Use of Particle Filtering to Clarify Impure Models and Data

Erik Bergland Zhaoshu Cao Arum Lee Richard McQueen
Olaoluwa Ogunleye Sameer Pokhrel Yumeng Wang Han Yong Wunrow
Pietro Zanin

June 10, 2023

Abstract

Filtering is a process that combines predictions from dynamic models of a system with noisy observational data to estimate the true state of the system. It addresses the discrepancies between the model and the data, which can occur due to missing features or errors in the observation process. One of the most well-known and efficient solutions is the Kalman filter, which requires that the dynamics are linear and both model and observational errors follow uncorrelated Gaussian noise. In more general situations, filtering methods can be formulated using Bayesian inference. The challenge then becomes how to implement these Bayesian updates in a way that is computationally efficient. One popular approach is particle filtering, which is widely used in science, applied math, and engineering for sequential signal processing and has gained significant attention from researchers in various fields. Such filters characterize the uncertainty in the dynamical state using a distribution of particles with probabilistic weights. They are particularly valuable when dealing with nonlinear and non-Gaussian problems. In our work, we have utilized particle filtering to address problems of this nature. We demonstrate the effectiveness and practicality of this approach through simulations.

1 Introduction

In many cases, it is essential to address the discrepancies between the output of a model and the observed data. Often, models are missing key features or the observation process contains errors. Thus, neither approach is wholly satisfactory on its own. In some cases, having knowledge about a system is essential before analyzing or designing applications related to it. However, obtaining the true state of the system directly might not be possible. Instead, we need to rely on measuring the system's output using instruments like sensors, which are generally noisy. By combining these measurements or data with an approximation of the system's model, known as the naive model, we can infer or estimate the system's true state. If the system is static, then simple methods like least square estimation can work. For a dynamical system with a real-time estimation of the true states, more sophisticated algorithms need to be applied. In many applications, it is common to assume that the process noise and measurement noise follow a Gaussian distribution, or can be approximated by one. Similarly, since the exact state is unknown, the state estimate is treated as a random variable, and observations of the state are assumed to be Gaussian random variables centered at the true value. This assumption makes the design of estimation filters much easier, and it forms the foundation of the Kalman filter family. However, in cases where the distribution is not Gaussian, we need a different method that can handle any type of distribution. In the work to follow, we focus on one such method: particle filtering.

Particle filtering is a sequential Monte Carlo method for tracking the state of a dynamical system over time, from noisy measurements. It is a powerful tool for a wide range of applications, including robotics, computer vision, and natural language processing. The basic idea of particle filtering is to represent the posterior distribution of the state at each time step as a set of weighted samples, called particles. The particles are propagated through time using a state transition model, and their weights are updated using a measurement model. The filter then selects the most likely particle at each time step, which can be used to estimate the state of the system. The state-space model can be nonlinear and the initial state and noise distributions can take any form required. For implementation, the particle filter is a relatively simple algorithm to implement, but it can be computationally expensive, especially for high-dimensional state spaces. However, there are a number of techniques that can be used to reduce the computational cost, such as importance sampling and resampling. Particle filter techniques provide a well-established methodology for generating samples from the required distribution without requiring assumptions about the state-space model or the state distributions.

The report will be organized as followed. Section 2.1 reviews the theoretical underpinnings of particle filtering. The rest of the report discusses three main aspects of the project: modeling and data generation, computation, and theoretical research. Each task was investigated by several group members. The Modeling and Data-Generation group (P. Zanin, H.Y. Wunrow, O. Ogunleye, E. Bergland, S. Pokhrel) generated test models and observational data for the purpose of testing the performance of the classical particle filter on model with moderate complexity. The models will be discussed in detail in section 2.2. The Computational group (H.Y Wunrow, Z. Cao, R. McQueen, Y. Wang, A. Lee, S. Pokhrel) implemented the bootstrap particle filtering algorithm and tested it on the data provided by the Modeling team. The simulation results will be discussed in section 3. Theoretical research was done by P. Zanin, E. Bergland, A. Lee, R. McQueen, S. Pokhrel to investigate more advanced variants of the particle filtering method, such as the ensemble Kalman filter and the particle flow method. The latter will be discussed in section 4. This will help our group gain an understanding of the problem proposed by Raytheon Technologies at MPI.

2 Background and preliminaries

2.1 Theoretical Basis for Particle Filtering

The particle filter algorithm is a *recursive* algorithm, which means that it can be used to estimate the state of the system over time. The algorithm is also *online*, which means that it can be used to estimate the state of the system as new measurements become available. Particle filtering is a powerful method for estimating the state of dynamic systems from noisy measurements. It is robust to outliers and can handle nonlinear systems. Particle filtering has been used successfully in a wide variety of applications, including weather forecasting, air traffic control, and robotics.

The particle filtering algorithm works as follows:

Algorithm 1 Particle Filtering Algorithm

```
1: Initialize a set of particles, where each particle represents a possible state of the system.
2: For each particle, predict the state of the system using a dynamic model.
3: For each particle, weight the particle according to its likelihood given the measurements.
4: Resample the particles, keeping the particles with the highest weights.
5: Repeat steps 2-4 until the desired accuracy is reached.
```

2.1.1 Particle Filtering as Data Assimilation

Particle filtering can be understood as a particular technique to address data assimilation: the problem of using (noisy) measurements of a system to correct the predictions of an imperfect model. To begin, suppose we wish to understand the behavior of a dynamical model with state vector $X(t) \in \mathbb{R}^m$. We suppose that we possess a probability density called the model-based probability transition kernel, denoted by $\pi_{X(t')|X(t)}(x'|x)$. This is the probability density associated with transitioning to state x' at time t' conditioned on beginning in state x at time t. In addition to this quantity, we suppose that we have another density $\pi_{Y_k|Z_k}(y_k|z_k)$. Here, Y_k represents a (noisy) observation of the system at time t_k, and Z_k represents the true state of the system. Ultimately, we wish to compute $\pi_{Z_k|Y_{1:k}}(z|y_{1:k})$. In other words, we wish to predict the probability that the system is in true state z after making the set of observations $y_1, y_2, ..., y_k$. In order to do so, we make use of the Bayes rule:

$$\begin{aligned}
\pi_{Z_{k+1}|Y_{1:k+1}}(z|y_{1:k+1}) &= N_{k+1}^{-1}\pi_{Y_{k+1}|Z_{k+1},Y_{1:k}}(y_{k+1}|z, y_{1:k})\pi_{Z_{k+1}|Y_{1:k}}(z|y_{1:k}) \\
&= N_{k+1}^{-1}\pi_{Y_{k+1}|Z_{k+1},Y_{1:k}}(y_{k+1}|z, y_{1:k})\int_{\mathbb{R}^m}\pi_{Z_{k+1}|Z_k,Y_{1:k}}(z|z_k, y_{1:k})\pi_{Z_k|Y_{1:k}}(z_k|y_{1:k})dz_k \\
&= N_{k+1}^{-1}\pi_{Y_{k+1}|Z_{k+1}}(y_{k+1}|z)\int_{\mathbb{R}^m}\pi_{Z_{k+1}|Z_k}(z|z_k)\pi_{Z_k|Y_{1:k}}(z_k|y_{1:k})dz_k
\end{aligned} \tag{1}$$

where N_{k+1}^{-1} is a normalization factor. In the second line, we have made use of the law of total probability. In the third line, we have also used the fact that our observations are independent to simplify the prefactor in front of the integral. The right-hand side is now entirely written in terms of quantities we know, and we may use it to compute the desired posterior distribution recursively. However, several pragmatic challenges immediately present themselves. First and foremost is the integral term: except in the case of extremely simple (e.g. Gaussian) distributions, this integral will be impossible to compute analytically. This difficulty is neatly addressed in particle filtering: first, we assume that our posterior distribution is well-approximated by a sum of delta functions with appropriate weights:

