Bayesian models and inferential methods for forecasting disease outbreak severity

by

Nicholas Lorenz Michaud

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Statistics

Program of Study Committee:

Jarad Niemi, Major Professor

Petrutza Caragea

Heike Hofmann

Vivekananda Roy

Chong Wang

Iowa State University

Ames, Iowa

2016

DEDICATION

This dissertation is dedicated to my grandfather, Edward N. Lorenz, who instilled in me a lifelong love of exploration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the support and guidance of Dr. Jarad Niemi, my major professor. His knowledge, enthusiasm, and ability to inspire and encourage me through the occasional sharp turns and sometimes murky paths that my research has taken have enormously contributed to the completion of this work.

I would also like to thank Dr. Perry de Valpine, Dr. Daniel Turek, and Dr. Christopher Paciorek from the University of California, Berkeley. In addition to the continued expert mentoring and feedback they have provided on matters both statistical and related to software development, they have also made me feel very much a member of the NIMBLE development team (even though I've never met any of them in person)!

I am extremely grateful to my committee members, Dr. Petrutza Caragea, Dr. Heike Hofmann, Dr. Vivekananda Roy and Dr. Chong Wang, and to Dr. Somak Dutta for participating in my final defense. This dissertation is much stronger and more complete for their ideas. .

Finally, but of no less importance, has been the unwavering support of my family and friends. Specifically, to my girlfriend Madi, my mom, my dad, and my sister, who have on innumerable occasions provided me with excellent advice and steady reassurance, know that this dissertation would never have been completed, or started, without you.

CHAPTER 1. OVERVIEW

In this introduction, we describe the content of the chapters of the thesis, also explaining how they relate to each other. The different thesis chapters are binded together by their examination into Bayesian methods of modeling and conducting inference on time series data for disease outbreaks. Of primary interest is forecasting the course of disease outbreaks. Such forecasts can be invaluable to health care providers and governmental organizations as a method for determining what, and how many, resources to prepare for future public health crises. As such, our foremost focus for modeling and inferential efforts is not in estimating parameters related to disease outbreaks, but rather in exploring techniques which can increase forecast accuracy.

Chapter 2 of the thesis gives an overview of state space models, and describes a variety of techniques through which forecasting and parameter inference can be done for such models. State space models are a class of models for time series data where each observation, say y_t at time t, is assumed to be related to an unobserved latent state x_t. The latent states evolve over time according to some evolution equation $f_\theta(x_{t+1}|x_t)$, where θ is a vector of all top-level, non-time-specific parameters. State space models provide a general conceptual framework through which different models of disease outbreaks can be examined. With respect to outbreak modeling, y_t will usually be some observed measure of outbreak severity at time t, while x_t will be the true, underlying level of outbreak severity in a population.

Of particular note in conducting inference on state space models are Sequential Monte Carlo (SMC) algorithms. SMC algorithms, also known as particle filters, proceed by representing the distribution of the latent state $f_\theta(x_{t+1}|x_t)$ as a discrete set of samples called particles. At time t, each particle is re-weighted according to how likely it is to have generated the observed data at time t. Particles which are likely to have generated the observation y_t receive high weights,

while unlikely particles are down-weighted. From this set of weighted particles, a resampling procedure is conducted, whereby highly weighted particles are more likely to be sampled. The resulting particles constitute an equally weighted sample from the filtering distribution $f_\theta(x_t|y_t)$, which itself is of great interest in general state space modeling and in disease outbreak modeling.

After providing an overview of SMC techniques for state space models in Chapter 2, we then describe how to build and conduct inference on such models in the NIMBLE R package (NIMBLE Development Team, 2015). NIMBLE is a hierarchical modeling framework, in some ways similar to JAGS (Plummer et al.,) and STAN (Stan Development Team, 2015), which allows users to specify hierarchical models, and then apply a range of inferential techniques to those models. The SMC algorithms described in Chapter 2 were written by the author of this thesis, and are currently available for use in NIMBLE version 0.5-1.

Chapter 3 turns towards modeling disease outbreaks, and specifically towards modeling seasonal influenza. Over the last decade, the use of data from social media and search engines in outbreak forecasting has become commonplace. Such "now-casting" data has great potential as an aide to modeling, as they are reported in near-real time and thus allow for on-demand forecasts, as compared to official metrics produced by the CDC which can take weeks to be released. However, social media and search engine data are prone to biases, and failing to account for these biases can lead to biased forecasts. In Chapter 3, we propose two general sets of Bayesian models which can take multiple, possibly biased data streams and use them to produce forecasts of epidemic severity. One set consists of a Dynamic Linear Model (DLM) framework, and the second uses a Susceptible - Infectious - Recovered (SIR) compartmental model framework. Inference on the models is conducted using the SMC techniques within the NIMBLE package, as described in Chapter 2. Results from both simulation studies and using real world data show that incorporating social media data can greatly benefit forecasts, and that modeling the bias in such data can likewise increase predictive accuracy if such bias is actually present in the data.

Chapter 4 builds upon the efforts of Chapter 3 by introducing a hierarchical epidemic model which can use data from previous disease outbreaks to inform forecasts for the current season. Similar to the models of Chapter 3, the hierarchical model can include multiple data

sources, some of which may be taken as biased. This model is no longer a stochastic state space model, but instead uses a deterministic function of four season-specific parameters to produce latent seasonal disease outbreak curves. Inference and forecasting are conducted using a Markov chain Monte Carlo (MCMC) algorithm. The hierarchical model is compared to a non-hierarchical model and it is found that the inclusion of data from previous seasons has a significant, beneficial effect on forecasting. Additionally, a major benefit of using a non-stochastic model for latent disease severity is seen in terms of inference efficiency, where forecasting results from the hierarchical model can be obtained in a fraction of the time that it takes the models of Chapter 3 to produce inference.

Chapter 5 describes two smaller projects related to disease outbreak forecasting and surveillance that have been completed. One is an open source, automatically updating application for monitoring CDC published disease surveillance data, and the other is a missing data model to increase the accuracy of ILINet indicators.

Finally, Chapter 6 overviews the results from the previous chapters, and proposes promising directions for future Bayesian disease outbreak forecasting research.

CHAPTER 2. Sequential Monte Carlo Algorithms in the NIMBLE Package

2.1 Introduction

State space models provide a method for analyzing time series data, where observations are assumed to be noisy measurements of unobserved latent states that evolve over time. State space models have been used in such diverse fields as population ecology (Knape and de Valpine, 2012), epidemiology (Andersson and Britton, 2012), economics (Creal, 2012), and meteorology (Wikle et al., 2013). With the broad applicability of state space models has come a variety of techniques for conducting inference. A common goal of inference for a state space model is determining the filtering distribution of the model, that is, the distribution of the latent states given data up to a certain time point. The filtering distribution can be computed analytically for models following a linear Gaussian framework using the Kalman filter (Kalman, 1960). However, for models that do not fit the linear Gaussian framework, analytical solutions are usually unavailable. For such models, inference and estimation is commonly performed using a set of flexible computational algorithms known as Sequential Monte Carlo (SMC) methods (Doucet et al., 2001).

SMC methods are attractive as they provide a general framework for conducting inference on any state space model. In addition, SMC methods generally perform "on-line" inference, that is, inference on filtering distributions that can be updated sequentially as more data are received. A variety of SMC methods currently exist, including the bootstrap filter, auxiliary particle filter, Liu and West filter, Storvik filter, particle learning algorithm, and others. In addition, algorithms such as Particle Markov chain Monte Carlo (MCMC) have been developed that place SMC methods within a broader MCMC framework. Although the different SMC

algorithms estimate the filtering distribution using a variety of techniques, they are tied together through their use of sequential importance resampling (Doucet et al., 2001) to update filtering estimates as new data are received.

The generality with which SMC methods can be applied makes them perfectly suited for use within the **NIMBLE** R software package. **NIMBLE** (de Valpine et al., 2015) allows Bayesian hierarchical models to be written using the BUGS language. These models can then be analyzed using **NIMBLE**'s library of model-generic algorithms. **NIMBLE**'s SMC algorithms are described in detail in this paper. **NIMBLE** also has a variety of MCMC algorithms for more general Bayesian inference, as well as a Monte Carlo Expectation Maximization (MCEM) algorithm. Additionally, **NIMBLE** provides a domain specific language (DSL) that users can use to write their own model-generic algorithms. These algorithms can be run in R, or compiled into C++ for more efficient execution.

Other current software packages that implement SMC algorithms include the **POMP** R package (King et al., 2016), the **LibBi** package (Murray, 2015), the **Biips** package (Todeschini et al., 2014), and the **vSMTC** C++ template library (Zhou, 2015). **NIMBLE** differs from the aforementioned software in its focus on providing an accessible DSL for writing algorithms that can be applied to any model written in the BUGS language. As such, the **NIMBLE** software package offers an ideal platform for extending existing SMC algorithms or programming new ones.

In Section 2.2, we introduce state space models and the idea of filtering distributions. We then describe a variety of algorithms that can be used for inference on filtering distributions. In Section 2.3, we provide examples of specifying state space models within **NIMBLE**. Inference is conducted using **NIMBLE**'s SMC algorithms and the results are analyzed. In Section 2.4, we demonstrate **NIMBLE**'s programmability by coding an SMC algorithm within NIMBLE's DSL.

