The state filter takes as input the parameters for the model, which are taken from the most recent output of the parameter filter. The state filter outputs an estimate for the predator and prey populations at time t, which are inputted to the parameter filter. Thus, while the state filter is running, parameters are kept **constant** and while the parameter filter is running, states are kept **constant**.

2.2 The Parameter Filter

The parameter filter is a new concept and is a slight modification of the state filter, the equations for which we will now describe. The equations which describe the parameter filter are:

$$w_k = w_{k-1} + v_{k-1} \quad (1)$$

$$y_k = h(f(\hat{x}_{k-1}; w_k), w_k) + \epsilon_k \quad (2)$$

Where the parameters are w_k, the most recent state estimate is $\hat{x}_{k-1}$, the process noise is v_k with Covariance Q_{w_k}, and the measurement noise is ϵ_k with covariance R_{w_k}. In equation 1, it is important to note that there is no nonlinear transformation applied to the parameters, like there is for the states. Instead, parameters stay the same from one iteration to the next (since they are meant to be constant values) apart from the process noise.

Next, in equation 2, the same observables are used in y_k as are for the state filter. Similarly, the measurement noise is thus the same and the same values for the measurement noise should be used between the state and parameter filter. However, the key difference is in $h(f(\hat{x}_{k-1}; w_k), w_k)$. The term $f(\hat{x}_{k-1}; w_k)$ is calculating an estimate for the unknown states at time t based on the parameters we've estimated in w_k. Then, the result of this estimate is passed to the measurement function. Since the state $\hat{x}_{k-1}$ is being held constant, this process thus tests the quality of the parameters. Now, using this setup we can describe the parameter filter equations: first, we initialize with:

$$\hat{w}_0 = E[w]$$

$$P_{w_0} = E[(w - \hat{w}_0)(w - \hat{w}_0)^T]$$

Next we create a set of sigma points for the parameters:

$$W_{k|k-1} = [\hat{w}_{k|k-1} \ \hat{w}_{k|k-1} + \sqrt{(n_x + \lambda)P_{w_{k|k-1}}} \ \hat{w}_{k|k-1} - \sqrt{(n_x + \lambda)P_{w_{k|k-1}}}]$$

Once we have a set of sigma points, we can use the measurement equations as follows:

$$Y_{k|k-1} = h(f(W_{k|l-1}; \hat{x}_{k-1|)})$$

Here, we are applying the transition function f and then the measurement function h to **each** of the parameter vectors in the set of sigma points W while keeping the state constant. This gives us the result Y which is a set of observables based on the estimated data. We now use our weighting scheme and estimates in Y to produce first the weighted mean and covariance:

$$\hat{y}_{k|k-1} = \sum_{i=0}^{2n_x} w_i^{(m)} Y_{i,k|k-1}$$

$$P_{y_k} = \sum_{i=0}^{2n_x} w_i^{(c)} (Y_{i,k|k-1} - \hat{y}_{k|k-1})(Y_{i,k|k-1} - \hat{y}_{k|k-1})^T + R_{w_k}$$

We can then also compute the covariance between the observables and the parameters:

$$P_{w_k y_k} = \sum_{i=0}^{2n_x} w_i^{(c)} (W_{i,k|k-1} - \hat{w}_{k|k-1})(Y_{i,k|k-1} - \hat{y}_{k|k-1})^T$$

This now allows us to compute the Kalman Gain:

$$K_k = P_{w_k y_k} P_{y_k}^{-1}$$

And finally the posterior estimates for the parameters and parameter covariance:

$$\hat{w}_k = \hat{w}_{k|k-1} + K_k(y_k - \hat{y}_{k|k-1})$$

$$P_{w_k} = P_{w_{k|k-1}} - K_k P_{y_k} K_k^T$$

Now, these estimates are both used in the next run of the parameter filter and the output w_k is also used and held constant during the next state estimation run.

As mentioned earlier, the measurements used for the state filter and parameter filter are the same. Thus the covariance matrix R_{w_k} should be set the same as it is for the states. However, the process noise here is different. From reading of Gove and Hollinger, they provide the following equation for setting Q_{w_k}:

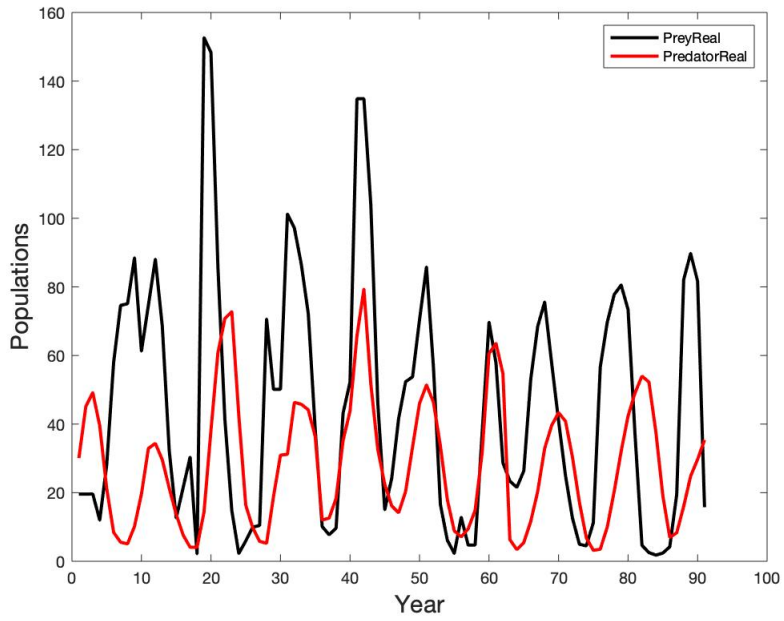
$$Q_{w_k} = ((1/\lambda) - 1)P_{w_k}$$

Where λ is value between 0 and 1. For use in the Predator Prey model, we've found that $\lambda = .99$ works best, which translates to:

$$Q_{w_k} = .01P_{w_k}$$

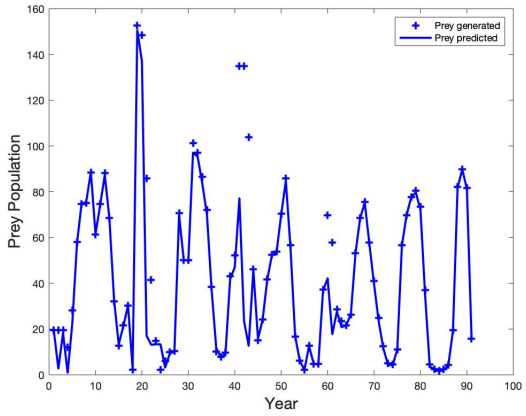
3 Results

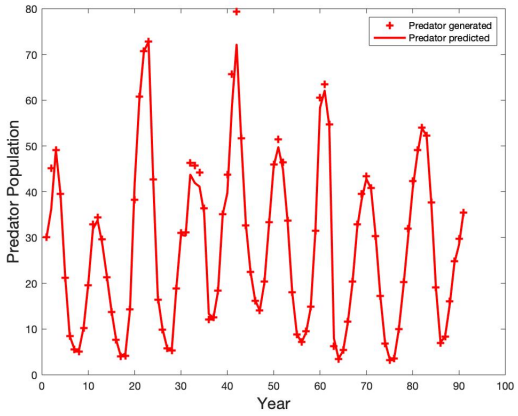
Now having the conceptual and theoretical understand for the Dual UKF, we can apply it to the Predator Prey model. As a reminder, the predator prey system of Hares and Lynx produces the following data:



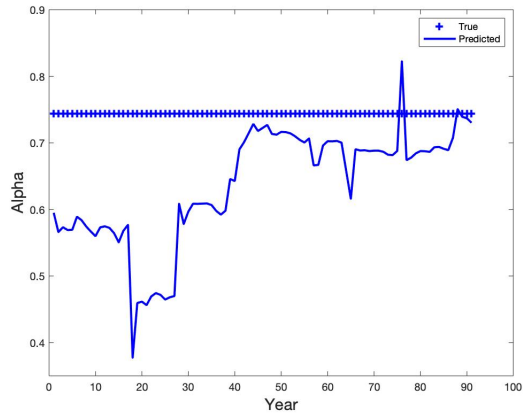
Now, first looking at our prey estimates overlayed with the real data (blue) and

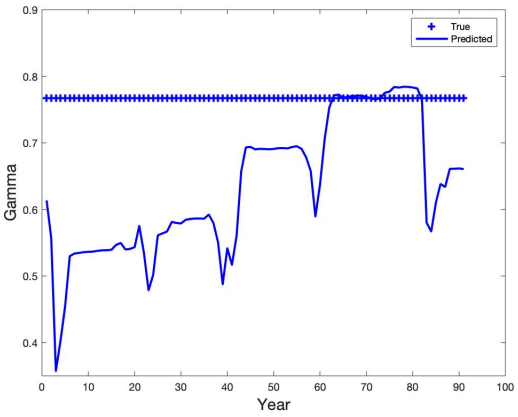
the predator estimates overlayed with the real data (red) we get:

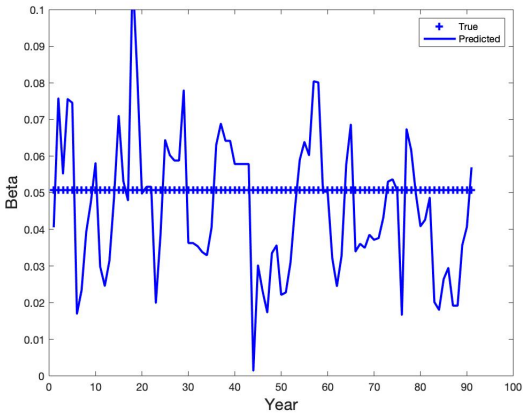


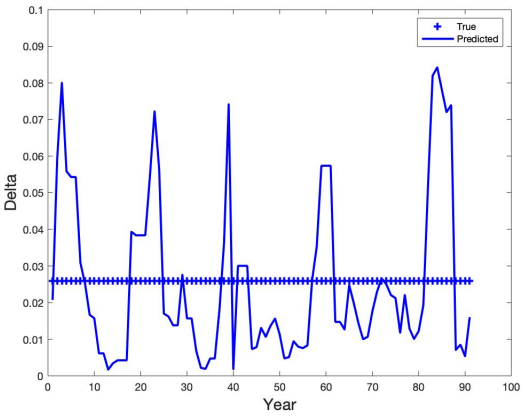


We can see that both for predator and prey we are able to do a good job of estimating the states. Next let's look at our resultant parameter estimates:









Looking at alpha and gamma first, we see that both are making steady climbs to their grue values and appear to be converging when they get there. It is true that we cannot be certain if they would move more given more data but overall this is satisfactory. For beta and delta, similarly to the joint estimation, they oscillate within a small range. The fact that the range of oscillation is small is ensuring to see, but we would like to see some sort of convergence in these values.

3.1 Covariance Matrices

In working with the Dual Estimation Algorithm, we have found that the algorithm is very sensitive to initializations of covariance matrices. Specifically, the initialization for P_{w_0} and Q_x proved to be crucial. Through tuning the parameters we arrived at the results of:

$$P_{w_0} = \begin{bmatrix} 0.5 & .001 & .001 & .001 \\ .001 & 0.5 & .001 & .001 \\ .001 & .001 & 0.02 & .001 \\ .001 & .001 & .001 & 0.02 \end{bmatrix}$$

The .001's here allow for non-zero covariance between parameters but ensure that they move "for the most part" independently. Now, the 0.5's in the first two spots along the diagonal are the variance of alpha and gamma respectively. These values are, in general, larger than their counterparts delta and beta, which is why they are set to 0.5 as opposed to 0.02 for beta and delta.

$$Q_x = 1000 * \begin{bmatrix} 1.3444 & 0.1594 \\ 0.1594 & 0.4845 \end{bmatrix}$$

The need to set Q_x to such a large value is the most interesting finding thus far. For the Dual UKF, I chose to analytically determine the process noise Q_x as follows:

1. Calculate the noise as the difference between the real data and the data generated by running the Lotka-Volterra Equations

2. Create a covariance matrix between the row representing the prey noise and the row for the predator noise to use for Q_x.

The following code snippet achieves this:

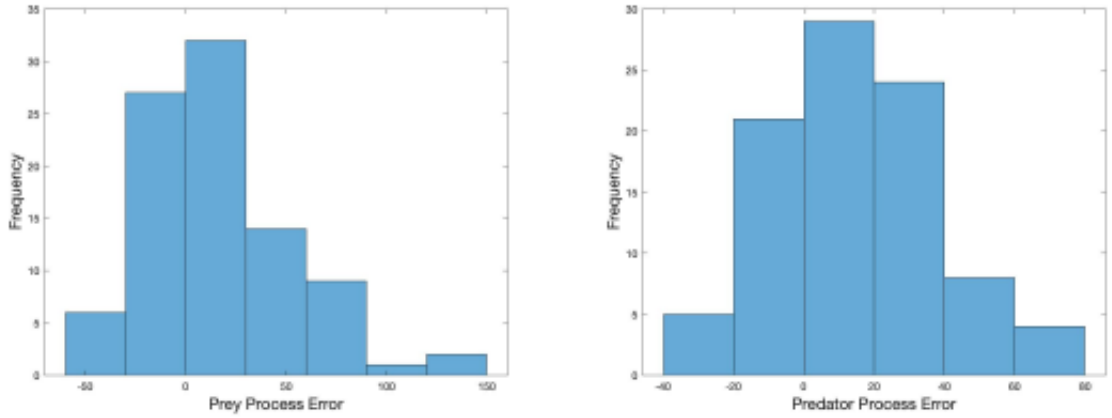
```
HLData = load('HaresLynxData.mat');  %Load dataset
rawData = HLData.Lotka_Volterra_Data;
x(1:2,:) = rawData(:, 2:3)'; %The real data

T = 91;
x_fake = zeros(Nx,T);
x_fake(:,1) = [x(1,1); x(2,1)];
tspan = [0 T];
sol = ode45(@(t, y) Lotka_Volterra_Model(t, y, truepar), tspan, x_fake(:,1)); %Use ODE solver
for t=2:T %Generate data for each time point
  x_fake(:,t) = deval(sol, t); %Get ODE solution for time t
end

prey_noise = x(1,:) - x_fake(1,:); %Difference between real and ODE
predator_noise = x(2,:) - x_fake(2,:);
cov(prey_noise, predator_noise) %Print covariance matrix
```

This choice of Q improved performance when used for the Dual. As a result, I initially went back to the Joint implementation to test Q in that context. However, this completely ruined the Joint UKF, a very surprising consequence. I believe this is where the benefit of the Dual UKF is most noticeable. By **separating** the noise of the states from the noise of the parameters, we have more control over the amount of noise the algorithm expects.