$$\pi_{Z_k|Y_{1:k}}(z|y_{1:k}) \approx \sum_{j=1}^{N} w_k^{(j)}\delta(z - z_j) \tag{2}$$

where the w_j sum to one. We refer to each z_j as a particle position. The integral can now be interpreted to be evolving the particles from positions representing the distribution at time t_k to that at time t_{k+1}, which can be accomplished simply by simulating a trajectory of our dynamical model with each particle position as an initial condition. This is referred to as the forecasting step. Once forecasting has been completed, we begin the analysis step. Here, we must make use of the remaining terms on the right-hand side to arrive at a new approximation written in terms of delta functions. This can be accomplished in several ways. The most obvious approach is to reweight the distribution using our measurement density function. This yields

$$w_{k+1}^{(j)} \propto w_k^{(j)}\pi_{Y_{k+1}|Z_{k+1}}(y_{k+1}|z_k) \tag{3}$$

and we normalize the weights to sum to one. While this approach has the benefit of simplicity, it can suffer from serious drawbacks numerically. As soon as a particle finds itself to be in an unlikely configuration for the system, it is assigned a very low weight. As such, it does not contribute to the approximation of the posterior distribution. All of the weight, and thus all of the responsibility for capturing the behavior of the distribution, is concentrated on a small number of particles. In high dimensions, this problem is particularly exaggerated. The curse of dimensionality in filtering methods is not merely a byproduct of increased model complexity as the dimension increases, but rather is related to the surrogate likelihood functions used to decide the weights. In the case of the commonly used Gaussian surrogate, for instance, the subset of N-dimensional space which is within some given number of standard deviations from the mean is spherical, the volume of which becomes smaller relative to unit volume as the dimension increases. Even if the posterior distribution is strongly localized, it may be the case that none of the particles land near its maximum and so the algorithm fails to converge to anything. The number of points expected to avoid this failure mode decreases exponentially as the dimension increases. Working with a distribution whose decay rate is slower than the Gaussian, such as a χ^2-squared, would mitigate these issues but not eliminate them.

2.1.2 Resampling Methods

In order to attempt to combat the problem discussed at the end of the previous section, we can make use of an approach called resampling. The central idea behind resampling is that instead of simply allowing particles that end up in a low-probability configuration to continue wandering the state space, we re-initialize them at one of the more probable locations found in the forecasting step. While this still necessitates a degeneracy of the approximation to the posterior by putting all the weight on the most likely particle locations, when the behavior of our simplified model is stochastic, each copy of the particle beginning at a given configuration can explore the state space in a different way, which will hopefully lead to a more balanced representation at future steps. The algorithm for resampling can be found below [1]:

Algorithm 2 Update scheme with resampling for particle filter

```
1: Input the initial model state-vector ensemble $\boldsymbol{Z} \in \mathcal{R}^{n \times N}$.
2: Input the measurement vector $\boldsymbol{d} \in \mathcal{R}^m$
3: Input the probability density of the data given a particle location $f(\boldsymbol{d}|\boldsymbol{z}_j)$
4: Find the logarithmic particle weights
       for $j = 1, N$ do
           $\tilde{w}_j = \ln f(\boldsymbol{d}|\boldsymbol{z}_j)$
       end for
5: Find the maximum of the logarithmic weights $\tilde{w}_{max} = max_j(\tilde{w}_j)$
6: Find unnormalized particle weights
       for $j = 1, N$ do
           $\tilde{w}_j = \exp(\tilde{w}_j - \tilde{w}_{max})$
       end for
7: Find the normalized particle weights
       for $j = 1, N$ do
           $w_j = \frac{\tilde{w}_j}{\Sigma_i \tilde{w}_i}$
       end for
8: Call the resampling function Resample(w,I) to get the desired indices $I$.
9: Resample ensemble
       for $j = 1, N$ do
           $\boldsymbol{z}_j = \boldsymbol{z}_{I_j}$
       end for
```

A key step in the resampling algorithm is to apply the resampling function. One common choice that resets all particles to a small number of likely locations is the Stochastic Universal Resampling Algorithm, detailed below [1]:

Algorithm 3 Stochastic Universal Resampling Algorithm

```
1: Input the initial weight $\boldsymbol{w} \in \mathcal{R}^N$.
2: $\hat{w}_1 = w_1$
3: for $j = 2, N$ do
       $\hat{w}_j = \Sigma_{i=1}^{j} w_j$
   end for
4: Generate a random number $u \sim [0, 1/N]$
5: Initialize k=1
6: Assign index of the sampled particles
   for $j = 1, N$ do
       while $u > \hat{w}_k$ do
           $k = k + 1$
       end while
       $I_j = k$
       $u = u + 1/N$
       k=1
   end for
```

The potential for degeneracy is clear in the proof of the convergence of this method, referred to in [2] as the bootstrap particle filter. The theorem states that for a given number of particles N, the error between the particle approximation and the true posterior is $O(N^{-\frac{1}{2}})$ for all timesteps $j \in [1, J]$ for some J. However, the constants associated with this bound are not uniform, and an examination of the proof in fact reveals that the constant $O(\lambda^J)$ for some $\lambda > 1$.

2.2 Modeling

In order to demonstrate the efficacy of particle filtering methods, we have applied them to a number of modeling tasks from a variety of applications. Each model is described below. The task for the particle filter is to use the behavior of a simplified version of each system combined with noisy observations of the full model to make predictions. We refer to the full model as the omniscient model, and the simplified model as the naive model.

2.2.1 SEI^rI^uR Metapopulation Model

In an epidemiological metapopulation model, the transmission of an infectious disease is represented on a network. Each node in the network represents a subpopulation in a particular location and the edges represent the movement of individuals between them. Each node comprises its own compartmental model with states for susceptible (S), exposed(E), reported infected (I^r), unreported infected (I^u), and recovered (R) populations. The separation of subpopulations on a network addresses the limitation of the "well-mixed population" assumption in standard compartmental models.

Omniscient Model. The metapopulation model used in this study was inspired by Pei et al. [3] where dynamics are split into daytime (equations 4) and nighttime transmission periods (equations 5).

$$\begin{aligned}
S_{ij}(t+dt_1) &= S_{ij}(t) - \frac{\beta_i(t)S_{ij}(t)\sum_k I^r_{ki}(t)}{N^d_i}dt_1 \\
&\quad - \mu\frac{\beta_i(t)S_{ij}(t)\sum_k I^u_{ik}(t)}{N^d_i}dt_1 \\
E_{ij}(t+dt_1) &= E_{ij}(t) + \frac{\beta_i(t)S_{ij}(t)\sum_k I^r_{ki}(t)}{N^d_i}dt_1 + \\
&\quad \mu\frac{\beta_i(t)S_{ij}(t)\sum_k I^u_{ik}(t)}{N^d_i}dt_1 - \frac{E_{ij}(t)}{Z}dt_1 \\
I^r_{ij}(t+dt_1) &= I^r_{ij}(t) + \alpha\frac{E_{ij}(t)}{Z}dt_1 - \frac{I^r_{ij}(t)}{D}dt_1 \\
I^u_{ij}(t+dt_1) &= I^u_{ij}(t) + (1-\alpha)\frac{E_{ij}(t)}{Z}dt_1 - \frac{I^u_{ij}(t)}{D}dt_1 \\
R_{ij}(t+dt_1) &= R_{ij}(t) + \frac{I^r_{ij}(t)}{D}dt_1 + \frac{I^u_{ij}(t)}{D}dt_1
\end{aligned} \tag{4}$$