2.2 Sequential Monte Carlo Methods for State Space Models

2.2.1 State Space Models

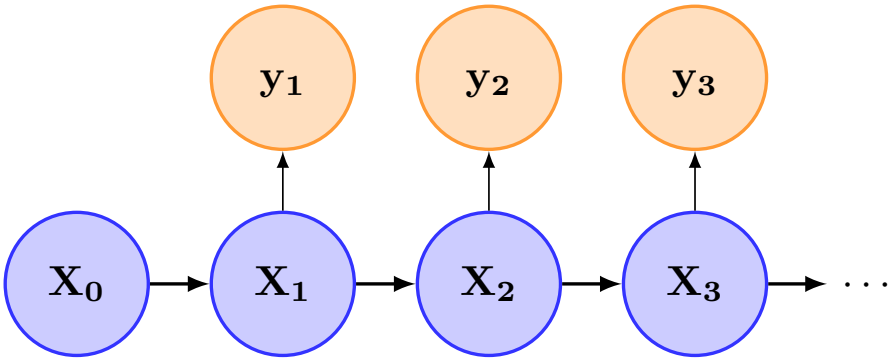

Figure 2.1: Diagram of a state space model.

State space models, also known as hidden Markov models, are used to model time series data or any data that arrives sequentially. The vector of data at each time t, labeled y_t, is assumed to be related to a latent, unobserved state x_t through an observation equation $y_t \sim p_t(y_t|x_t, \theta)$. Here, θ is a vector of top-level parameters that are assumed not to change with time. In addition to the observation equation describing the dependence of y_t on x_t, x_t depends on x_{t+1} through a transition equation $x_{t+1} \sim p_t(x_{t+1}|x_t, \theta)$. Both the observation and transition equation are stochastic. Frequently, the observation and transition equations remain constant over all time points, in which case the t subscript is dropped and they are written as $p(y_t|x_t, \theta)$ and $p(x_t|x_{t-1}, \theta)$. State space models have the following conditional independence property: $[x_{t+1}|x_{1:t}, \theta] = [x_{t+1}|x_t, \theta]$, where $x_{1:t} = (x_1, \ldots, x_t)$. Figure 2.1 shows the dependence structure assumed by state space models. Note that we assume no observation exists for $t = 0$, and that x_0 comes from a known distribution $p(x_0|\theta)$.

2.2.2 Filtering Algorithms

Often, the distribution $f(x_t|y_{1:t}, \theta)$, known as the filtering distribution for x_t, is of interest. Consider a situation where new data are received sequentially in time, and data are currently available up to time t – that is, $y_{1:t}$ is known. Upon receiving y_{t+1}, the filtering distribution $f(x_{t+1}|y_{1:t+1}, \theta)$ provides information about the latent state at the most recent time point, given

the entire set of data. Similarly, the distribution $f(x_{1:t+1}|y_{1:t+1}, \theta)$, known as the smoothing distribution, may be of interest. For certain types of state space models, these distributions are analytically tractable. For example, the Kalman filter (Kalman, 1960) can be used to derive the filtering distribution for state space models in which both the observation and transition equations are linear and have a Gaussian error term. However, there are a wide variety of state space models for which no analytical form of the filtering distribution is available.

For such intractable models, two estimation methods for the filtering distribution exist. MCMC algorithms can be used to draw samples from $f(x_{1:t}|y_{1:t}, \theta)$, if the fixed parameters are assumed to be known, or from $f(x_{1:t}, \theta|y_{1:t})$ if they are unknown. However, MCMC algorithms for state space models can encounter poor mixing for $x_{1:t}$, which is frequently of high-dimension and whose elements are highly dependent on each other. Certain MCMC algorithms have been designed that work towards alleviating the issue of slow mixing – for example, the forward-filtering backward-sampling (FFBS) algorithm of Frühwirth-Schnatter (1994) and Carter and Kohn (1994) obtain samples from $p(x_{1:t}|y_{1:t}, \theta)$ by first iterating through the model forwards in time, and then backwards. Carlin et al. (1992) provide an example of a Gibbs Sampler MCMC algorithm targeting the filtering distribution that uses a mixture of multivariate normal distributions to approximate the observation and transition equations.

A second group of methods for estimating the filtering distribution for non-linear state space models are known as sequential Monte Carlo methods, or particle filters. These methods rely on importance sampling to estimate filtering distributions sequentially in time, using the samples from the previous time point to generate samples for the next time point. In Section 2.2.3 and Section 2.2.4, two types of SMC methods (the Bootstrap filter and Auxiliary particle filter) are described, each of which can be used to generate samples from the filtering distribution $p(x_t|y_{1:t}, \theta)$ or the smoothing distribution $p(x_{1:t}|y_{1:t}, \theta)$. A third method, known as the Liu and West filter, can be used to concurrently sample values of any top-level parameters along with latent states, resulting in an approximation of $p(x_t, \theta|y_{1:t})$. The Liu and West filter is described in Section 2.2.5.

In addition to sampling from the filtering and smoothing distributions of the latent states, SMC algorithms can be used to approximate the marginal likelihood of a state space model,

$p(y_{1:t}|\theta)$. Just as filtering distributions for state space models are intractable for a wide range of model specifications, likelihoods for state space models are often unavailable analytically as well. Particle filters provide broadly applicable methods for estimating these likelihoods, which can in turn be used for model selection, or in an MCMC framework to obtain samples from $p(\theta|y_{1:t})$. In Section 2.2.6, a particle MCMC algorithm is detailed that uses particle filters to estimate likelihoods within a Metropolis-Hastings MCMC sampling scheme for θ.

The Ensemble Kalman filter (ENKF) can also be used to conduct inference on the filtering distribution of the latent states. Similar to SMC techniques, the ENKF approximates the filtering distribution via a collection of particles that are propagated forwards in time. However, whereas SMC methods use importance sampling to select particles at each time point, the ENKF instead shifts particles towards the filtering distribution using an approximation to the Kalman gain matrix. The ENKF is described in Section 2.2.7.

2.2.3 Bootstrap Filter

The bootstrap filter of Gordon et al. (1993) uses importance sampling to sequentially generate samples from $f(x_t|y_{1:t})$ at each time t. Note that since the Bootstrap filter assumes fixed values of top level parameters, we omit the dependence on θ in our notation. Specifically, suppose that we have P samples, called particles, from $f(x_{t-1}|y_{1:t-1})$ labeled $x_{t-1}^{(p)}$ for $p \in \{1, \ldots, P\}$. The Bootstrap filter first propagates each of these particles forward according to a proposal distribution $q(x_t|x_{t-1}^{(p)}, y_t)$. Importance weights, $\pi_t^{(p)}$, are then calculated for each particle, and the propogated particles are resampled according to these weights. This results in an equally weighted sample $(x_t^{(p)})_{p=1}^P$ from $f(x_t|y_{1:t})$.

Specifically, note that Step 10 in Algorithm 1 creates an equally weighted sample from the target distribution. Additionally, an estimate of the likelihood $p(y_{1:T})$ can be obtained by $\tilde{p}(y_{1:T}) = \prod_{t=1}^T \tilde{p}(y_{t|1:t-1})$, where $\tilde{p}(y_{t|1:t-1})$ is given in line 13 of the algorithm. We also note that in Step 10 in Algorithm 1, δ is defined as the Dirac delta function.

The resampling step of Algorithm 1 is performed to reduce particle degeneracy. Particle degeneracy is a feature of sequential importance sampling algorithms that lack a resampling step. In such algorithms, as particles are repeatedly propagated forward, a small number of particles

Algorithm 1 Bootstrap Filter

1: **for** p in $1:P$ **do**
2: Generate $x_0^{(p)} \sim p(x_0)$
3: Set $\pi_0^{(p)} = \frac{1}{P}$
4: **end for**
5: **for** t in $1:T$ **do**
6: **for** p in $1:P$ **do**
7: Generate $\tilde{x}_t^{(p)} \sim q(x_t|x_{t-1}^{(p)}, y_t)$
8: Calculate $w_t^{(p)} = \frac{p(\tilde{x}_t^{(p)}|x_{t-1}^{(p)})p(y_t|\tilde{x}_t^{(p)})}{q(\tilde{x}_t^{(p)}|x_{t-1}^{(p)}, y_t)}\pi_t^{(p)}$
9: Normalize $w_t^{(p)}$ as $\pi_t^{(p)} = \frac{w_t^{(p)}}{\sum_{i=1}^P w_t^{(i)}}$
10: Sample $x_t^{(p)} \sim \sum_{i=1}^P \pi_t^{(i)} \delta(x - \tilde{x}_t^{(i)})$
11: Set $\pi_t^{(p)} = \frac{1}{P}$
12: **end for**
13: Calculate $\tilde{p}(y_{t|1:t-1}) = \frac{1}{P}\sum_{p=1}^P w_t^{(p)}$
14: **end for**

will have most of the weight placed on them, while the majority of particles will have practically zero weight (Doucet et al., 2000). Thus the algorithms will spend computational effort in propagating and weighting particles that contribute little to our knowledge of the target distribution. Resampling ensures that mostly highly-weighted particles will be propogated forwards, increasing algorithm efficiency and providing a better estimate of the target distribution.