Running this code also gives us an opportunity to look at the shape of the process noise. If you recall from our previous discussions of kalman filters, they operate under an assumption of Gaussian noise with mean 0 and some covariance. Looking at histograms of the noise values gives us the following:



As expected, the noise appears relatively Gaussian, with the Prey noise being slightly right skewed. Moreover, Prey noise has mean 16.12 and Predator noise has mean 13.5241. Of course, these are not 0, which would be ideal, but the shape of the noise does support our assumption of the Gaussian noise setup.

4 Comparison of the Joint and Dual UKF's

We now have two working UKF implementations for the Lotka-Volterra model: a joint and dual UKF. We would now like to compare the two to understand which is performing better in this context. Visually speaking, the two algorithms appear to perform similarly well, thus we will need to define some sort of numeric criteria to assess the algorithm's quality. To do this we will use the following six values

1. Final 50% MSE for Prey population estimates
2. Final 50% MSE for Predator population estimates
3. Difference between final Alpha prediction and true value
4. " " Beta " "
5. " " Delta " "
6. " " Gamma " "

The reasoning behind using only the final 50% MSE as opposed to the entire timeframe is to allow the UKF algorithms some time to learn before judging

them. The values for all 6 criteria are presented below:

	Dual UKF	Joint UKF
Final 50% Prey MSE*	50.7984	51.3580
Final 50% Predator MSE*	0.6321	19.2451
Alpha Error	0.0134	0.0177
Beta Error	.1064	0.0001
Delta Error	.0062	0.0116
Gamma Error	.0099	0.1005

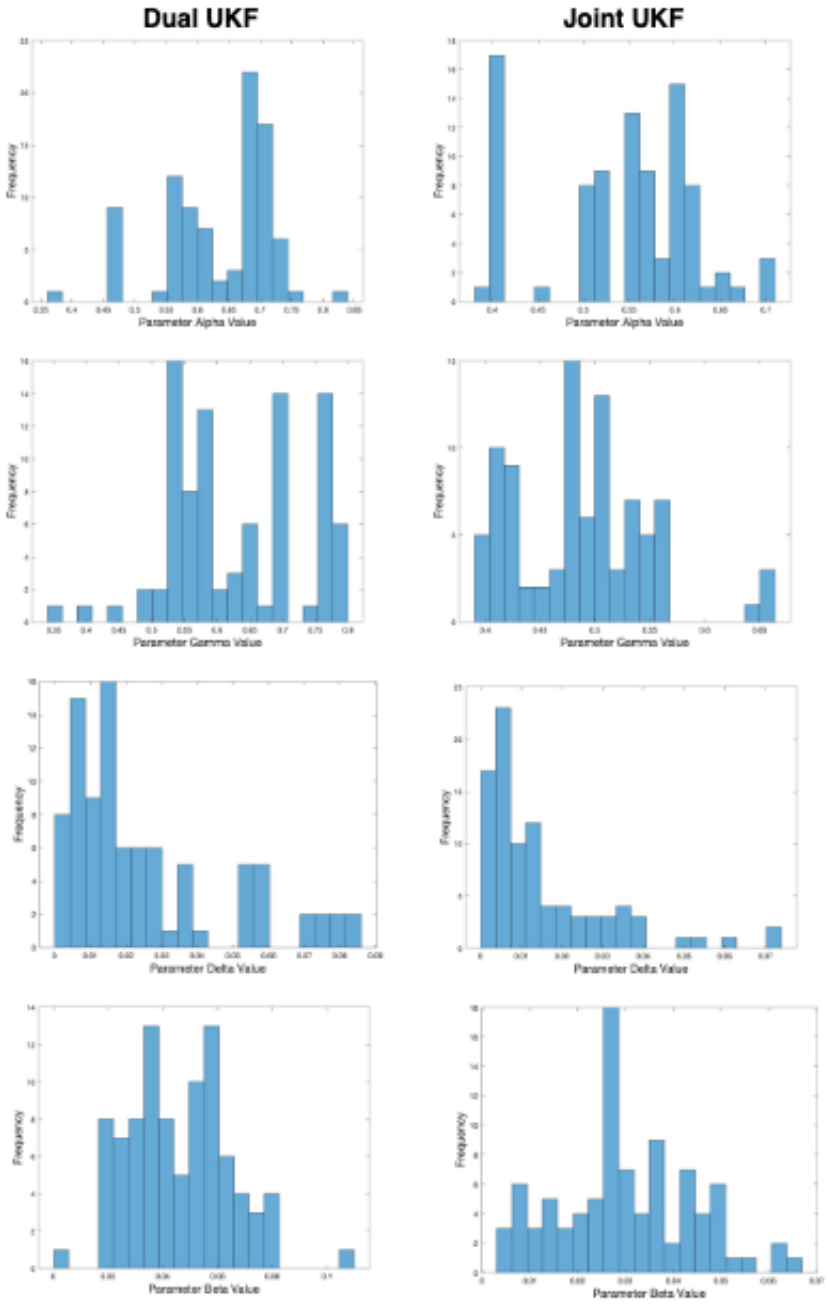
Note that the better values are those that are smaller and highlighted in green.

Apart from Beta Error, the Dual UKF appears to out perform the Joint UKF in all categories. As a result, we currently have more faith in the Dual UKF for use in the Predator Prey system.

Both algorithms have clear benefits depending on the scenario. The joint UKF allows for covariances between states and parameters, while the dual separates all covariances, most notably for noise, allowing for more control over their values. In this scenario, the advantages the dual UKF present appear more valuable. However, when moving to new models, most notably the T1D model, there is no guarantee that the same will be true once again.

We also now have an opportunity to look at histograms of the parameter values we estimate at each iteration. An important caveat is that we are not looking for a particular distribution here, but are just interested in how much time the algorithm spends at particular parameter values. The UKF is a process that looks at data **as it comes in**. This means that, since the algorithm does not have all the data available to it right away, we should not expect for it to find the parameter values right away. However, histograms of parameter values are

still helpful:



As stated, we do not see clear distributions across these histograms. What is most notable, however, is that, for each parameter, there appears to be a few values where the parameter prefers to be. These are denoted by the various peaks in the histogram. It is likely that these represent local optima where the parameters tend to "get stuck" on their way to the final estimates.

5 T1D State Estimation Introduction

We are now moving on to working with the Type 1 Diabetes model. We are starting with state estimation using generated data. We are using our State Estimation Code from a couple weeks ago. We ran this code for four different cases: wild-type mouse with no apoptotic wave, wild-type mouse with apoptotic wave, NOD mouse without apoptotic wave, NOD mouse with apoptotic wave.

6 Model

The model that we are working with is the T1D model from the original paper. This is a single compartment model of the pancreas which contains 12 differential equations. These equations represent the states of this model. The only observable for this model is glucose, which is also one of the states. There are a large number of parameters for this model, for the state estimation, they were taken from the files given.