$$\begin{aligned}
S_{ij}(t+1) &= S_{ij}(t+dt_1) - \frac{\beta_j(t+dt_1)S_{ij}(t+dt_1)\sum_k I^r_{kj}(t+dt_1)}{N^n_j}dt_2 \\
&\quad - \mu\frac{\beta_j(t+dt_1)S_{ij}(t+dt_1)\sum_k I^u_{kj}(t+dt_1)}{N^n_j}dt_2 \\
E_{ij}(t+1) &= E_{ij}(t+dt_1) + \frac{\beta_j(t+dt_1)S_{ij}(t+dt_1)\sum_k I^r_{kj}(t+dt_1)}{N^n_j}dt_2 \\
&\quad + \mu\frac{\beta_j(t+dt_1)S_{ij}(t+dt_1)\sum_k I^u_{kj}(t+dt_1)}{N^n_j} - \frac{E_{ij}(t+dt_1)}{Z}dt_2 \\
I^r_{ij}(t+1) &= I^r_{ij}(t+dt_1) + \alpha\frac{E_{ij}(t+dt_1)}{Z}dt_2 - \frac{I^r_{ij}(t+dt_1)}{D}dt_2 \\
I^u_{ij}(t+1) &= I^u_{ij}(t+dt_1) + (1-\alpha)\frac{E_{ij}(t+dt_1)}{Z}dt_2 - \frac{I^u_{ij}(t+dt_1)}{D}dt_2 \\
R_{ij}(t+1) &= R_{ij}(t+dt_1) + \frac{I^r_{ij}(t+dt_1)}{D}dt_2 + \frac{I^u_{ij}(t+dt_1)}{D}dt_2
\end{aligned} \tag{5}$$

where $N^d_i = N_{ii} + \sum\limits_{k\neq i} I^r_{ki} + \sum\limits_{k\neq i}(N_{ik} - I^r_{ik})$ and $N^n_i = \sum_k N_{ki}$ for $i = 1, \ldots, 17$. Here, $S_{ij}, E_{ij}, I^r_{ij}, I^u_{ij}$ and N_{ij} are the susceptible, exposed, reported infected, unreported infected and total populations in the subpopulation commuting from location j to location $i (i \leftarrow j)$; $\beta_i(t)$ is the time-varying transmission rate of reported infections in location i ; μ is the relative transmissibility of unreported infections; Z is the average latency period (from infection to contagiousness); D is the average duration of contagiousness; α is the fraction of documented infections; θ is a multiplicative factor adjusting random movement; dt_1 and dt_2 are the durations of daytime and nighttime transmission ($dt_1 + dt_2 = 1$); and N^d_i and N^n_i are the daytime and nighttime populations of county i. We assume the I^r_{ij} population is immobile and does not participate in human movement. We integrate Eqs. 4 and 5 using a Poisson process to represent the stochasticity of the transmission process.

Naïve model. The naïve model we use for data assimilation is done by collapsing our network into a single SEI$^{\text{u}}$I$^{\text{r}}$R model with no distinction between daytime and nighttime transmission.

$$\begin{aligned}
S(t+1) &= S(t) - \frac{\beta(t)S(t)I^r(t)}{N} - \mu\frac{\beta(t)S(t)I^u(t)}{N} \\
E(t+1) &= E(t) + \frac{\beta(t)S(t)I(t)}{N} + \mu\frac{\beta(t)S(t)I^u(t)}{N} - \frac{E(t)}{Z} \\
I^r(t+1) &= I^r(t) + \alpha\frac{E(t)}{Z} - \frac{I^r(t)}{D} \\
I^u(t+1) &= I^u(t) + (1-\alpha)\frac{E(t)}{Z} - \frac{I^r(t)}{D} \\
R(t+1) &= R(t) + \frac{I^r(t)}{D} + \frac{I^r(t)}{D}
\end{aligned} \tag{6}$$

The observations used by the particle filter were the daily case counts $i(t)$, defined to be the daily change in $I^r(t)$, i.e.,

$$i(t) = I^r(t+1) - I^r(t). \tag{7}$$

2.2.2 Coupled oscillators

Coupled oscillators consist of interacting systems that communicate through a network structure given by a connectivity matrix A. They tend to exhibit very interesting and non-trivial phenomena such as multistability. This richness makes them an excellent test case for particle filtering.

As before, we will use a naive model for filtering and an omniscient model for data generation. The former is called Brusselator and represents different chemical concentrations, while the latter are genetic oscillators that represent different cell concentrations. The fact that they are coupled with a network structure means that we can increase the dimension arbitrarily.

The difference between the naive model and the omniscient model relies on some parameters related to how fast the system gets to the final state. Making them homogeneous or heterogeneous can change the system very significantly as the system exhibits symmetry-breaking phenomena [4]. The naive model is simulated in the homogeneous regime, while the data is generated using the heterogeneous regime.

The equations for the Brusselators [5] are

$$\dot{x}_i = 1 - x_i(1 + b - c_i x_i y_i), \quad \dot{y}_i = x_i(b - c_i x_i y_i) - d(Ly)_i, \tag{8}$$

where $i \in \{1, ..., N\}$, N is the number of oscillators, d is the magnitude of the coupling, c is the magnitude of the nonlinear part, and L is the Laplacian of the network that couples the oscillators. The network can be directed.

We always have a fixed point $(x^*, y_i^*) = (1, \frac{b}{c_i})$ for all i, but with coupling there will be multistability in general, and the final state is strongly dependent on the initial conditions. All variable values are available to the naive model (as opposed to making a projective measurement).

For the coupled genetic oscillators [6], we have the following $3N + 1$ dimensional system:

$$\dot{u}_i = \frac{\alpha_1}{1+v_i^{\beta}} - u_i + \frac{\alpha_3 \omega_i^{\eta}}{1+\omega_i^{\eta}}, \qquad \dot{v}_i = \frac{\alpha_2}{1+u_i^{\gamma}} - v_i, \tag{9}$$
$$\dot{\omega}_i = \epsilon_i(\frac{\alpha_4}{1+u_i^{\gamma}} - \omega_i) + 2d(\omega_e - \omega_i), \dot{\omega}_e = \frac{d_e}{N}\sum_{i=1}^{N}(\omega_i - \omega_e).$$

Again, we always have a fixed point $\omega_i = \omega_e,\ u_i = u,\ v_i = v$ for all values of i which is given by the following implicit equations:

$$\frac{\alpha_4}{1+u^{\gamma}} = \omega_e,\ \frac{\alpha_2}{1+u^{\gamma}} = v,\ \frac{\alpha_1}{1+v^{\beta}} - u + \frac{\alpha_3 \omega_e^{\eta}}{1+\omega_e^{\eta}} = 0. \tag{10}$$

However, we can also have other behaviors, depending on what we choose for the parameters and for the initial conditions. We can use heterogeneous or homogeneous ϵ, and altering that will usually have consequences on the final states.

The parameters here are

$$\alpha_1, \alpha_2, \alpha_3, \alpha_4, \beta, \eta, \gamma, d_e, d, \tag{11}$$

where d_e and d are the magnitudes of the coupling, and the others model the interactions.

The omniscient model uses heterogeneous c_i, and then the naive model uses homogeneous ones.