However, resampling particles at each time point can lead to a loss of particle "diversity" (Doucet et al., 2000), as many of the resampled particles at each time point will have the same value. Thus it has been proposed (Smith et al., 2001) that resampling should take place only if particle degeneracy becomes too significant. An estimate of particle degeneracy is the effective sample size, calculated at each time t as $ESS = \frac{1}{\sum_{p=1}^P \pi_t^{(p)}}$. To combat particle degeneracy, it is recommended in Smith et al. (2001) that a resampling step should be conducted (Step 10) in the bootstrap filter only if the effective sample size becomes too low, indicating many particles with low weights. As a criterion for when a resampling step should take place, a threshold τ must be chosen with $0 \leq \tau \leq 1$, such that the algorithm will resample particles whenever $\frac{ESS}{P} < \tau$. Note that choosing $\tau = 0$ will mean that the resampling step is never performed, and choosing $\tau = 1$ will ensure that sampling is performed at each time point. To perform the above algorithm without resampling, simply remove Steps 10 and 11. If the resampling step is not

performed, the set $(\tilde{x}_t^{(p)}, \pi_t^{(p)})$ will constitute a weighted sample from the target distribution. Various methods for resampling particles have been proposed, including systematic resampling, residual resampling, and multinomial resampling (Doucet and Johansen,).

Additionally, samples from the smoothing distribution $p(x_{1:t}|y_{1:t})$ can be obtained within Algorithm 1 by recording the lineage of each particle at time t. Following the notation of Andrieu et al. (2010), define $B_n^{(p)}$ as the index of the ancestor particle at time n that gives rise to particle p at time t. Each particle p will then provide a sample from the smoothing distribution as $(x_1^{B_1^{(p)}}, \ldots, x_{t-1}^{B_{t-1}^{(p)}}, x_t^{(p)})$.

2.2.4 Auxiliary Particle Filter

The auxiliary particle filter algorithm (APF) of Pitt and Shephard (1999a) uses importance sampling similarly to the Bootstrap filter, but includes an additional "look-ahead step". At each time point t, the Auxiliary particle filter algorithm calculates first-stage weights $w_{t|t-1}^{(p)}$ for particles from time $t-1$. These weights are calculated using an estimate of the likelihood of the current data given each particle from the previous time point, labeled $q(y_t|x_{t-1}^{(p)})$. Particles with high first-stage weights correspond to values of the latent state at time $t-1$ that are likely to generate the observed data at time t. If available, calculating these weights using the true density $q(y_t|x_{t-1}^{(p)}) = p(y_t|x_{t-1}^{(p)})$ is an optimal choice (Pitt and Shephard, 2001). However, this density is usually not available. Pitt and Shephard (1999a) recommend choosing an auxiliary variable $\tilde{x}_{t|t-1}^{(p)}$ and then setting $q(y_t|x_{t-1}^{(p)}) = p(y_t|\tilde{x}_{t|t-1}^{(p)})$. Possible methods for choosing $\tilde{x}_{t|t-1}^{(p)}$ include simulating a value from $p(x_t|x_{t-1}^{(p)})$, or taking $\tilde{x}_{t|t-1}^{(p)} = E(x_t|x_{t-1}^{(p)})$.

The first-stage weights are used to sample P particles from time $t-1$, labeled $\tilde{x}_{t-1}^{(p)}$ for $p = 1, \ldots, P$. The sampled particles are then propagated forwards by a proposal distribution $q(x_t^{(p)}|\tilde{x}_{t-1}^{(p)}, y_t)$ and reweighted using second-stage weights $w_t^{(p)}$, providing a weighted sample from $p(x_t|y_{1:t})$. The APF as shown in Pitt and Shephard (1999a) optionally includes a second resampling step after Step 12, using the second-stage weights. However, the algorithm using a single resampling step has been shown to be more efficient (Carpenter et al., 1999).

In a manner similar to the Bootstrap filter, the APF can be used to obtain an estimate of the likelihood $p(y_{1:T})$ as $\tilde{p}(y_{1:T}) = \prod_{t=1}^{T} \tilde{p}(y_{t|1:t-1})$, where $\tilde{p}(y_{t|1:t-1})$ is given in line 14 of the

Algorithm 2 Auxiliary Particle Filter

for p in $1:P$ **do**
  Generate $x_0^{(p)} \sim p(x_0)$
  Set $\pi_0^{(p)} = \frac{1}{P}$
end for
for t in $1:T$ **do**
  for p in $1:P$ **do**
    Compute $w_{t|t-1}^{(p)} = q(y_t|x_{t-1}^{(p)})\pi_{t-1}^{(p)}$
    Normalize $w_{t|t-1}^{(p)}$ as $\pi_{t|t-1}^{(p)} = \frac{w_{t|t-1}^{(p)}}{\sum_{i=1}^{P} w_{t|t-1}^{(i)}}$
    Sample $\tilde{x}_{t-1}^{(p)} \sim \sum_{i=1}^{P} \pi_{t|t-1}^{(i)} \delta(x - x_{t-1}^{(i)})$
    Sample $x_t^{(p)} \sim q(x_t|\tilde{x}_{t-1}^{(p)}, y_t)$
    Calculate $w_t^{(p)} = \frac{p(x_t^{(p)}|\tilde{x}_{t-1}^{(p)})p(y_t|x_t^{(p)})}{q(y_t|\tilde{x}_{t-1}^{(p)})q(x_t^{(p)}|\tilde{x}_{t-1}^{(p)}, y_t)}$
    Normalize $w_t^{(p)}$ as $\pi_t^{(p)} = \frac{w_t^{(p)}}{\sum_{i=1}^{P} w_t^{(i)}}$
  end for
  Calculate $\tilde{p}(y_{t|1:t-1}) = \left(\sum_{p=1}^{P} \frac{w_t^{(p)}}{P}\right)\left(\sum_{p=1}^{P} w_{t|t-1}^{(p)}\right)$
end for

APF algorithm.

2.2.5 Liu and West Filter

Unlike the Bootstrap and Auxiliary particle filters, the Liu and West filter (Liu and West, 2001) allows inference to be conducted on both the latent states and the fixed parameters. Although variations on the Liu and West filter with the potential for increased efficiency have been proposed (Polson et al., 2008), we present the original filter. At each time point t, the Liu and West filter provides samples from $f(x_t, \theta|y_{1:t})$, the joint posterior distribution of the latent states and fixed parameters. Suppose we have a sample of P particles from $f(x_{t-1}, \theta|y_{1:t-1})$, labeled $(x_{t-1}^{(p)}, \theta_{t-1}^{(p)})$ for $p = 1, \ldots, P$. Note that θ_{t-1} is not meant to imply that the θ parameters vary over time, as they are fixed parameters, but is rather a notation to denote our estimates of θ at time $t-1$.

The Liu and West filter proceeds by first calculating first-stage weights for both the latent states and fixed parameters, similar to the Auxiliary particle filter. For each particle, an auxiliary value for the latent state is calculated and labeled $\tilde{x}_{t|t-1}^{(p)}$. Liu and West choose

$x_{t|t-1}^{(p)} = E(x_t|x_{t-1}^{(p)}, \theta_{t-1}^{(p)})$. An auxiliary value for the fixed parameters is also calculated as $\theta_{t|t-1}^{(p)} = a\theta_{t-1}^{(p)} + (1-a)\bar{\theta}_{t-1}$, where $\bar{\theta}_{t-1}$ is the average of all θ particles at stage $t-1$, and where a is a known shrinkage coefficient. These auxiliary values $(x_{t|t-1}^{(p)}, \theta_{t|t-1}^{(p)})$ are then given first-stage weights $w_{t|t-1}^{(p)}$ and resampled using these weights.

After the first-stage resampling, the fixed parameters are propagated forwards by a normal kernel density with mean $\theta_{t|t-1}^{(p)}$ and variance h^2V_{t-1}, where h^2 is a scaling parameter and V_{t-1} is the covariance matrix of the parameter particles $\theta_{t-1}^{(p)}$ with weights $\pi_{t-1}^{(p)}$ from the previous time point. Finally, state particles are propagated forwards and given second stage weights $w_t^{(p)}$.