7 Implementation

In order to implement the state estimation, it was first necessary to make sure that every reference to the number of states or observables was set to the correct number (12 states and 1 observable). I added a line to run the document that defined all the parameters, and removed all other references to parameters. I needed to change the measurement function so that it took in a vector of states and returned a vector of observables. In this case, this was very simple since I only needed to take in the states and return the specific one that referred to glucose.
After the program ran, we then had to mess with some of the filter parameters to make sure the program ran as smoothly as possible. This was mostly done with trial and error. Ideally we would find a more systematic way to do this, but especially given how long this takes to run, we can't run that many variations. The current values of the parameters are:
α = 10e-4
β = 2
κ = 0
P_0 (initial covariance matrix) is a matrix of ones
Q (process noise matrix is) diag(diag([repmat(10e-2,Nx)]))
R (measurement noise) diag(5)

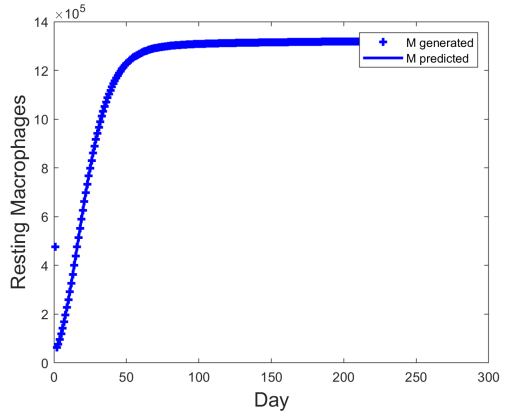
It may be necessary to refine some of these parameters, most specifically R when we run this on real data

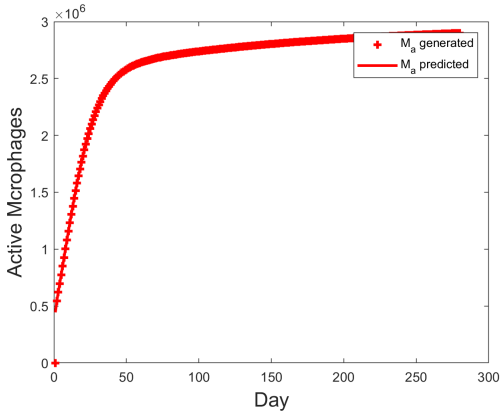
8 Results

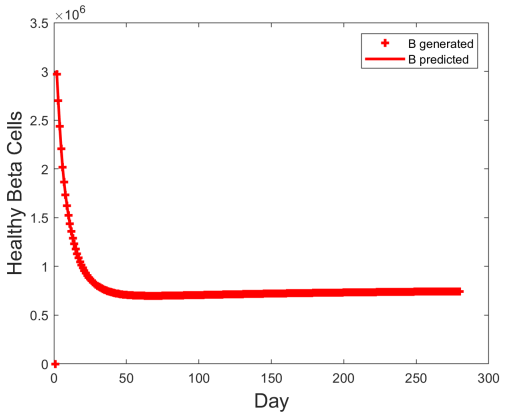
Below are the results of the model for each case. We show the discrepancies between all the generated and predicted states.