2.2.3 Modified Lorenz 96 Model

As a prototype for chaotic behavior, the Lorenz 63 [7] system is well-known in the dynamical systems community. It takes the form

$$\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= x(r-z) - y \\ \dot{z} &= xy - bz \end{aligned} \tag{12}$$

where σ, r, and b are parameters of the system. This coupled system of ODEs is ultimately derived from more sophisticated meteorological models. The '63' in the name refers to the year of its discovery. Over three decades later, the Lorenz 96 [8] system was proposed. The dimension can be any natural number $N \geq 4$, and the evolution equations take the form

$$\dot{x_i} = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F \tag{13}$$

where F is a forcing parameter, and by convention we take $x_{-1} = x_{N-1}$ and $x_0 = x_N$. As in the other models detailed in this report, we perform particle filtering by introducing an omniscient model to generate data and a simplified model for use in evolving the particle positions. To that end, we consider the modified system

$$\dot{x_i} = (x_{i+1} - \epsilon x_{i-2})x_{i-1} - x_i + F \tag{14}$$

for some $\epsilon > 0$. For the omniscient model, we take $\epsilon = F = 0.1$, while the simplified model will take $\epsilon = 0$. In addition, after each update step when computing a trajectory of the omniscient model, we add a mean-zero Gaussian noise term with variance proportional to the magnitude of the solution to the deterministic model. The low-level model is given access to the full output of the omniscient model, and we have set the number of dimensions to be $N = 100$.

3 Simulation Results

In this section, we present the results of the SEI^{u}I^{r}R metapopulation model described in the previous section.

3.1 $SEI^{r}I^{u}R$ Metapopulation Model

3.1.1 Synthetic Data

We generated synthetic data using the parameters outlined in table 1. For the time-varying

Parameter	Description	Value
μ	Relative transmission rate of unreported cases	0.5
Z	Average latency period (days)	4
D	Average infectious period (days)	4
α	Reporting proportion	0.3
n_t	Number of days	100
n_{loc}	Number of locations	17
$E(0)$	Initial number of exposed	0
$I^{u}(0)$	Initial number of unreported infections	10

Table 1: SEI^{u}I^{r}R parameters

transmission rates $\beta_i(t)$ we prescribe a sigmoid functional form

$$\beta_i(t) := b_i^{initial} + \frac{b_i^{end} - b_i^{initial}}{1 + \exp\{-0.1(t-30)\}}$$

where

$$b^{initial} = [0.2, 2.5, 0.35, 3, 2, 1.7, 0.6, 3., 0.3, 2, 2.5, 3., 3, 5.5, 2, 3.2, 5.1]$$
$$b^{end} = [0.5, 0.5, 0.75, 9, 7, 3.7, 2, 3., 2., 5, 6.5, 2., 3, 4.5, 2, 3.8, 5.1].$$

Our population matrix N is outlined in figure 1. In the figure, the rows represent the **origin** and columns represent the **destination**, and we see that people seem to stay in their origin from the distribution of colors in the diagonal.

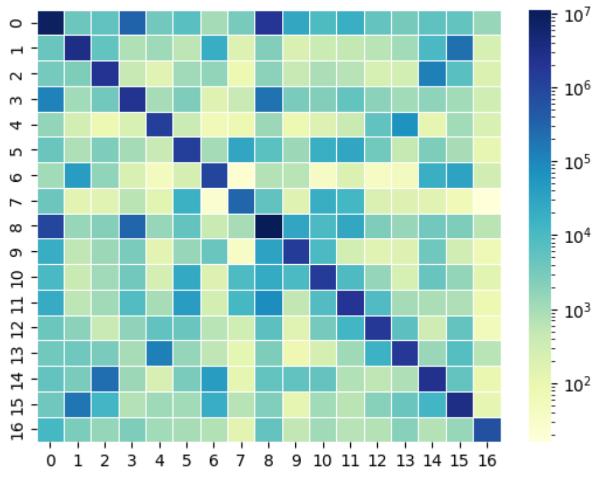

Figure 1: Population Matrix

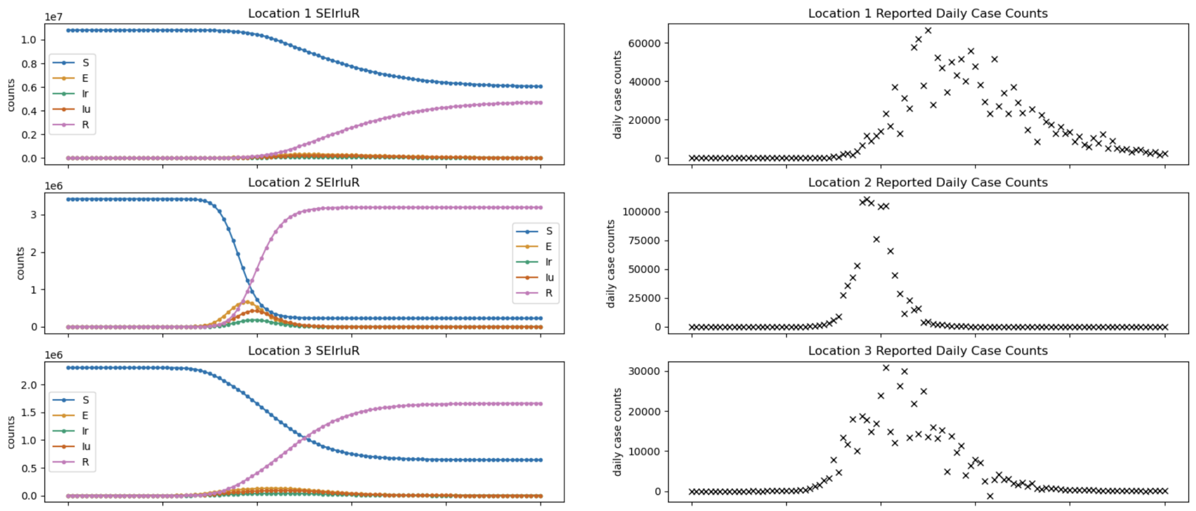

Figure 2: Synthetic data and observations for locations 1 - 3

We aggregated daily case counts across locations to be used by the naïve model in equation 6. Similarly, the transmission rate used in the naïve model is taken to be the average of the transmission rates across locations

$$\beta(t) = \frac{1}{n_{loc}} \sum_{i=1}^{n_{loc}} \beta_i(t).$$

The synthetic data and observations for locations 1-3 are presented in figure 2. The plots for the rest of the locations are presented in the appendix to save space. Now, using the bootstrap particle filter in algorithm 2 and naïve model in equation 6, we compute the posterior distributions for the states and observations. Figure 3 shows the obtained result. The blue points represent the output from the particle filter with the weight of the particle proportional to the size of the point. The yellow plot represents the true values and the black crosses ($\times$) represent the noisy observations. The output of the particle filter is close to the true value but does not follow the exact path. One of the reasons for this might be the fact that the naïve model does not represent the true system but is just an approximation of it and suffers from model misspecification. However, we do notice that the particle representation of the posterior distribution does shift its weight towards the observations around day 40. The posterior distribution plots for the remaining states can be found in the appendix.