Algorithm 3 Liu and West Filter

for p in $1:P$ **do**
  Generate $x_0^{(p)} \sim p(x_0)$
  Generate $\theta_0^{(p)} \sim p(\theta)$
  Set $\pi_0^{(p)} = \frac{1}{P}$
end for
for t in $1:T$ **do**
  for p in $1:P$ **do**
    Compute $x_{t|t-1}^{(p)} = E(x_t|x_{t-1}^{(p)}, \theta_{t-1}^{(p)})$
    Compute $\theta_{t|t-1}^{(p)} = a\theta_{t-1}^{(p)} + (1-a)\bar{\theta}_{t-1}$
    Compute $w_{t|t-1}^{(p)} = \pi_{t-1}^{(p)} p(y_t|x_{t|t-1}^{(p)}, \theta_{t|t-1}^{(p)})$
    Normalize $w_{t|t-1}^{(p)}$ as $\pi_{t|t-1}^{(p)} = \frac{w_{t|t-1}^{(p)}}{\sum_{i=1}^{P} w_{t|t-1}^{(i)}}$
    Sample $(\tilde{x}_{t-1}^{(p)}, \tilde{\theta}_{t-1}^{(p)}) \sim \sum_{i=1}^{P} \pi_{t|t-1}^{(i)} \delta(x - x_{t-1}^{(i)}) \delta(\theta - \theta_{t-1}^{(i)})$ and set $\tilde{x}_{t|t-1}^{(p)} = E(x_t|\tilde{x}_{t-1}^{(p)}, \tilde{\theta}_{t-1}^{(p)})$
    Sample $\theta_t^{(p)} \sim N(\tilde{\theta}_{t-1}^{(p)}, h^2 V_{t-1})$
    Sample $x_t^{(p)} \sim p(x_t|\tilde{x}_{t-1}^{(p)}, \theta_t^{(p)})$
    Calculate $w_t^{(p)} = \frac{p(y_t|x_t^{(p)}, \theta_t^{(p)})}{p(y_t|\tilde{x}_{t|t-1}^{(p)}, \tilde{\theta}_{t|t-1}^{(p)})}$
    Normalize $w_t^{(p)}$ as $\pi_t^{(p)} = \frac{w_t^{(p)}}{\sum_{i=1}^{P} w_t^{(i)}}$
  end for
end for

2.2.6 Particle MCMC Methods

Particle MCMC methods (Andrieu et al., 2010) also allow joint sampling from the posterior distribution of the states and the fixed parameters, but in a manner quite different from the Liu and West filter. Particle MCMC takes advantage of the ability of certain particle filters to provide estimates of the marginal likelihood of the data, that is, $\tilde{p}(y_{1:T}|\theta) \approx \int_X p(y_{1:T}|x_{1:T}, \theta) dx_{1:T}$. For example, the Bootstrap filter and Auxiliary filter (Pitt, 2002) can both be used to provide unbiased estimates of the marginal likelihood, as detailed in Sections 2.2.4 and 2.2.3. Below, we detail the Particle Marginal Metropolis Hastings (PMMH) algorithm, one of three algorithms

provided in (Andrieu et al., 2010).

At each iteration i, the PMMH algorithm first proposes a value θ^* for the model parameters θ from a proposal distribution $q(\theta^*|\theta^{i-1})$. Using this proposed value for θ, a particle filter is then run, which provides an estimate of $\tilde{p}(y_{1:T}|\theta^*)$, the likelihood of our data given our proposed parameters . This marginal likelihood estimate is used to calculate a Metropolis-Hastings acceptance probability for the proposed parameters, labeled p^*, via the equation given in Step 6 of Algorithm 4. With probability p^*, we accept the proposed parameters and set $\theta^i = \theta^*$. Otherwise, we set $\theta^i = \theta^{i-1}$. Thus each iteration of the algorithm will provide us with a single sample from $p(\theta|y_{1:T})$. If we are also interested in getting samples from the filtering distribution of the latent states, an index k can be randomly sampled from $\{1, \ldots, P\}$ at each iteration i, and the particle chain $x_{1:T}^{(k)}$ can be drawn from the output of the particle filter at that iteration.

Algorithm 4 PMMH Algorithm

1: Choose an initial value θ^0
2: Run an SMC algorithm to get a sample $x_{1:T}^0 \sim p(x_{1:T}|y_{1:T}, \theta^0)$ and a marginal likelihood estimate $\hat{p}(y_{1:T}|\theta^0)$
3: **for** iteration $i \geq 1$ **do**
4: Sample $\theta^* \sim q(\theta|\theta^{i-1})$
5: Run an SMC algorithm to get a sample $x_{1:T}^* \sim p(x_{1:T}|y_{1:T}, \theta^*)$ and a marginal likelihood estimate $\hat{p}(y_{1:T}|\theta^*)$
6: Compute $p^* = 1 \wedge \frac{\hat{p}(y_{1:T}|\theta^*)p(\theta^*)}{\hat{p}(y_{1:T}|\theta^{t-1})p(\theta^{t-1})} \frac{q(\theta^{t-1}|\theta^*)}{q(\theta^*|\theta^{t-1})}$
7: Generate $r \sim unif(0, 1)$
8: **if** $p^* > r$ **then**
9: Set $\theta^i = \theta^*$ and $x_{1:T}^i = x_{1:T}^*$
10: **else**
11: Set $\theta^i = \theta^{i-1}$ and $x_{1:T}^i = x_{1:T}^{i-1}$
12: **end if**
13: **end for**

2.2.7 Ensemble Kalman Filter

In Section 2.2.2, the Kalman filter was mentioned as providing an analytic solution to the filtering problem when working with a linear, Gaussian state space model. When using a model with non-linear transition equations or observation equations, however, the Kalman filter is no longer applicable. One solution to the filtering problem for Gaussian state space models with non-linear transition or observation equations is the Ensemble Kalman filter, which uses a particle representation of the latent states at each time point. Although the EnKF's particle representation mirrors that of the Bootstrap and Auxiliary Particle filters described in Sections 2.2.3 and 2.2.4, the EnKF updates the latent state particles using a fundamentally different approach than the SMC methods described previously. Instead of using a sequential importance sampling framework to resample particles, the EnKF first propagates particles forward using the transition equation, and then adjusts their position using a Monte Carlo approximation to the Kalman gain matrix. An overview of the ENKF can be found in Gillijns et al. (2006). In addition, Evensen (2003) provides a comprehensive list of papers that either use or propose modifications to the EnKF.

The EnKF assumes the following forms for the observation and transition equations:

$$x_t = f(x_{t-1}) + w_t \tag{2.1}$$

$$y_t = g(x_t) + v_t \tag{2.2}$$

where w_t and v_t are normally distributed error terms with covariance matrices Q_t and R_t respectively. At each time t, assume that we have a sample of P particles from $p(x_{t-1}|y_{t-1})$ labeled $x_{t-1}^{(p)}$ for $p = 1, \ldots, P$. The particles are propagated forward according to equation 2.1, giving a sample $\tilde{x}_t^{(p)}$. From these particles, a length P vector of latent errors $E_t^x = (\tilde{x}_t^{(1)} - \bar{\tilde{x}}_t, \ldots, \tilde{x}_t^{(P)} - \bar{\tilde{x}}_t)$ is calculated, where $\bar{\tilde{x}}_t$ is the average latent state taken over all particles at time t. Additionally, a vector of observation errors is calculated as $E_t^y = (\tilde{y}_t^{(1)} - \bar{\tilde{y}}_t, \ldots, \tilde{y}_t^{(P)} - \bar{\tilde{y}}_t))$, where $\tilde{y}_t^{(p)} = g(\tilde{x}_t^{(p)})$. From these error vectors, an approximate Kalman gain matrix $\tilde{K}_t$ is calculated, which in turn is used to adjust the $\tilde{x}_t^{(p)}$ particles to provide a sample from $p(x_t|y_{1:t})$.

Algorithm 5 Ensemble Kalman Filter

for p in $1:P$ **do**
  Generate $x_0^{(p)} \sim p(x_0)$
end for
for t in $1:T$ **do**
  for p in $1:P$ **do**
    Generate $\tilde{x}_t^{(p)} \sim p(\tilde{x}_t|x_{t-1}^{(p)})$
    Calculate $\tilde{y}_t^{(p)} = g(\tilde{x}_t^{(p)})$
  end for
  Calculate $E_t^x = (\tilde{x}_t^{(1)} - \bar{\tilde{x}}_t, \ldots, \tilde{x}_t^{(P)} - \bar{\tilde{x}}_t)$
  Calculate $E_t^y = (\tilde{y}_t^{(1)} - \bar{\tilde{y}}_t, \ldots, \tilde{y}_t^{(P)} - \bar{\tilde{y}}_t)$
  Calculate $\tilde{P}_t^{xy} = \frac{1}{P-1} E_t^x (E_t^y)^T$
  Calculate $\tilde{P}_t^{yy} = \frac{1}{P-1} E_t^y (E_t^y)^T$
  Calculate $\tilde{K}_t = \tilde{P}_t^{xy} (\tilde{P}_t^{yy})^{-1}$
  for p in $1:P$ **do**
    Generate $v_t^{(p)} \sim N(0, R_t)$
    Calculate $x_t^{(p)} = \tilde{x}_t^{(p)} + \tilde{K}_t(y_t + v_t^{(p)} - g(\tilde{x}_t^{(p)}))$
  end for
end for

2.3 Using Sequential Monte Carlo Methods in the NIMBLE Package

This section describes how to specify a statistical model in the BUGS language and manipulate that model using the **NIMBLE** package within the R (R Core Team, 2015a) statistical programming language. After describing some of the tools that **NIMBLE** provides to interact with models in Section 2.3.1, we demonstrate the available SMC methods within NIMBLE. Section 2.3.2 describes **NIMBLE**'s SMC methods for inference in models with no unknown parameters. Section 2.3.3 describes available SMC methods for state space models with unknown parameters. A supplement to the paper includes a full R script of all code shown below.