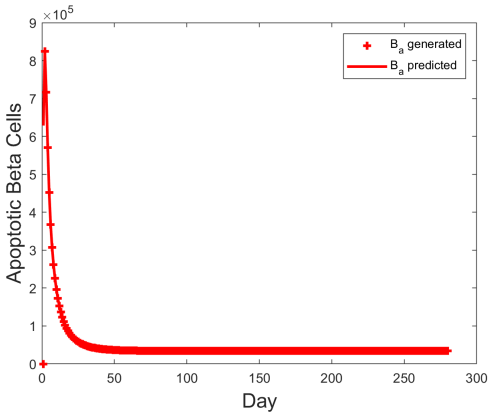
8.1 Wild Type Mouse

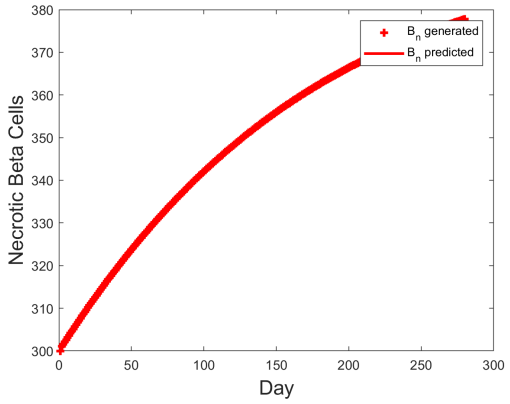
8.1.1 Without Wave

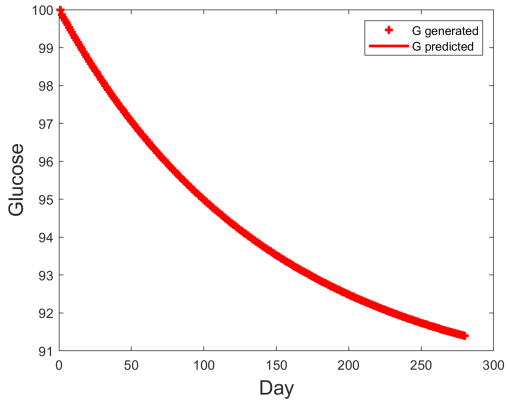


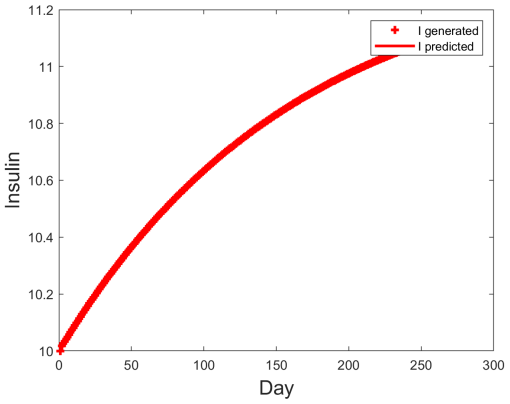


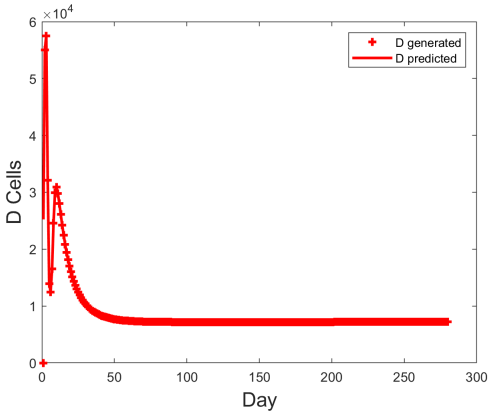


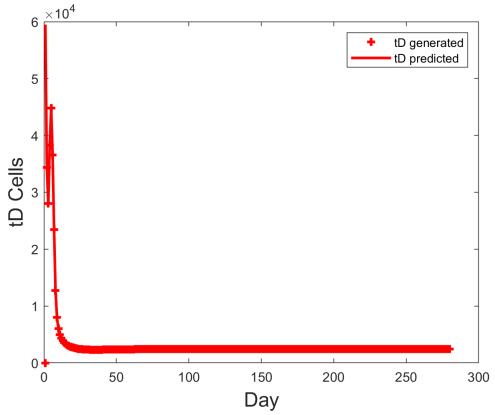


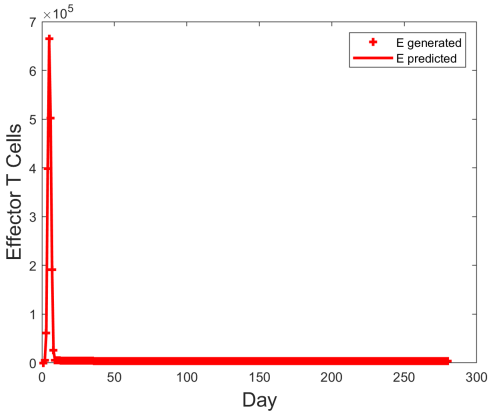

Necrotic Beta Cells
Day
B_n generated
B_n predicted

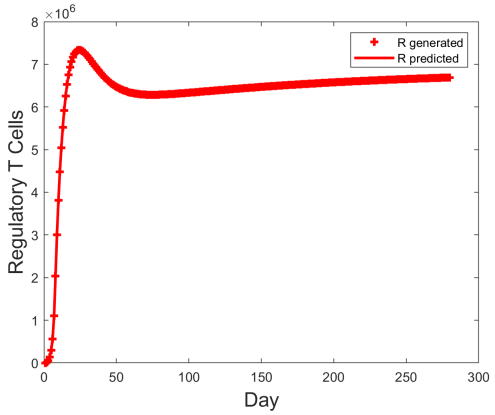

Glucose
Day
G generated
G predicted

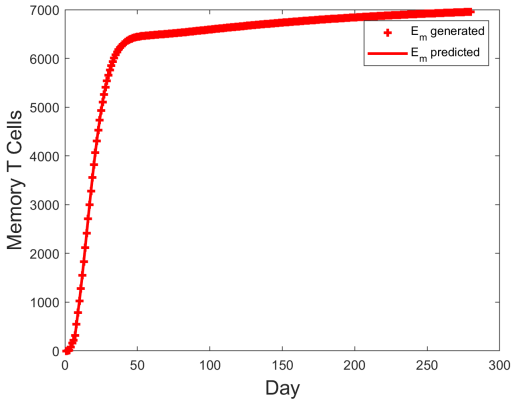

Insulin
Day
I generated
I predicted


×10^4
D Cells
Day
D generated
D predicted

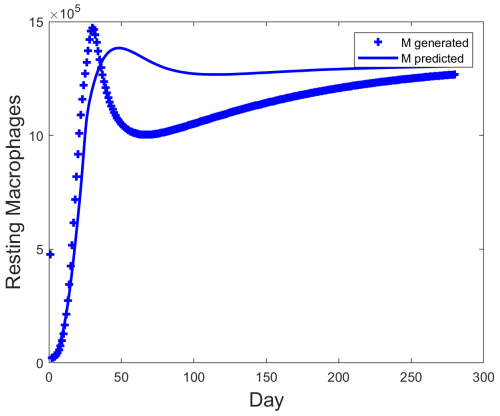

×10^4
tD Cells
Day
tD generated
tD predicted

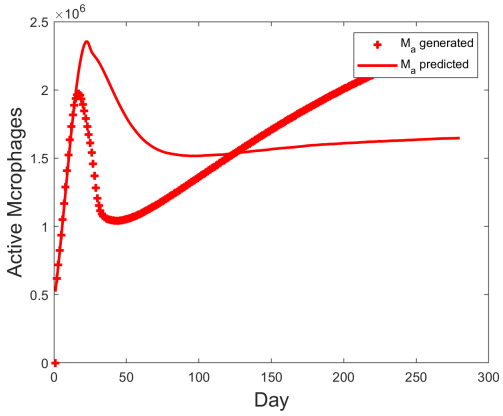

×10^5
Effector T Cells
Day
E generated
E predicted

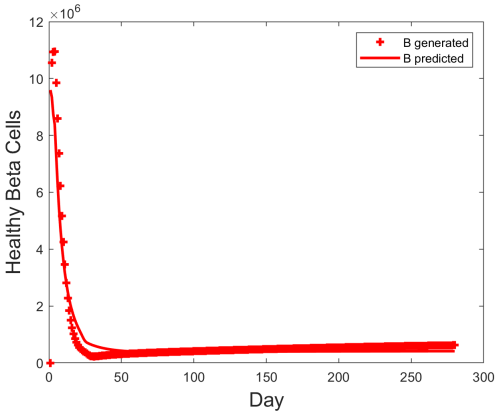


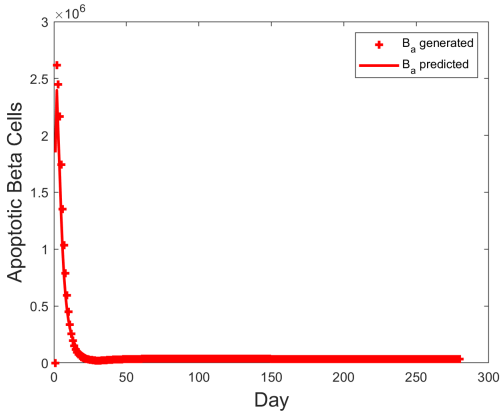


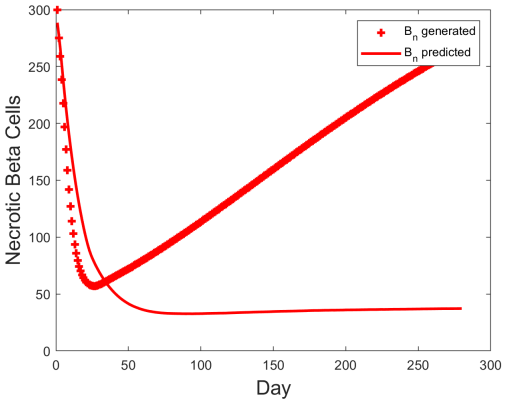
8.1.2 With Wave

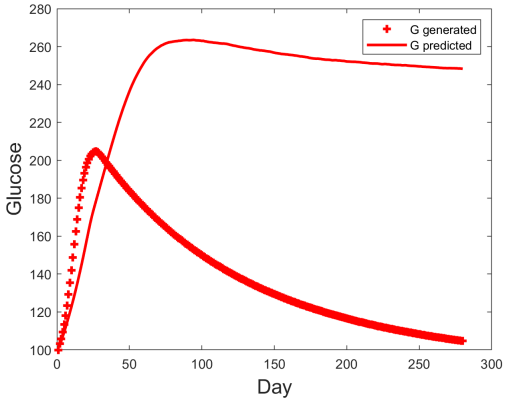