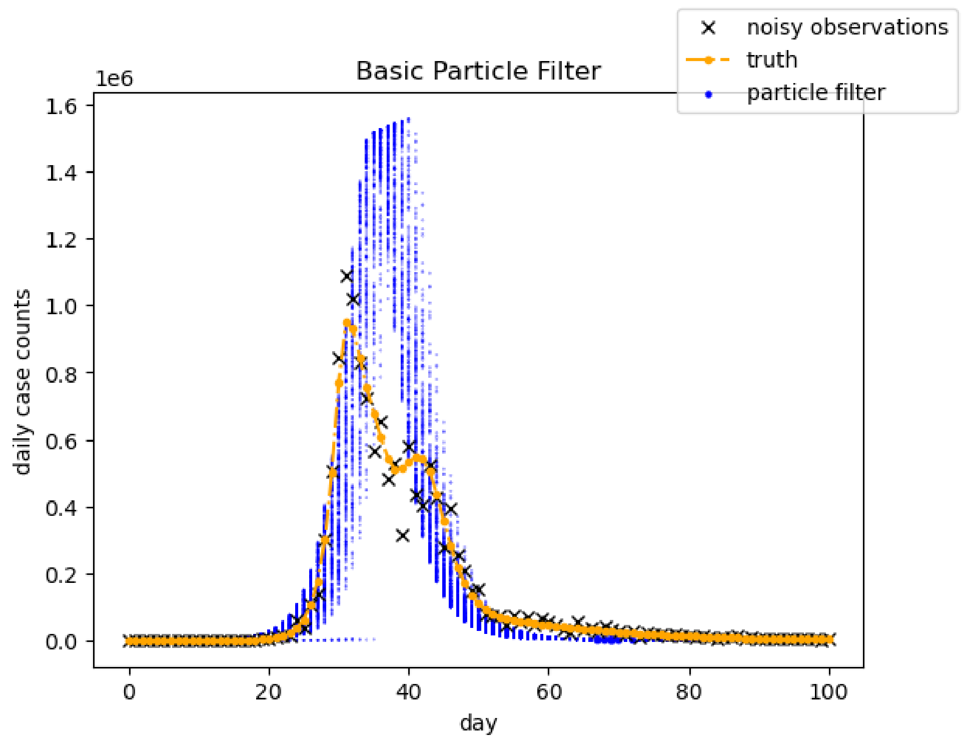

Figure 3: Posterior from Particle Filter

4 Discussion

4.1 Future Work: particle flow

The previous sections explained the strengths and weaknesses of traditional particle filters. One of the main drawbacks is the difficulty of analyzing data in higher dimensions, due to particle degeneracy.

This drawback can be addressed using a technique called particle flow, which seeks to discover an appropriate transformation from the prior PDF $f(\vec{z})$ to the posterior PDF $f(\vec{z} \mid \vec{d})$ in a smooth way. This can be done in a deterministic or stochastic manner. We will focus on the latter here as it is more interesting theoretically and in practice seems to lead to better results.

The smooth flow can be described by an SDE

$$d\vec{z} = \vec{m}_s(\vec{z})ds + d\vec{q}, \tag{15}$$

where $\vec{z}$ is the vector that contains the positions of our particles after the forecasting step. In the case of particle flow, we assume that all particles are weighted equally and focus solely on adjusting their position to reflect the posterior distribution.

We can describe the density of this flow using the Fokker-Planck equation

$$\frac{\partial p_\lambda(\vec{z})}{\partial \lambda} = -\nabla_z \cdot (p_\lambda(\vec{z})\vec{m}_\lambda) + \frac{1}{2}\nabla_z \cdot (C_f \cdot \nabla_z p_\lambda(\vec{z})), \tag{16}$$

with $f_0(\vec{z}) = f(\vec{z})$, $f_1(\vec{z}) = f(\vec{z} \mid \vec{d})$ and C_f is the covariance matrix that defines the stochastic part of the flow.

The goal is to find C_f and $\vec{m}_s$ such that the boundary conditions are satisfied, so we can characterize the flow correctly.

4.1.1 Likelihood Factorization

The most popular way to solve this problem is to use the following ansatz

$$p(x,\lambda) = c(\lambda) f(d \mid z)^{\lambda} f(z), \tag{17}$$

where $c(\lambda)$ is the appropriate normalization such that $p(x,1) = f(z \mid d)$.

Taking the logarithm of this ansatz and making use of the Fokker-Planck equation to simplify, we find that

$$\frac{1}{p_\lambda(z)} \frac{\partial p_\lambda(z)}{\partial \lambda} = \frac{1}{c} \frac{\partial c(\lambda)}{\partial \lambda} + \log(f(d \mid z)). \tag{18}$$

We want to avoid dealing with the normalization function, as it involves a complicated integral. One way to do that is by taking the gradient of both sides with respect to $\vec{z}$, so

$$\frac{1}{c}\frac{\partial c}{\partial \lambda} + \log(f(d \mid z)) = -\frac{1}{p_\lambda(\vec{z})} \nabla_z \cdot (p_\lambda(\vec{z}) \vec{m}_\lambda) + \frac{1}{2 p_\lambda(\vec{z})} \nabla_z \cdot (C_f \cdot \nabla_z p_\lambda(z)) \tag{19}$$
$$\nabla_z \log(f(d \mid z)) = -m_s^T \Delta \log(p_\lambda(z)) - \nabla_z(\nabla_z \cdot \vec{m}_s) - \nabla_z \log(p_\lambda(\vec{z})) \nabla_z \cdot \vec{m}_s + \frac{1}{2} \nabla_z \left(\frac{\nabla_z \cdot (C_f \nabla_z p_\lambda(\vec{z}))}{p_\lambda(\vec{z})} \right),$$

We note that we have infinitely-many solutions for this PDE, as we have N equations, but $\frac{N(N-1)}{2}$ due to C_f.

One useful class of solutions is presented in [9]

$$\vec{m}_s = -(\nabla_z^2 \log(f_s))^{-1} (\nabla_z \log(f(d \mid z)))^T \equiv \vec{m}_s^*, \quad C_f = -(\nabla_z^2 \log(f_s))^{-1} \nabla_z^2 \log(f(d \mid z)) \equiv C_f^*. \tag{20}$$

It is important to notice that the invariance of the normalization to the solution is crucial: algorithms that do not depend are the ones that work better.

As evidenced in the paper, the resulting flow induced by the SDE can yield significant improvements compared to other techniques. However, two key issues present themselves.

The first is that we are restricting the possible solutions by only solving for the gradient of the unknowns. In particular, there is a sort of gauge invariance of the type

$$\vec{m}_s = \vec{m}_s^* + \vec{\phi}(\lambda), \qquad C_f = C_f^* + A(\lambda), \tag{21}$$

where $\vec{\phi}(\lambda)$ and $A(\lambda)$ are respectively a vector and matrix that only depend on λ that lead to the same $c(\lambda)$, but different $\vec{m}_s$ and different C_f.

The equation that ϕ and A must respect is

$$2 \nabla_z p_\lambda(\vec{z}) \cdot \phi(\lambda) = Tr(A(\lambda) \cdot \nabla_z^2 p_\lambda(\vec{z})). \tag{22}$$

We plan to explore these relations and the different solutions that they lead to.

The second problem is the possible stiffness of the problem. By stiffness, we mean that standard numerical methods can lead to erroneous calculations, so we need very small time steps to integrate the flow, leading to very high computation times for what is essentially meant to be a conditioning step.

In the very recent paper [10], these issues were addressed in some detail. The authors analyze the condition number of the Jacobian of the SDE, as it is usually a good proxy for the stiffness of the system, via the condition number. This is defined to be

$$\kappa(J) = \begin{cases} || J |||| J^{-1} || & \text{if } J \text{ is not singular} \\ \infty \text{ else} \end{cases} . \tag{23}$$

The Jacobian depends on the solutions m_s and C_f, which motivates us to vary them appropriately. In this case, it takes the form

$$J(\lambda) = \nabla_\alpha m_s \mid_{\alpha=0} + \frac{1}{2} C_f C_F^T \nabla_z \nabla_z^T \log(p). \tag{24}$$

While one wants to minimize the condition number, one must also be careful with other error sources that can arise with very low condition numbers.

In the paper, the authors develop an algorithm that realizes this trade-off efficiently and obtains solutions that are more robust to stiffness.

Even with this improvement, stiffness still occurs in some cases. Our goal in the future is to think about what can be done to actually avoid stiffness in a broad and robust way.