2.3.1 Creating and Manipulating BUGS Models Within NIMBLE

The **NIMBLE** package uses the BUGS language to specify models. We will not describe model specification in the BUGS language here – interested readers can find a brief overview of writing BUGS models in the **NIMBLE** User Manual (NIMBLE Development Team, 2015), or a more detailed guide in Lunn et al. (2012). Instead, we focus on how to interact with BUGS models using **NIMBLE**. To introduce **NIMBLE**'s features, we will use a linear Gaussian state space model in which there are no unknown parameters. Such a model will allow us to validate the results of our sequential Monte Carlo algorithms with the analytic solutions provided by the Kalman filter. Let y_t be the observed data at time point t, let x_t be the latent state at time point t, and suppose we have 10 time points. The model is:

$$
\begin{aligned}
x_1 &\sim N(0, 1) \\
x_t &\sim N(0.8 * x_{t-1}, 1) \quad for \quad t = 2, \ldots, 10 \\
y_t &\sim N(x_t, 0.5) \quad for \quad t = 1, \ldots, 10
\end{aligned}
$$

where $N(\mu, \sigma^2)$ denotes the normal distribution with mean μ and variance σ^2. We remark that although this example model is relatively simple, the algorithms presented in Sections 2.3.2 and 2.3.3 can be applied to any state space model written in BUGS.

To use the **NIMBLE** package, we first must load it by calling

```
> library(nimble)
> set.seed(1)
```

Models written in the BUGS language are read by the `nimbleCode` function. For example, BUGS code for the linear Gaussian model can be written and read into the **NIMBLE** package by calling

```
> exampleCode <- nimbleCode({
+   x[1] ~ dnorm(0, var = 1)
+   y[1] ~ dnorm(x[1], var = .5)
+   for(t in 2:10){
+     x[t] ~ dnorm(.8x[t-1], var = 1)
+     y[t] ~ dnorm(x[t], var = .5)
```

```
+   }
+ })
```

Once the code has been read in, a `nimbleModel` object can be created, which will allow **NIMBLE** to manipulate the model. To create a `nimbleModel`, we use the `nimbleModel` function, using the `exampleCode` as an argument. We also need to provide `data` to the `nimbleModel` in the form of a named list. In this example, we will use a placeholder vector of 0's as data and fill in other values below.

```
> exampleModel <- nimbleModel(code = exampleCode,
+                             inits = list(x = rep(0, 10)),
+                             data = list(y = rep(0, 10)))
```

To fill in values for the data in our model, we first simulate values of the latent states using **NIMBLE**'s `simulate` function. The `simulate` function is used to obtain random draws for stochastic nodes in our model. Once values for x have been simulated, the `calculate` function is used to calculate values for any deterministic dependencies that our latent nodes have. Note that although there were no deterministic dependencies for any of our x nodes specified in the BUGS model, **NIMBLE** will sometimes create additional deterministic nodes, often to account for non-standard parameterizations of distributions. Finally, the simulated x states are used to simulate values for our data. Additional information about the `calculate` and `simulate` functions can be found in Section 2.4.1.

```
> simulate(exampleModel, 'x')
> calculate(exampleModel, exampleModel$getDependencies('x'))
> simulate(exampleModel, 'y', includeData = T)
> exampleModel$y
```

Next, we compile the model using the `compileNimble` function. Compiling a model generates C++ code, compiles that code, and loads it back into R with an object that can be used just like the uncompiled model. The values in the compiled model will be initialized from those of the original model in R, but the original and compiled models are distinct objects so any subsequent changes in one will not be reflected in the other. Compiling a `nimbleModel` is necessary to run compiled algorithms in C++, which run much faster than uncompiled algorithms in R.

```
>  compileNimble(exampleModel)
```

2.3.2 Inference For Models With Known Parameters

Now that we have our compiled `nimbleModel` object for the example model, we can use algorithms from NIMBLE's library to conduct inference. These algorithms are all written as functions within **NIMBLE**'s DSL. We begin by demonstrating the use of the Bootstrap filter to estimate the filtering distribution $f(x_t|y_{1:t})$. The algorithm for the Bootstrap filter is provided in Section 2.2.3.

```
> exampleBootstrapFilter <- buildBootstrapFilter(exampleModel, nodes = 'x',
+                  control = list(saveAll = T, thresh = .9))
```

The `buildBootstrapFilter` function builds a Bootstrap filter for the model given in the first argument. The `nodes` argument gives the name (or names) of the latent states to be filtered. Importantly, **NIMBLE** filters require the latent states to have the same dimension at each time point. The `control` argument allows the user to specify a list of options that can fine-tune the performance of **NIMBLE** filtering algorithms. For example, the `saveAll` control list argument is a logical argument that determines whether the algorithm should store a matrix with samples from the filtering distribution for all time points, or only for the final time point. Additional arguments to the control list can be found by calling `help(buildBootstrapFilter)`.

Once the Bootstrap filter has been built for the example model, it can be run in R by calling the `run` method of the filter, taking the number of particles to use as an argument. When run, the filter will return an estimate of the log likelihood of the data. For example, the code below runs the Bootstrap filter with 100 particles.

```
> exampleBootstrapFilter$run(100)
[1] -15.3636
```

For users wishing to write their own algorithms, constructing and running **NIMBLE** functions in R allows for easy testing and debugging of algorithm logic. Once an algorithm has been successfully constructed in R, it can be compiled into C++ for efficient execution. Similarly to compiling the model in Section 2.3.1, algorithms can be compiled using the `compileNimble` function. Below, we compile the Bootstrap filter algorithm and run it with 10,000 particles.

```
> cBootstrapFilter <- compileNimble(exampleBootstrapFilter, project = exampleModel)
> cBootstrapFilter$run(10000)
```

The Bootstrap filter, like most filters in **NIMBLE**, saves two arrays with samples from the filtering distribution. One array, named `mvEWSamples`, contains equally weighted samples from the filtering distribution. The second array, `mvWSamples`, contains non-equally weighted samples from the filtering distribution along with weights for each sample. These arrays are **NIMBLE** `modelValues` objects, as described in Section 2.4.1, but can be easily converted to R matrices via the `as.matrix` function. The code below accesses the `mvEWSamples` and plots 95% credible intervals for the filtering distribution at each time point, along with points for the observed data.

```
> EWSamples <- as.matrix(cBootstrapFilter$mvEWSamples)
> bootQuants <- apply(EWSamples, 2, quantile, probs = c(.025, .975))
> matplot(y = t(bootQuants), type = "l", col = 1,
+         lty = 2,  xlab = "Time", ylab = "")
> lines(exampleModel$x, type = "o")
> legend("topleft", lty = c(1, 2, 2), pch = c(1, NA, NA, NA), bty = "n",
+ legend = c("observed value", "filtered interval"))
```

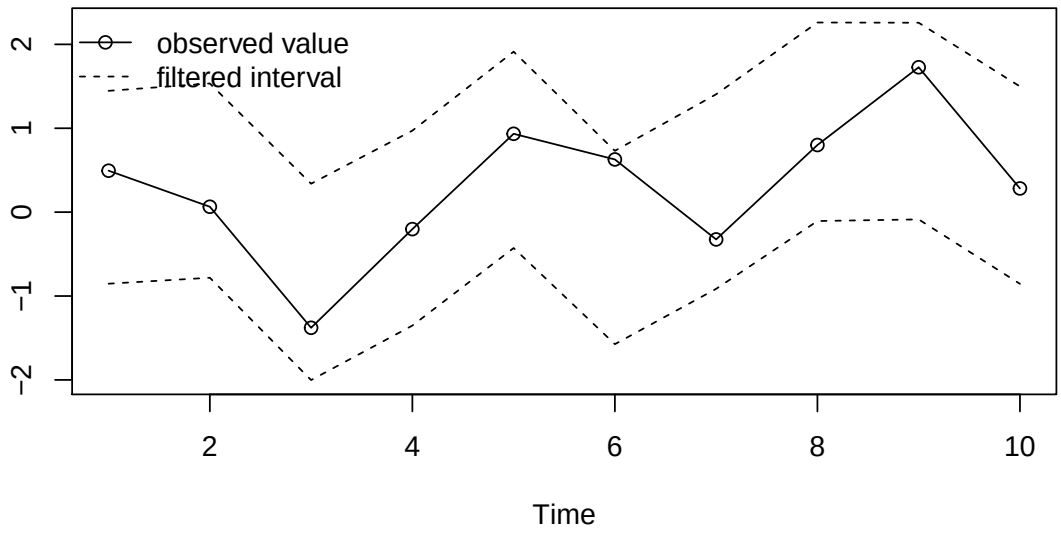

Figure 2.2: 2.5 % and 97.5 % quantiles (dotted lines) of the filtering distribution from the Bootstrap filter. True values of the latent states (open dots, connected by solid lines) fall within the filtering quantiles at each time point.