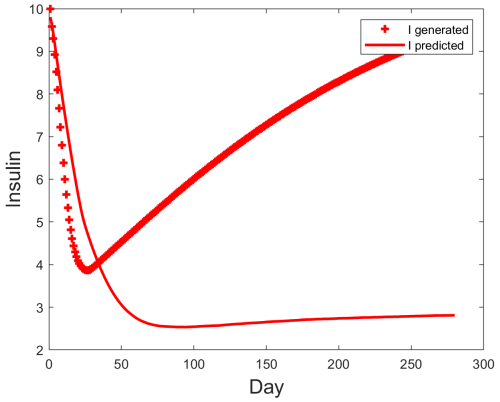


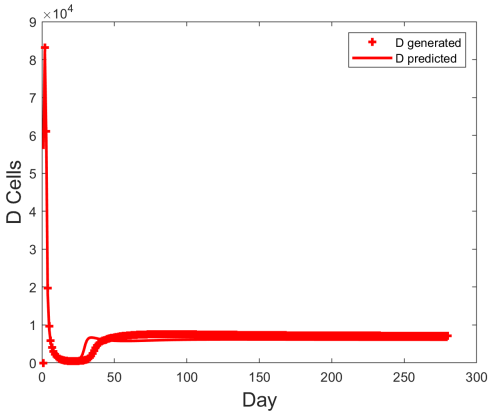


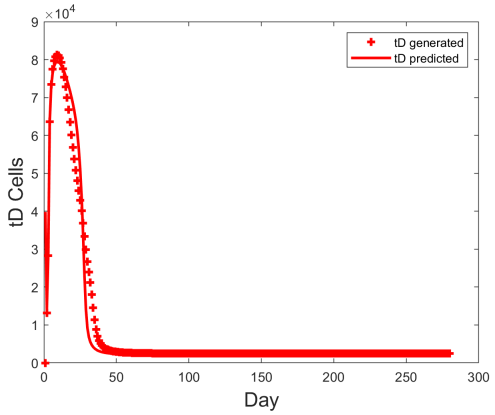


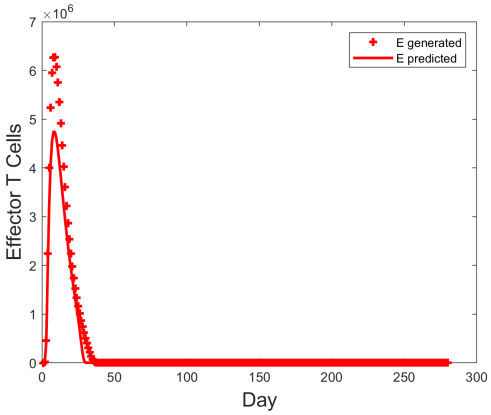

Necrotic Beta Cells
Day
B_n generated
B_n predicted

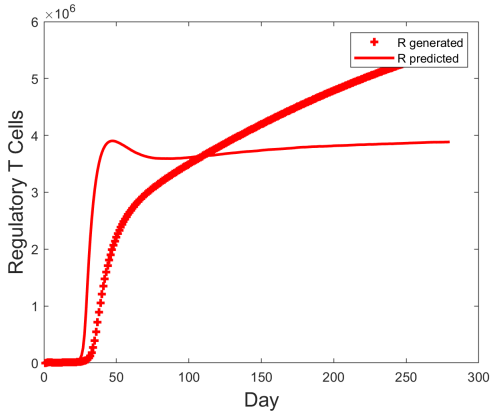

Glucose
Day
G generated
G predicted

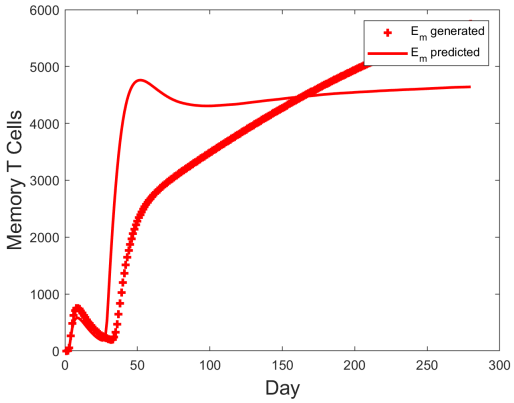

Insulin
Day
I generated
I predicted


×10^4
D Cells
Day
D generated
D predicted


×10^4
tD Cells
Day
tD generated
tD predicted

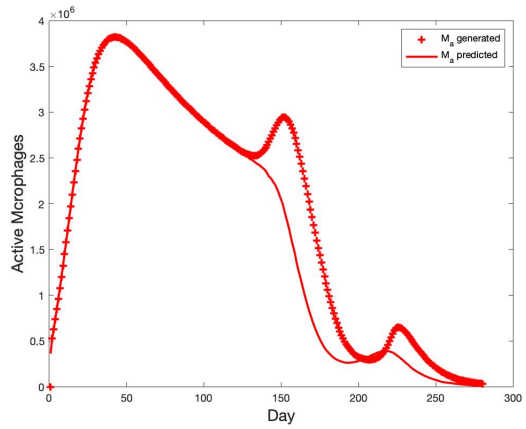

×10^6
Effector T Cells
Day
E generated
E predicted

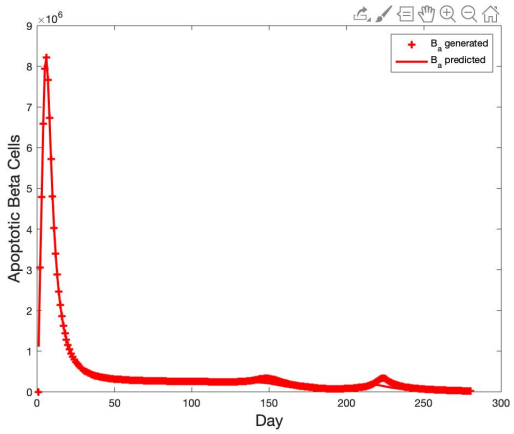


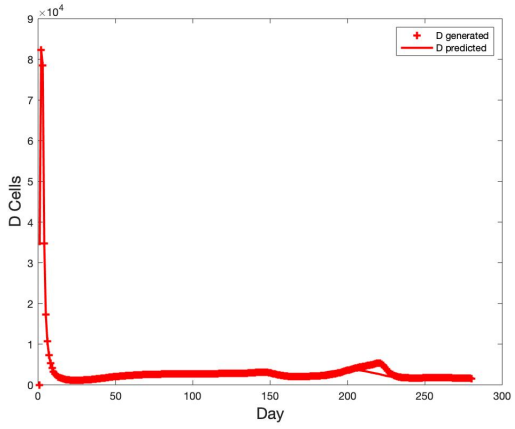


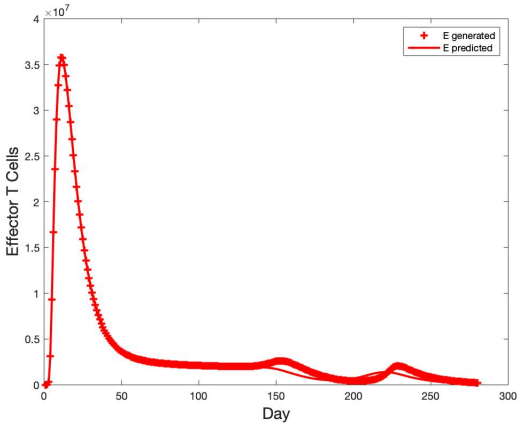
8.2 NOD Mouse

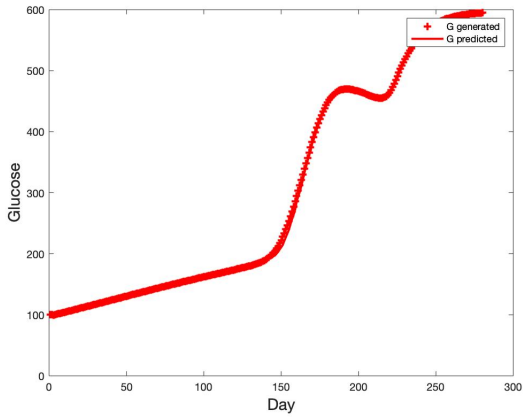
8.2.1 Without Wave

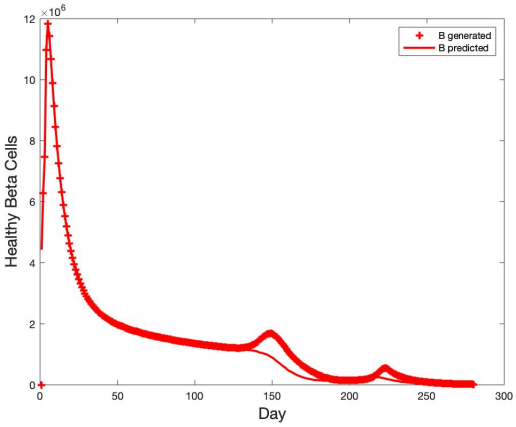


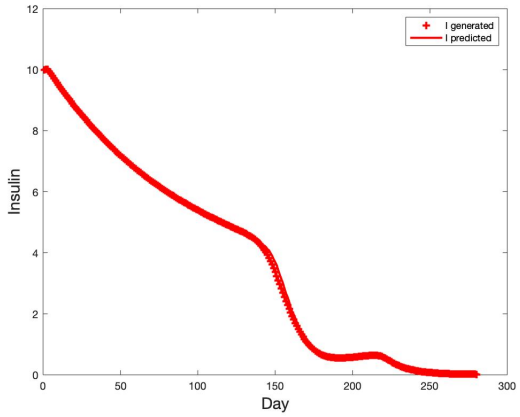


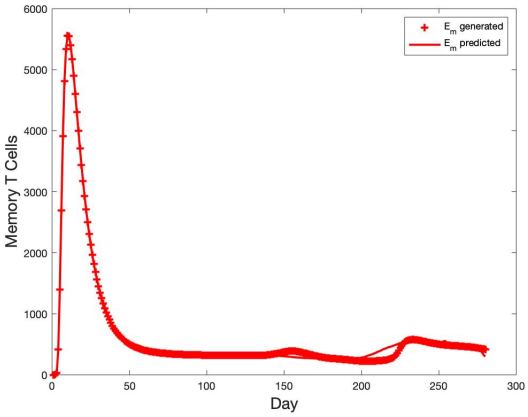


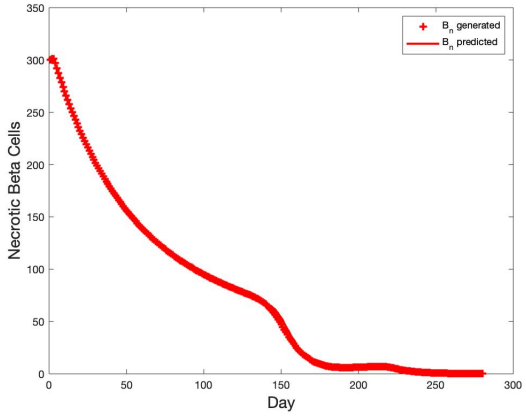


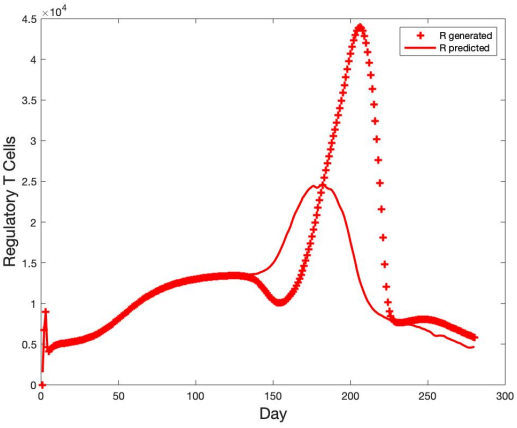

G generated
G predicted