5 Conclusion

In this report, we have demonstrated both the benefits and deficiencies of particle filtering use resampling. In particular, we examined the efficacy of using a basic SEI^uR^r to compute predictions for the true state of a more general $SEI^rI^uR^r$ with network structure imposed, given noisy realizations of the latter. While the predictions made in this case dis have some predictive power, they failed to resolve the more fine-grained multimodal structure of the ground truth. In the future, we hope to remedy these setbacks by applying more powerful methods such as particle flow. In particular, we hope to gain a strong enough understanding of this method to make headway on the problem proposed by Raytheon Technologies at next's weeks MPI. To that end, we also discussed a number of further tests for particle filtering methods that we can make use of to benchmark the performance of any proposed solutions to the PDEs discussed in Section 4. We would like to thank the organizers of the GSMMC, SIAM, and Professor Peter Kramer for supporting us as we explored these questions.

6 Appendix

6.1 Synthetic Data Plots

In this section, we present the comparison of the synthetic data and the observations for locations 4-17 as shown in figure 4-8.

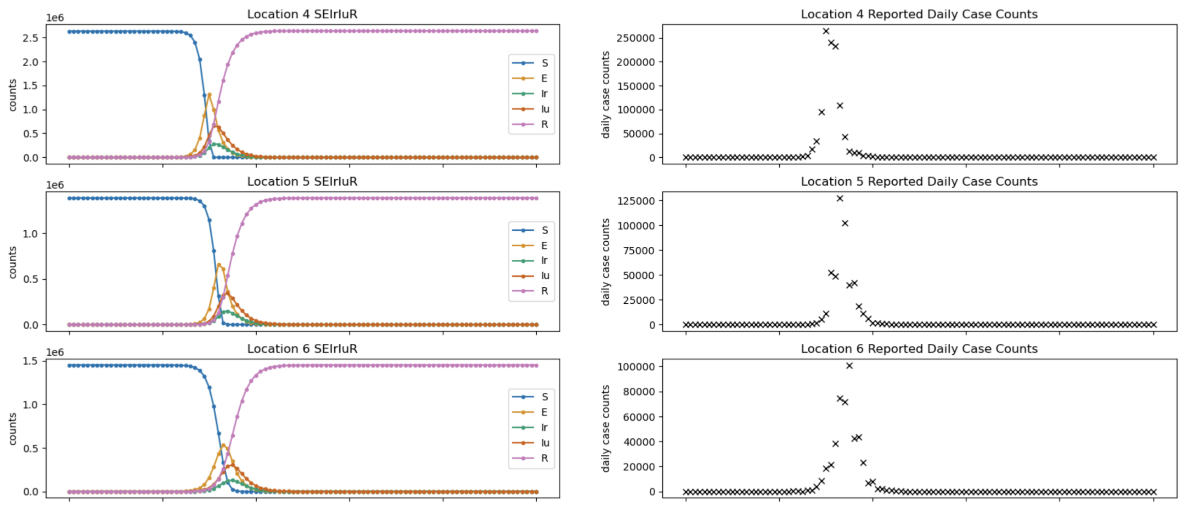

Figure 4: Synthetic data and observations for locations 4 - 6.

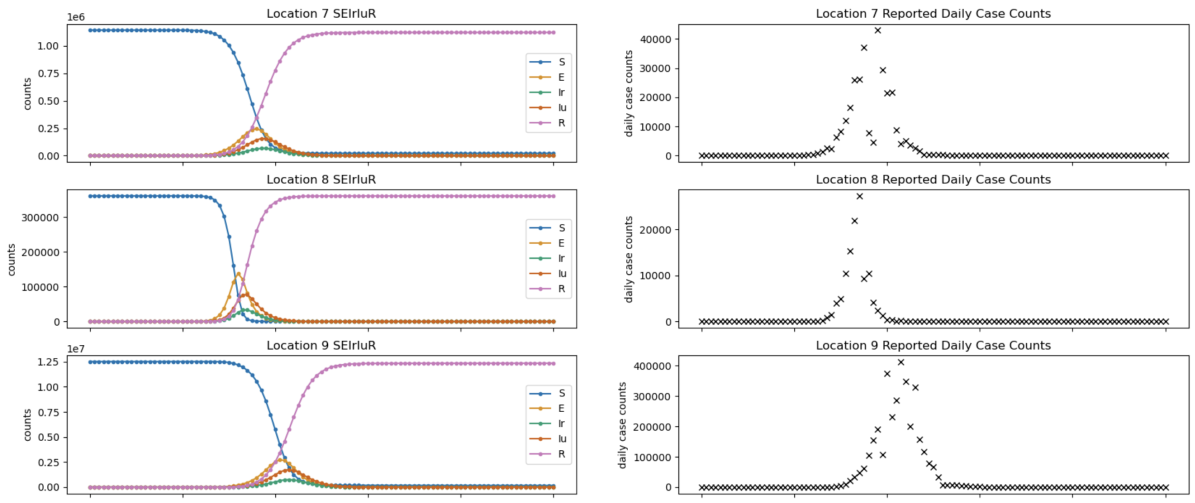

Figure 5: Synthetic data and observations for locations 7 - 9.

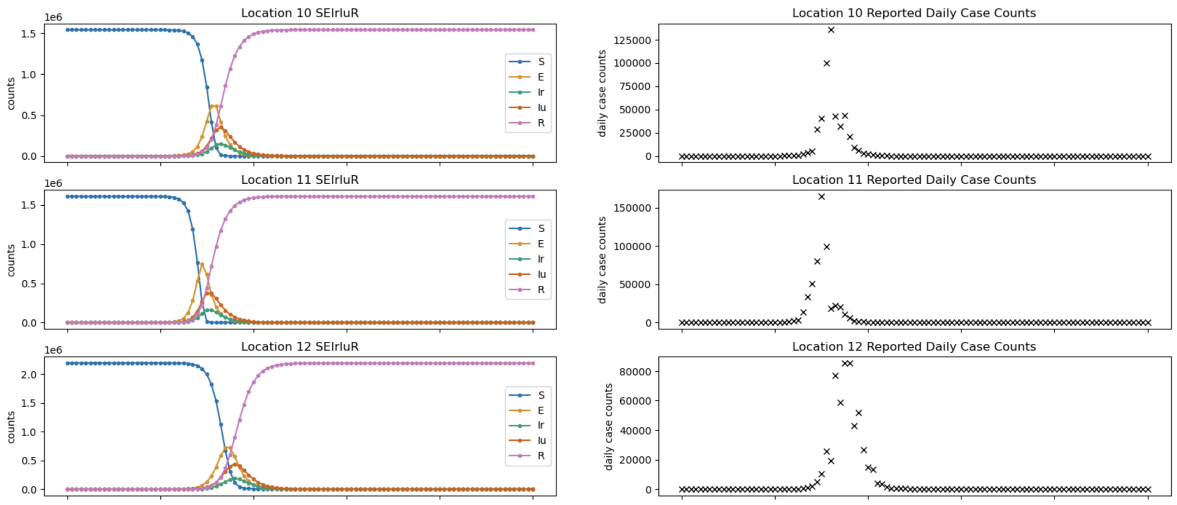

Figure 6: Synthetic data and observations for locations 10 - 12.