Next, we demonstrate **NIMBLE**'s Auxiliary particle filter algorithm. The Auxiliary particle filter is constructed in **NIMBLE** similarly to the Bootstrap filter, using a call to the `buildAuxiliaryFilter` function. As detailed in Section 2.2.4, the Auxiliary particle filter uses a lookahead function to select promising particles to propagate forwards at each time point. **NIMBLE**'s auxiliary filter allows users to choose between two lookahead functions: one that uses a simulation from the transition equation $\tilde{x}^{(p)}_{t|t-1} \sim p(x_t|x^{(p)}_{t-1})$, and one that uses the expected value of the transition equation $\tilde{x}^{(p)}_{t|t-1} = E(x_t|x^{(p)}_{t-1})$, via the `lookahead` control list argument.

```
> exampleAuxiliaryFilter <- buildAuxiliaryFilter(exampleModel, nodes = 'x',
+  control = list(saveAll = T,  smoothing = F, lookahead = 'mean'))
> cAuxiliaryFilter <- compileNimble(exampleAuxiliaryFilter,
+                                   project = exampleModel, resetFunctions = T)
> cAuxiliaryFilter$run(100000)
```

The final method we demonstrate for models with no unknown parameters is the Ensemble Kalman filter, which can be built similarly to the Bootstrap and Auxiliary Particle filters via a call to `buildEnsembleKF`. Note that the Ensemble Kalman filter, as described in Section 2.2.7, does not produce weights with its particle estimates. Thus only one output array, named `mvSamples`, is available for this algorithm.

```
> exampleEnsembleKF <- buildEnsembleKF(exampleModel, nodes = 'x',
+ control = list(saveAll = T))
> cEnsembleKF <- compileNimble(exampleEnsembleKF,
+                              project = exampleModel, resetFunctions = T)
> cEnsembleKF$run(10000)
> filterSamples <- as.matrix(cEnsembleKF$mvSamples)
```

Since our example model has normal transition and observation equations, the filtering distribution can also be calculated analytically using the Kalman filter (Kalman, 1960). Below, we use the **dlm** package (Petris, 2010) to apply a Kalman filter to our model, and compare the analytic filtering distribution provided by the Kalman filter to the approximate filtering distribution given by the Auxiliary particle filter. Note that the quantiles in Figure 2.3 align almost exactly for the two filters.

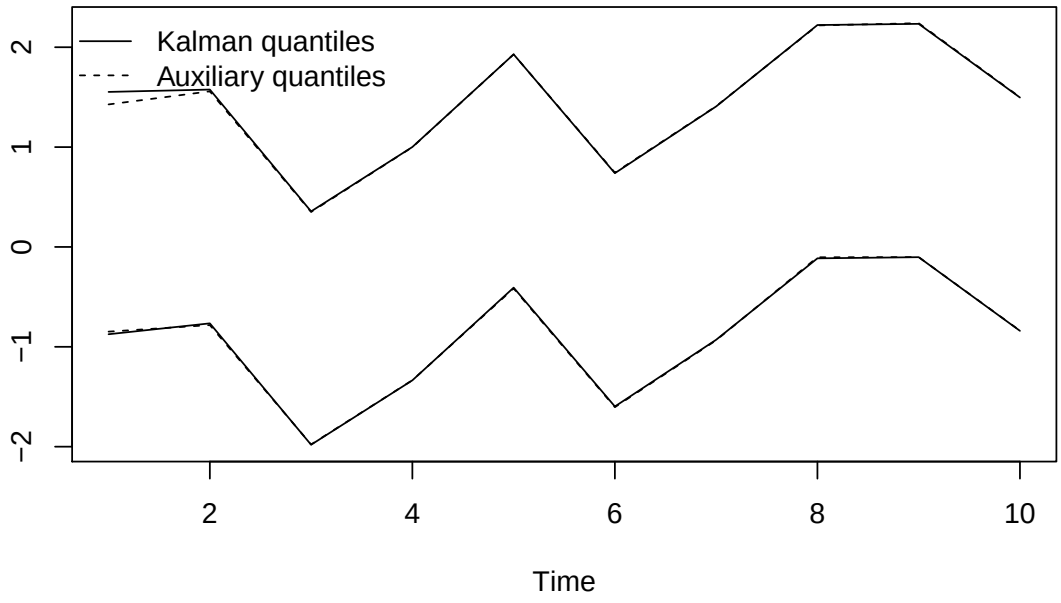

Figure 2.3: 2.5 % and 97.5 % quantiles of the filtering distribution for both the ensemble Kalman filter (solid line) and the auxiliary particle filter (dotted line).

2.3.3 Inference for Models With Unknown Parameters

The example model in the previous section had no unknown parameters – an unlikely scenario for dealing with real data. We next demonstrate **Nimble**'s Liu and West filter and PMCMC algorithms, both of which can be used to estimate the posterior distributions of unknown top-level parameters in state space models. To demonstrate these algorithms, we first construct a stochastic volatility model, which we use to model latent volatility in daily exchange rates. Then, we demonstrate coding and conducting inference on a stochastic SIR model.

We use the stochastic volatility model outlined in Pitt and Shephard (1999a). Let r_t be the exchange rate at time t, and define y_t as 100 times the daily log return, that is, $y_t = 100 \times (log(r_t) - log(r_{t-1}))$ for $t = 2, \ldots, T$, 100 times the daily log return. Our stochastic volatility model is then

$$y_t = \varepsilon_t \beta \exp(\frac{x_t}{2}), \quad \varepsilon_t \sim N(0,1)$$
$$x_t = \phi x_{t-1} + \nu_t, \quad \nu_t \sim N(0, \sigma^2)$$

where $N(\mu, \sigma^2)$ denotes the normal distribution with mean μ and variance σ^2. In this model, β can be interpreted as the constant volatility, while x_t is the latent, evolving volatility. Following Pitt and Shephard (1999b), prior distributions are placed on the parameters β, ϕ, and σ as follows:

$$\phi^* \sim B(18,1), \quad \phi = 2\phi^* - 1$$
$$\sigma \sim IG(5, \frac{1}{20})$$
$$\beta \sim IG(5, \frac{1}{20})$$

Above, $B(a,b)$ denotes the beta distribution with parameters a and b, and $IG(c,d)$ denotes the inverse gamma distribution with shape parameter c and scale parameter d. The stochastic volatility model can be written in BUGS code as

```
> stochVCode <- nimbleCode({
+  x[1] ~ dnorm(mean = phi*x0, var = sigma^2);
+  y[1] ~ dnorm(0, var = (beta*exp(x[1]/2))^2 );
+  eps[1] ~ dnorm(0,1)
+  for(t in 2:T){
+    x[t] ~ dnorm(mean = phi*x[t-1], var = sigma^2);
+    y[t] ~ dnorm(0, var = (beta*exp(x[t]/2))^2 );
+  }
+
+  x0 ~ dnorm(mean = 1, var = sigma^2);
+  phiStar ~ dbeta(18, 1);
+  phi <- 2*phiStar - 1;
+  sigma <- 1/sigmaInv;
+  sigmaInv ~ dgamma(5, 20);
```

```
+  beta <- 1/betaInv;
+  betaInv ~ dgamma(5, 20);
+ })
```

We use as data for our model exchange rates for the U.S. Dollar (USD) quoted in Euros (EUR) starting on January 1st, 2012, and continuing for 66 days after that. This data set can be found in the **stochvol** R package (Kastner, 2016). The **stochvol** package also includes a `logret` function to calculate log returns.

```
> library(stochvol)
> data(exrates)
> y <-  100*logret(exrates$USD[exrates$date>"2012-01-01"])
```

We next create and compile a `nimbleModel` for the above BUGS code, again following Pitt and Shephard (1999a) by using as starting values $\beta = .5992$, $\phi = .9702$, $\sigma = .178$, and providing T as a constant.

```
> stochVolModel <- nimbleModel(code = stochVCode, name='stochVol',
+                    data = list(y = y), constants = list(T = 67),
+                    inits = list(beta = .5992,  phi = .9702,
+                                 sigma = .178))
> compileNimble(stochVolModel)
```

To build a Liu and West filter (as detailed in Section 2.2.5) for the stochastic volatility model in **NIMBLE**, we use the `buildLiuWestFilter` function. The `buildLiuWestFilter` function has a similar syntax to the Bootstrap and Auxiliary particle filters. One key difference is that the Liu and West filter requires specification not only of the latent states (via the `nodes` argument), but also of the top level parameters to be estimated (via the `params` control list argument). Additionally, the Liu and West filter does not return an estimate of the log likelihood of the data.

```
> stochVolLiuWestFilter <- buildLiuWestFilter(model = stochVolModel, nodes = "x",
+                  control = list(params = c( "betaInv", "phiStar", "sigmaInv"),
+                                 saveAll = F))
> cLiuWestFilter <- compileNimble(stochVolLiuWestFilter, project = stochVolModel)
> cLiuWestFilter$run(5000)
```

Once the Liu and West filter has been run, we can extract the posterior distribution of top-level parameters The code below creates a histogram of the posterior distribution of the σ parameter.

```
> sigmaSamples <- 1/as.matrix(cLiuWestFilter$mvEWSamples, 'sigmaInv')
```

```
> hist(sigmaSamples, xlab = "Sigma", main ="")
```

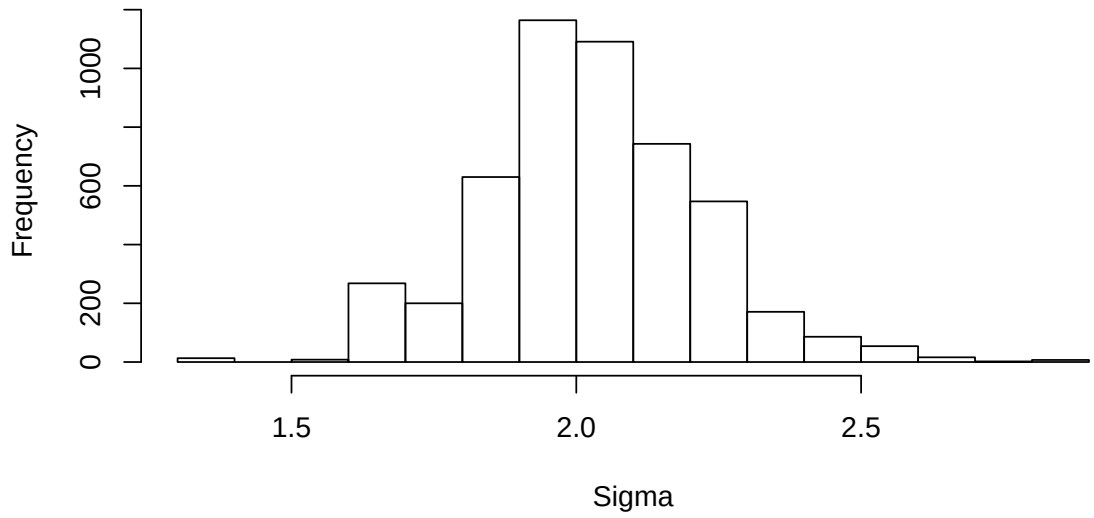

Figure 2.4: Histogram of the posterior distribution of σ.