Glucose
Day

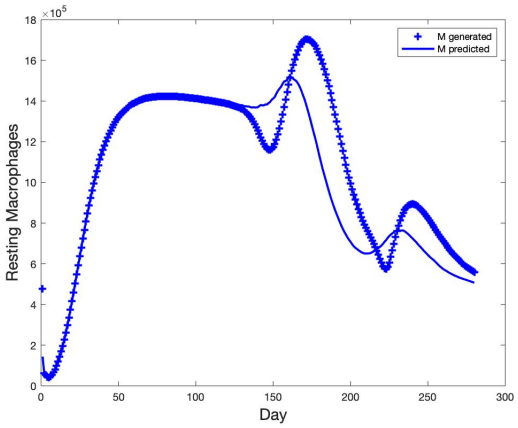

B generated
B predicted
Healthy Beta Cells
Day

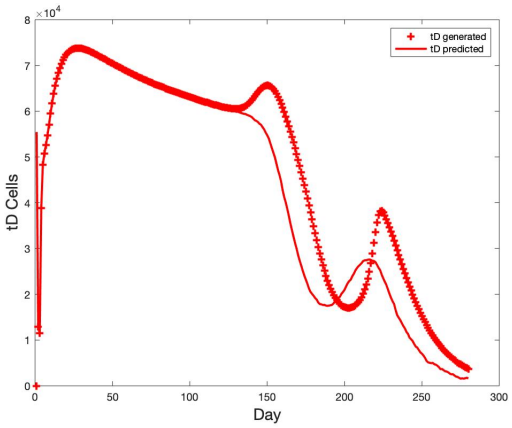

I generated
I predicted
Insulin
Day


E_m generated
E_m predicted
Memory T Cells
Day

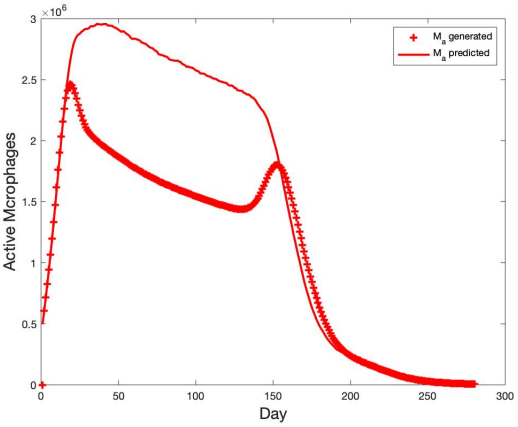

B_n generated
B_n predicted
Necrotic Beta Cells
Day

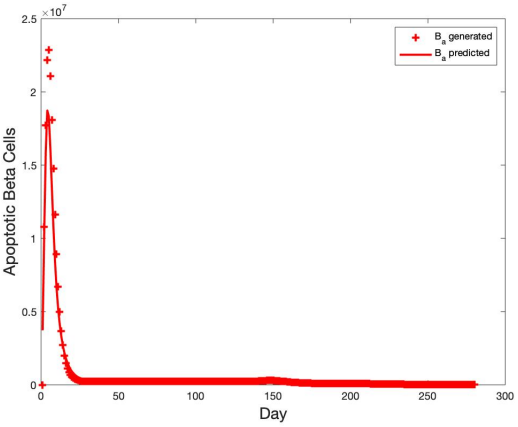

R generated
R predicted
Regulatory T Cells
Day

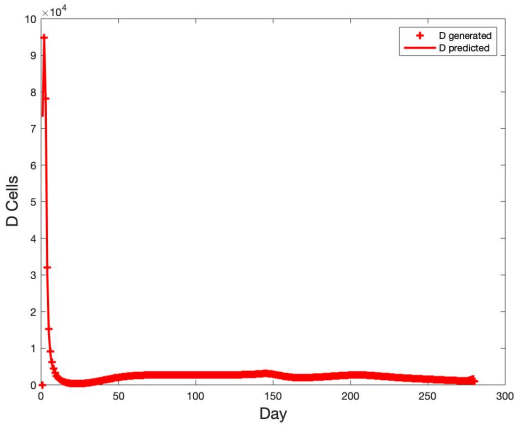


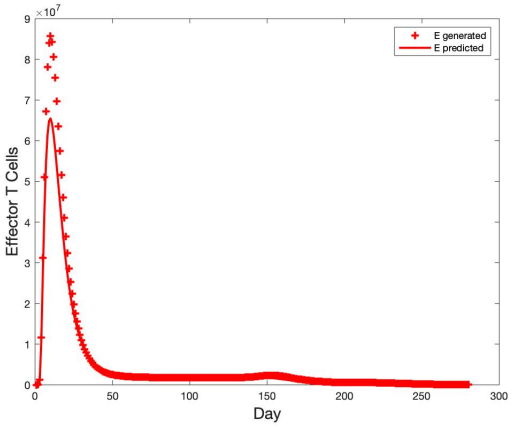


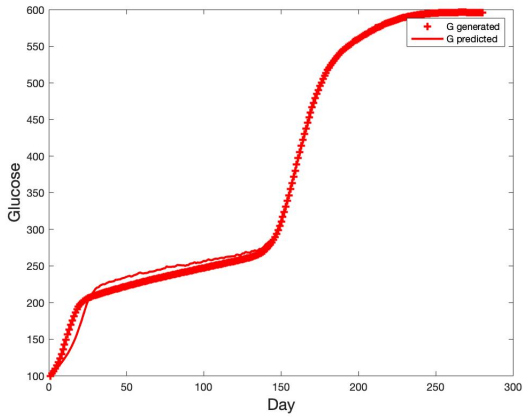
8.2.2 With Wave

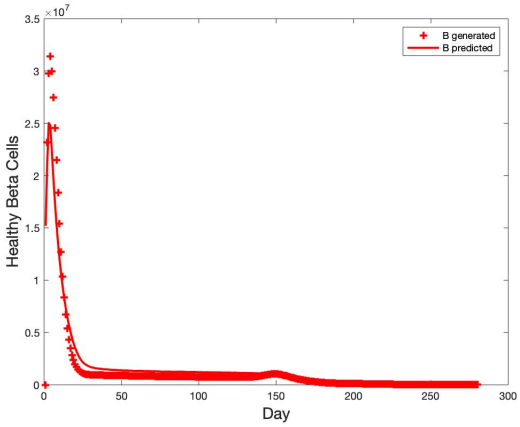


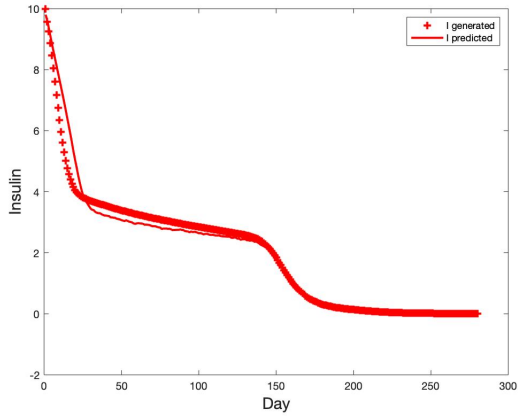


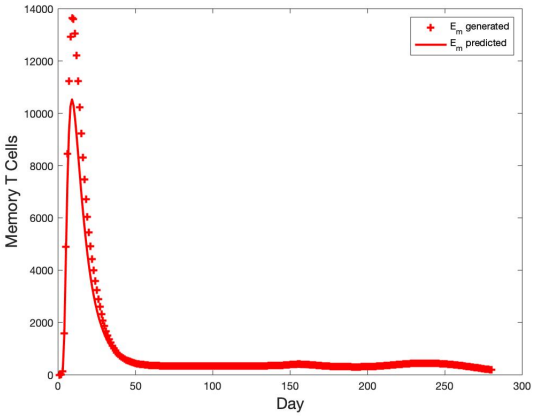


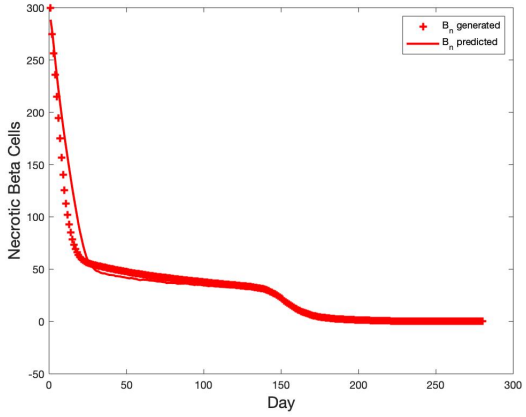


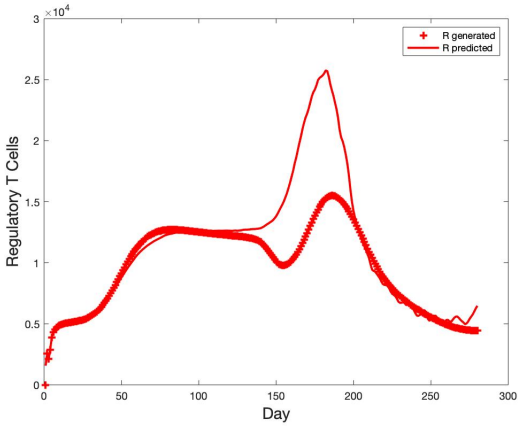

G generated
G predicted
Glucose
Day

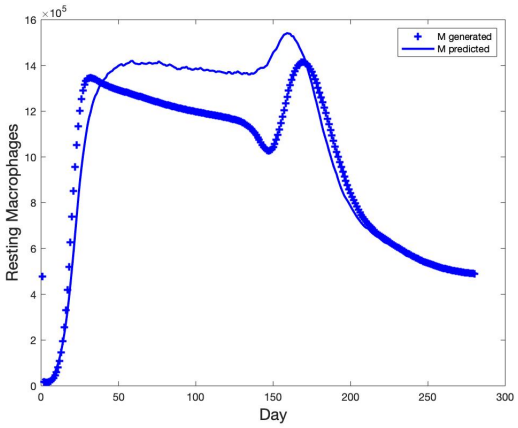

×10^7
B generated
B predicted
Healthy Beta Cells
Day

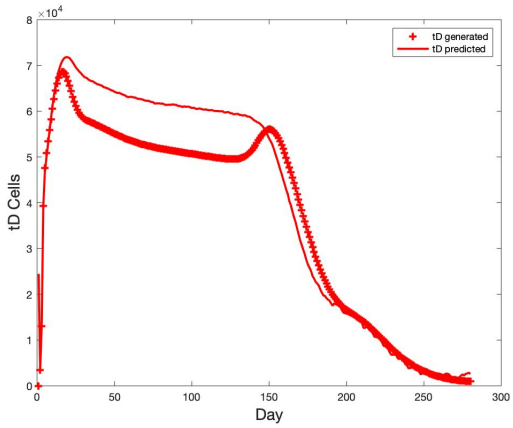

I generated
I predicted
Insulin
Day


E_m generated
E_m predicted
Memory T Cells
Day


B_n generated
B_n predicted
Necrotic Beta Cells
Day


×10^4
R generated
R predicted
Regulatory T Cells
Day





9 Takeaways

In the absence of a major shift in the states such as the apoptotic wave, state estimation works slightly better with a more stable system like the wild-type mouse as opposed to a more volatile system such as the NOD mouse. When something such as the apoptotic wave is introduced, it causes some states to shift quickly; if the filter does not catch this it can reach a different steady state. The sharper the shift, the more likely the filter is to miss it and reach a different steady state

10 Next Steps

Run the estimation with real data:
Right now, we are running this only on generated data. The next step is to run it on the actual data. Parameter Estimation: The other important next step is to transition to joint parameter and state estimation using both the joint and dual Kalman filters.