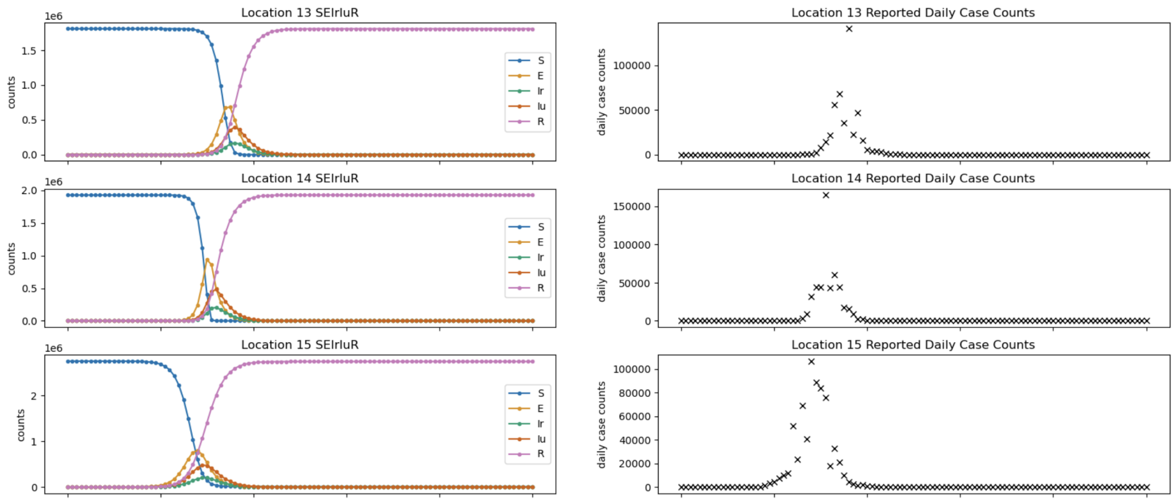

Figure 7: Synthetic data and observations for locations 13 - 15.

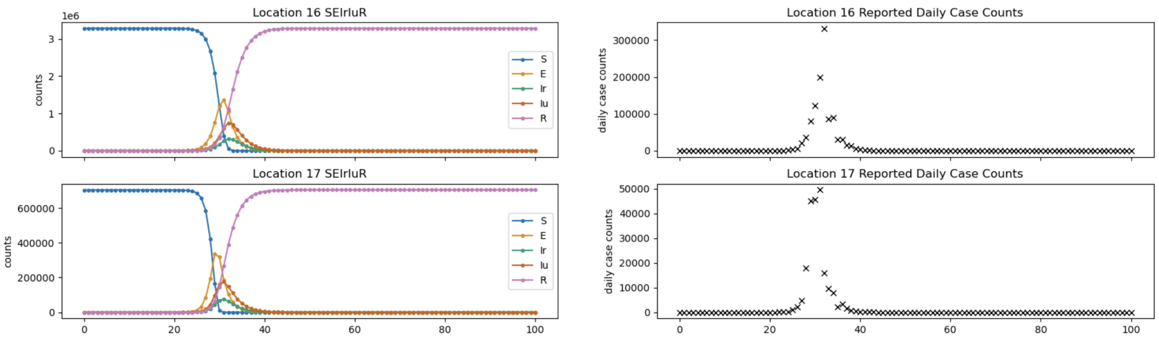

Figure 8: Synthetic data and observations for locations 16 - 17.