An alternative set of methods for inference on top-level parameters in state space models is Particle MCMC, detailed in Section 2.2.6. **NIMBLE** has a Particle Marginal Metropolis Hastings (PMMH) algorithm included in its library of MCMC samplers. The PMMH sampler takes advantage of **NIMBLE**'s existing MCMC framework, in which specific samplers can be specified for different nodes in a model. For a full description of **NIMBLE**'s MCMC capabilities, reference Chapter 7 of the **NIMBLE** User Manual (NIMBLE Development Team, 2015).

The PMMH sampler uses a normal proposal distribution, and can be applied to sample either scalar parameters (using the `RW_PFilter` sampler) or vectors of parameters (using the `RW_PFilter_block` sampler) . To implement the PMMH algorithm, we first set up an MCMC specification for our stochastic volatility model using the `configureMCMC` function. The PMMH sampler can be added to the MCMC specification with a call to the `addSampler` function. Additional options to customize the sampler can be specified within the `control` list.

```
> stochVolMCMCSpec <- configureMCMC(stochVolModel, nodes = NULL,
+                     monitors = c('beta', 'phi', 'sigma', 'x'),
+                     thin = 20)
```

```
> stochVolMCMCSpec$addSampler(
+   target = c('betaInv', 'phiStar', 'sigmaInv', 'x'),
+   type = 'RW_PF_block',
+   control = list(propCov = .1*diag(3),
+                  adaptive = TRUE,
+                  latents = 'x',
+                  resample = T
+   )
+ )
```

The `control` list argument is used to set the initial proposal covariance, here specified to be a 3×3 diagonal matrix with 0.1 entries on the diagonal. Additionally, the PMMH sampler can be set use an adaptive algorithm to tune the proposal covariance matrix as the algorithm runs. The `resample` argument allows the algorithm to resample $p(\hat{y}_{1:T}|\theta^{i-1})$ at the beginning of each iteration (before Step 4 in Algorithm 4). This can help to reduce the chance that the algorithm gets "stuck" at a particular set of parameters due to a high marginal likelihood estimate. High likelihood estimates can arise naturally due to the stochastic nature of particle filter likelihood estimation.

Once the PMMH sampler is added to the MCMC specification, the algorithm can built using the `buildMCMC` function, and then compiled. Posterior samples are stored in `cMCMC$mvSamples`. Below we demonstrate running **NIMBLE**'s PMMH algorithm for 1,400 iterations. Note that the first 1,000 samples of our MCMC output are discarded as a burn-in period.

```
> stochVolMCMC <- buildMCMC(stochVolMCMCSpec)
> cMCMC <- compileNimble(stochVolMCMC, project = stochVolModel, resetFunctions = T)
> cMCMC$run(1400)
> mcmcOut <- as.matrix(cMCMC$mvSamples)[-c(1:1000),]
```

The **coda** package provides tools for analyzing MCMC output (Plummer et al., 2006). The code below creates a trace plot and posterior density plot for the β parameter in our model.

```
> library(coda)
> mcmcOut <- as.mcmc(mcmcOut)
> traceplot(mcmcOut[,'beta'],
+           main = "Trace plot for Beta")
```

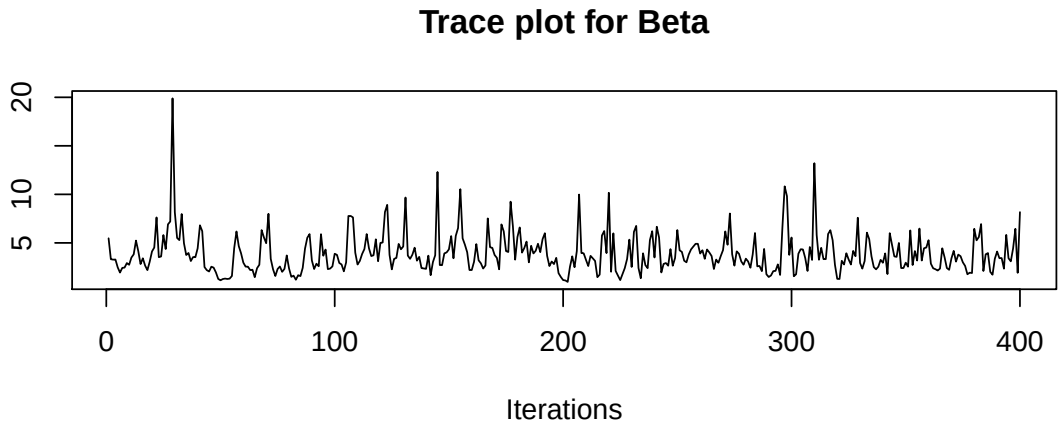

Figure 2.5: Trace plot for β.

```
> densplot(mcmcOut[,'beta'], main = "Density plot for Beta")
```

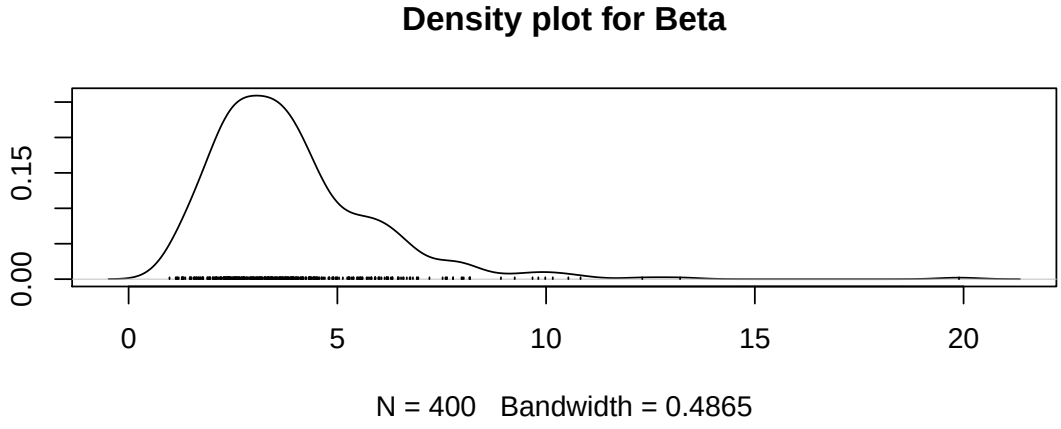

Figure 2.6: Posterior density estimate for β over 400 iterations of NIMBLE's particle MCMC algorithm.

We next build a stochastic chain binomial SIR model in NIMBLE. The chain binomial SIR model is a compartmental model which partitions a population at time t into three compartments: the S_t compartment, in which population members are susceptible to infection from a disease, the I_t compartment, in which population members have contracted a disease and are capable of spreading it, and the R_t compartment, where population members have recovered

from the disease and are no longer capable of being infected. Transitions between compartments are defined by

$$\begin{aligned} S_t &= S_{t-1} - \Delta_{1,t} \\ I_t &= I_{t-1} + \Delta_{1,t} - \Delta_{2,t} \\ R_t &= R_{t-1} + \Delta_{2,t} \\ \Delta_{t,1} &\sim \text{Binom}(S_{t-1}, 1 - e^{-\beta \frac{I_{t-1}}{N}}) \\ \Delta_{t,2} &\sim \text{Binom}(I_{t-1}, 1 - e^{-\gamma}) \end{aligned}$$

where β is a parameter which measures the number of transmission-sufficient contacts made by a single individual in S per time interval, and γ is a parameter which measures the average amount of time a population member spends in the I compartment. Additionaly, initial states for the compartments are set by

$$\begin{aligned} S_0 &= N - I_0 \\ I_0 &\sim \text{Binom}(N, p_0) \\ R_0 &= 0 \end{aligned}$$

For more information on the chain binomial specificaiton for an SIR model, see Lekone and Finkenstädt (2006). We additionally use ILINet data to inform our model. ILINet data provides an estimate of the proportion of all individuals in the U.S. who are currently infected with an influenza-like illness. We relate ILINet data, labeled ILI_t at time t to the latent I state of our SIR model by

$$ILI_t = \text{logit}\left(\frac{I_t}{N}\right) + \epsilon_t$$

where $\epsilon_t \sim N(0, \sigma_I^2)$ is a noise term. Thus the logit of ILINet data is assumed to be an unbiased, noisy obersvation of the logit of the I compartment.