6.2 Particle Filter Posterior Distribution Plots

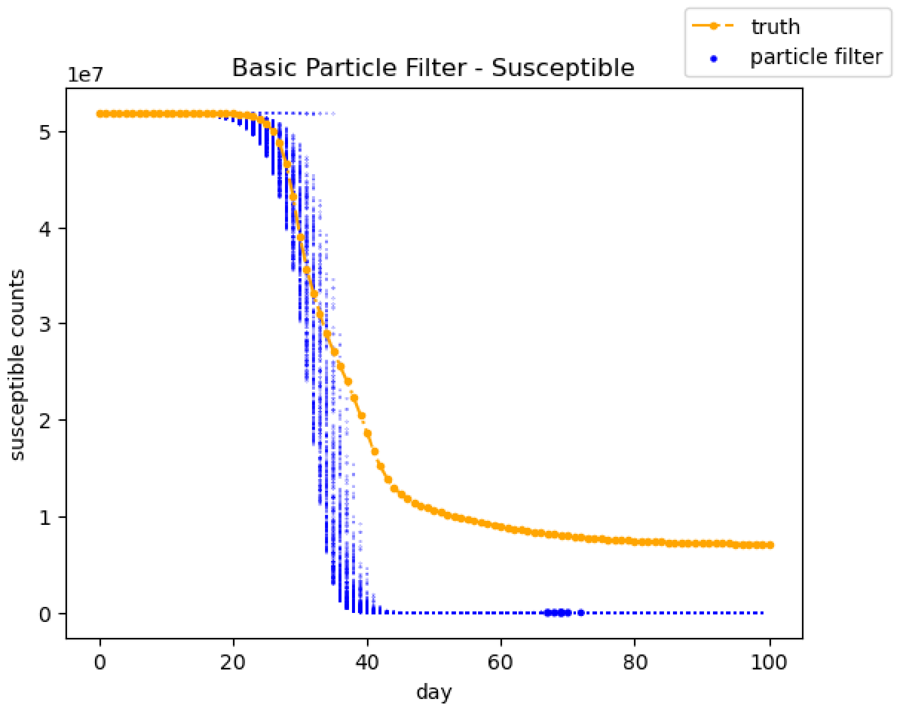

Figure 9: Posterior for S state from Particle Filter

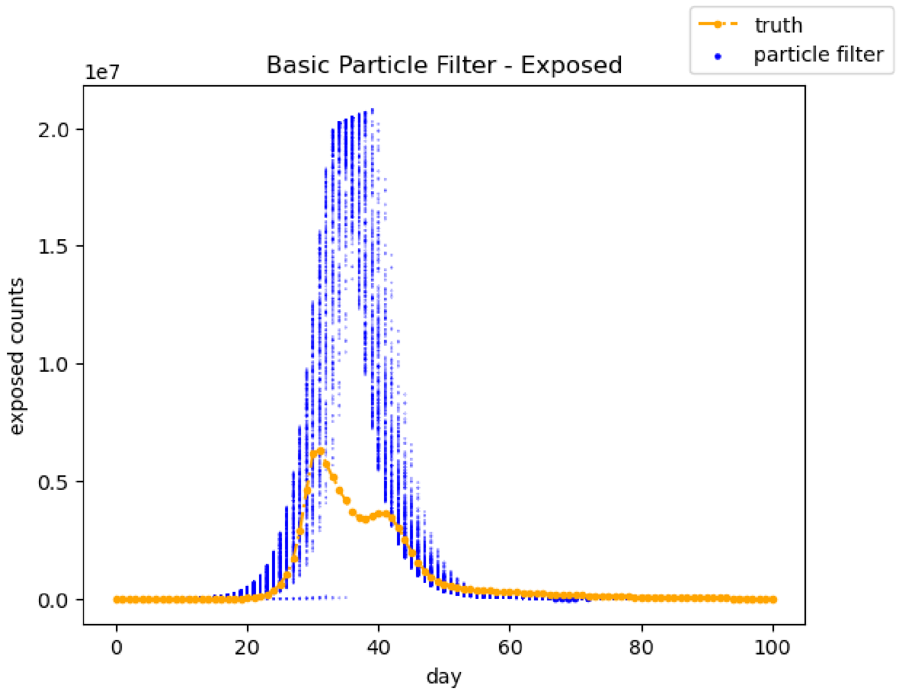

Figure 10: Posterior for E state from Particle Filter

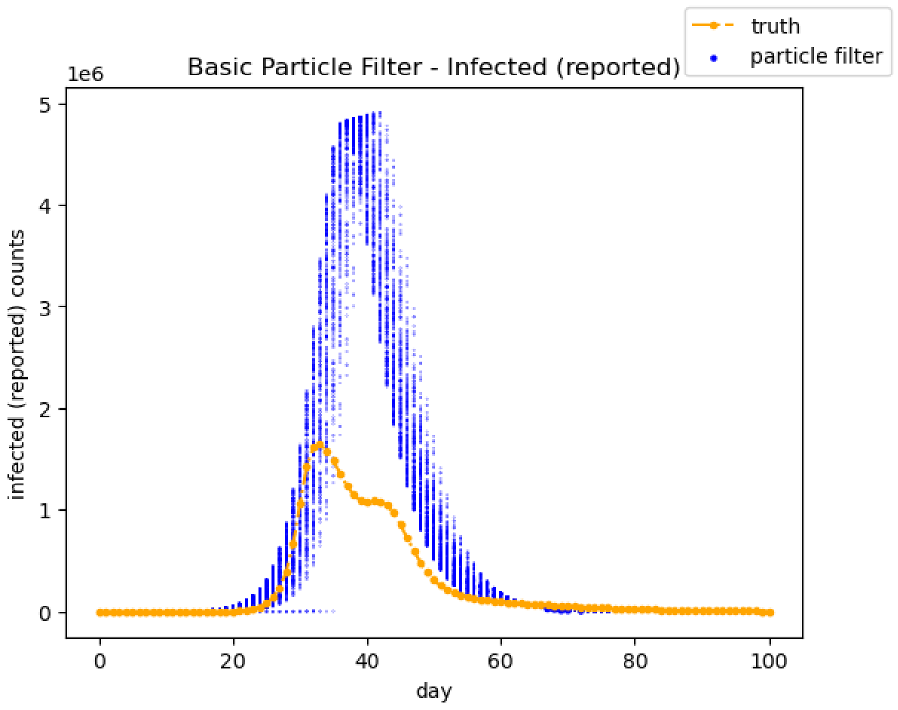

Figure 11: Posterior for I^r state from Particle Filter

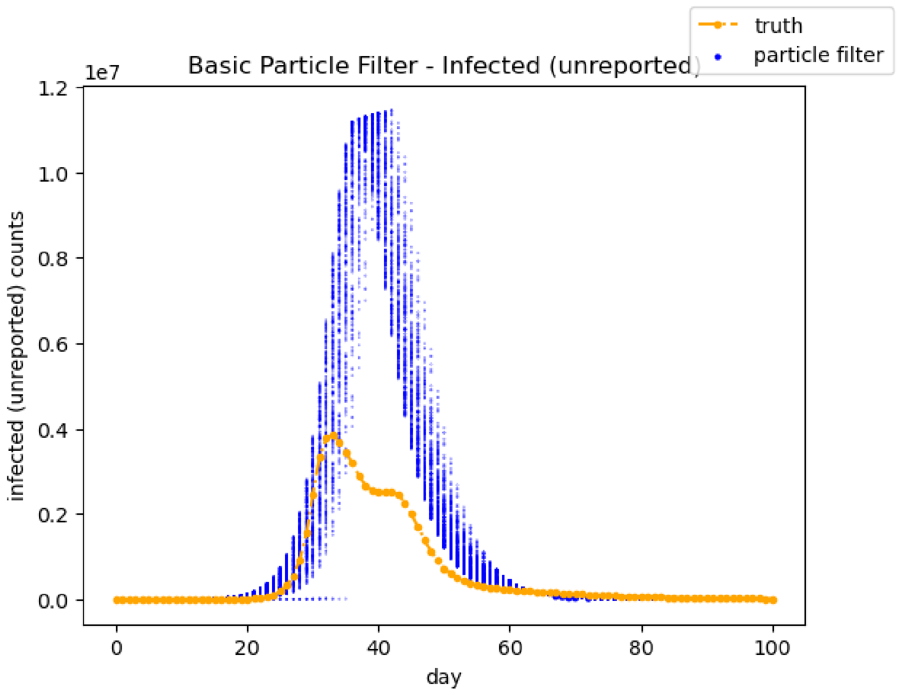

Figure 12: Posterior for I^u state from Particle Filter

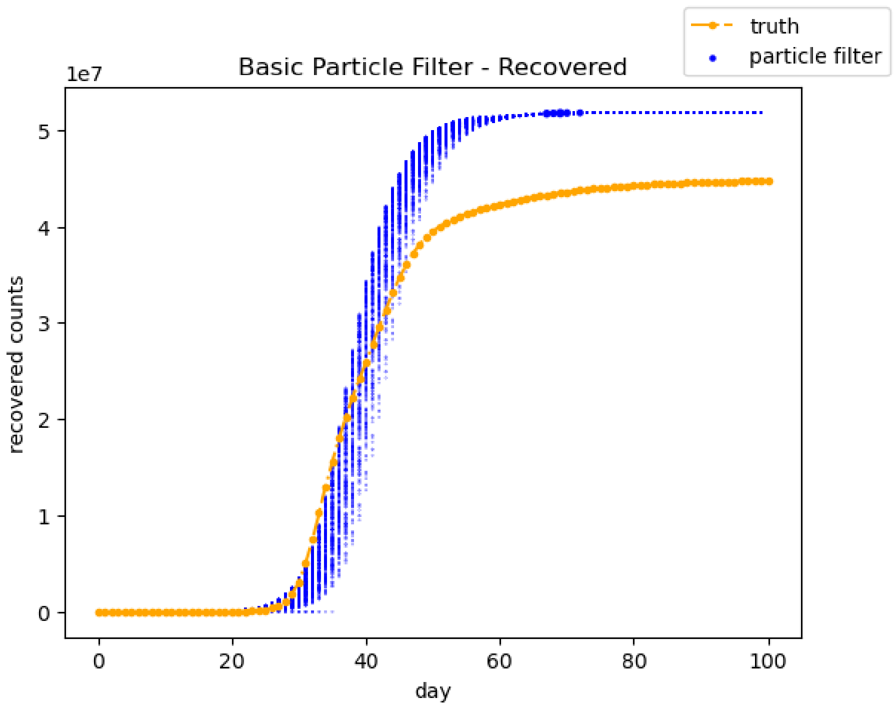

Figure 13: Posterior for R state from Particle Filter

References

[1] G. Evensen, F. C. Vossepoel, and P. J. van Leeuwen, *Data assimilation fundamentals: A unified formulation of the state and parameter estimation problem.* Springer Nature, 2022.

[2] D. Sanz-Alonso, A. Stuart, and A. Taeb, “Inverse problems and data assimilation,” February 2023.

[3] S. Pei, S. Kandula, and J. Shaman, “Differential effects of intervention timing on covid-19 spread in the united states,” *Science advances*, vol. 6, no. 49, p. eabd6370, 2020.

[4] A. Koseska, E. Volkov, and J. Kurths, “Parameter mismatches and oscillation death in coupled oscillators,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 20, no. 2, 06 2010, 023132. [Online]. Available: https://doi.org/10.1063/1.3456937

[5] I. Prigogine and R. Lefever, “Symmetry breaking instabilities in dissipative systems. ii,” *The Journal of Chemical Physics*, vol. 48, no. 4, pp. 1695–1700, 1968.

[6] A. Kuznetsov, M. Kæ rn, and N. Kopell, “Synchrony in a population of hysteresis-based genetic oscillators,” *SIAM Journal on Applied Mathematics*, vol. 65, no. 2, pp. 392–425, 2004.

[7] E. N. Lorenz, “Deterministic Nonperiodic Flow.” *Journal of Atmospheric Sciences*, vol. 20, no. 2, pp. 130–148, Mar. 1963.

[8] E. Lorenz, “Predictability: a problem partly solved,” Ph.D. dissertation, Shinfield Park, Reading, 1995 1995.

[9] F. Daum, J. Huang, and A. Noushin, “New theory and numerical results for gromov’s method for stochastic particle flow filters,” in *2018 21st International Conference on Information Fusion (FUSION)*, 2018, pp. 108–115.

[10] L. Dai and F. Daum, “On the design of stochastic particle flow filters,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 59, no. 3, pp. 2439–2450, 2023.