Define $\theta_t = (\Delta_{1,t}, \Delta_{2,t}, S_t, I_t, R_t)$. The above chain binomial SIR model can be written in BUGS code as

```
> SIRCode <- nimbleCode({
+
```

```
+  probIR <- 1 - exp(-gamma)
+
+  IStart ~ dbinom(p0, N)
+
+  theta[1,1] <- 0
+  theta[2,1] <- 0
+  theta[3,1] <- N - IStart
+  theta[4,1] <- IStart
+  theta[5,1] <- 0
+
+  ILINet[1] ~ dILI(theta[1:5,1], sigmaI = sigmaI, N = N)
+  for(t in 2:EndWeek){
+    theta[1:5, t] ~ dSIR(theta[1:5, t-1], probIR, beta, N)
+    ILINet[t] ~ dILI(theta[1:5,t], sigmaI = sigmaI, N = N)
+  }
+
+  beta ~ dgamma(5, 10)
+  gamma ~ dgamma(5, 10)
+  p0 ~ dbeta(1, 20)
+  sigmaI ~ T(dt(0, 2, 10), 0, )
+ })
```

The above BUGS model takes advantage of another of NIMBLE's features: user-defined distributions. NIMBLE allows users to specify their own distributions, which can in turn be used generically in any BUGS code. Our SIR model includes two user-defined distributions: the θ_t vector follows a `dSIR` distribution, and ILI_t follows a `dILI` distribution. We provide an example of a user-defined distribution via the `dSIR` density function below.

```
> dSIR <- nimbleFunction(
+  run = function(theta = double(1), theta_previous = double(1),
+                 probIR = double(0), beta = double(0), N = double(0),
+                 log = integer(0, default = 0)) {
+
+    returnType(double(0))
+
+    S_previous <- theta_previous[3]
+    I_previous <- theta_previous[4]
+    Delta_1 <- theta[1]
+    Delta_2 <- theta[2]
+
```

```
+    lambda <- beta*I_previous/N
+    probSI <- 1 - exp(-lambda)
+
+    p1 <- dbinom(Delta_1, S_previous, probSI)
+    p2 <- dbinom(Delta_2, I_previous, probIR)
+
+    p <- p1*p2
+    if(log)
+      p <- log(p)
+    return(p)
+  })
```

The `dSIR` density function returns the product of the two binomial probabilities which specify the transition from S_t, I_t, R_t to $S_{t+1}, I_{t+1}, R_{t+1}$. For more information on creating user-defined distributions, see Chapter 5.2.5 of the NIMBLE user manual (NIMBLE Development Team, 2015)

Up to date ILINet data can be obtained using the **cdcfluview** R package (Rudis, 2015). We use as data for our model ILINet data from MMWR week 40 of 2015 through MMWR week 20 of 2016. Below we load the ILINet data and create our `nimbleModel`.

```
> library(cdcfluview)
> ILINet <- get_flu_data(region = "national",
+                        data_source = "ilinet",years = 2015)
> ILINet <- ILINet$X.UNWEIGHTED.ILI[1:33]/100
>
> SIRModel <- nimbleModel(code = SIRCode, name='sirModel',
+                    data = list(ILINet = ILINet),
+                    constants = list(EndWeek = 33,
+                                     N = 10000),
+                    inits = list(beta = .5, gamma = .45,
+                                 p0 = 0.02, sigmaI = .1))
> compileNimble(SIRModel)
```

We conduct inference on the parameters of our SIR model using NIMBLE's block PMCMC sampler. Below, we construct the sampler and run it for 6,000 iterations, using a thinning interval of 3.

```
>   SIRMCMCSpec <- configureMCMC(SIRModel, nodes = NULL,
+                        monitors = c('beta', 'gamma', 'p0', 'sigmaI', "theta"),
+                        thin = 3)
```

```
>   SIRMCMCSpec$addSampler(
+     target = c('beta', 'gamma', 'p0', 'sigmaI'),
+     type = 'RW_PF_block',
+     control = list(propCov = .05*diag(4),
+                    adaptive = TRUE,
+                    adaptInterval = 250,
+                    latents = 'theta',
+                    resample = T,
+                    m = 1500)
+   )
> SIRMCMC <- buildMCMC(SIRMCMCSpec)
> cSIRMCMC <- compileNimble(SIRMCMC, project = SIRModel)
> cSIRMCMC$run(6000)
> SIROut <- as.matrix(cSIRMCMC$mvSamples)
```

Once the algorithm has been run, we can view trace plots of the β and γ parameters of our SIR model using the code below. Notably, the β parameter does not seem to be mixing very well. This could possibly be resolved by running the algorithm for more iterations or increasing the number of particles used to estimate the likelihood at each iteration.

```
> traceplot(as.mcmc(SIROut[,'beta']), main = "Trace plot for Beta")
```

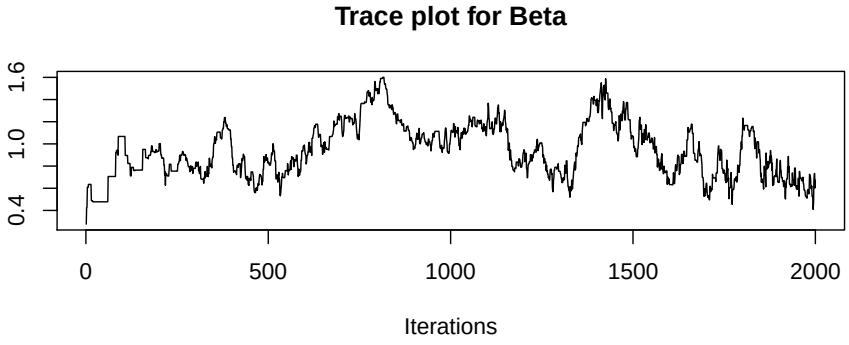

Figure 2.7: Trace plot for the β parameter of a chain binomial SIR model.

```
> traceplot(as.mcmc(SIROut[,'gamma']), main = "Trace plot for Gamma")
```

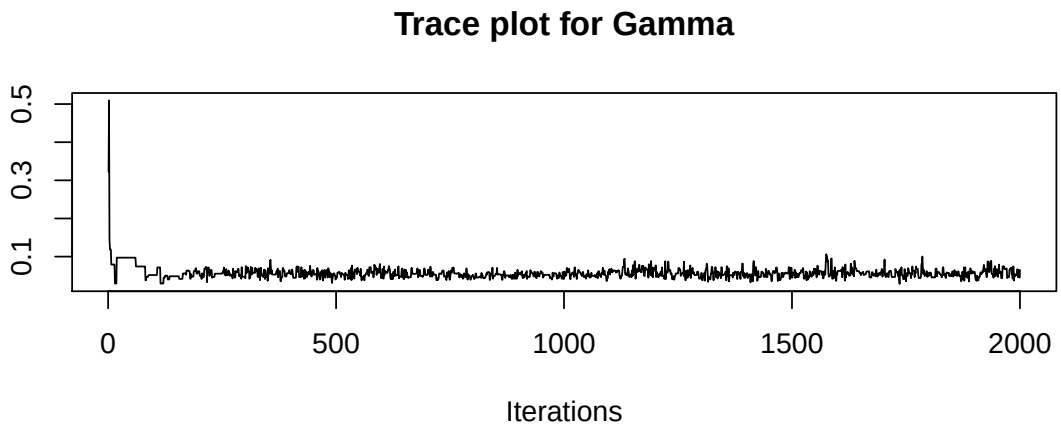

Figure 2.8: Trace plot for the γ parameter of a chain binomial SIR model.

2.4 Programming SMC Algorithms in NIMBLE

In this section, we demonstrate how **NIMBLE** can be used to program model-generic algorithms, that is, algorithms that can be applied to any model written in BUGS. In Section 2.4.1, we give an overview of how functions are written in **NIMBLE**. In Section 2.4.2 demonstrate NIMBLE's model-generic approach to programming by providing code for an example SMC algorithm written in **NIMBLE**'s DSL.

2.4.1 Writing Functions in NIMBLE

We first provide an overview of the basic concepts of programming in **NIMBLE** by describing the use of `setup` and `run` code, the use of `modelValues` objects, and the `calculate` and `simulate` functions. A more detailed presentation of writing functions in **NIMBLE** can be found in Chapter 9 of the NIMBLE user manual (NIMBLE Development Team, 2015).

2.4.1.1 setup Code and run Code

Functions written in **NIMBLE** consist of two different types of code: `setup` code and `run` code. When a function is called in **NIMBLE**, the setup code is evaluated first. `setup` code is written in R, and is primarily used to extract model information for later use in the `run` code.

For this reason, `setup` code is often provided with **NIMBLE** model objects as arguments, as well as additional parameters governing the algorithm to be run. As an example, the setup code for building an Auxiliary particle filter (as seen in Section 2.3.2) is run by calling the `buildAuxiliaryFilter` function. The `buildAuxiliaryFilter` function is provided with the name of the model that is being used, as well as additional information specifying the details of the filter. Since `setup` code is written in R, it can use any available R function to assist in preparing the algorithm to be run.

After `setup` code has been used to prepare the algorithm, `run` code is executed. `run` code is written in the **NIMBLE** domain specific language (DSL), and can only use functions that are included within that DSL. The functions available in the **NIMBLE** DSL can be thought of as a narrow subset of R functions. Writing within the **NIMBLE** DSL allows `run` code to be compiled into C++, in turn providing efficient execution of an algorithm's computations. `run` code can make use of objects created in the `setup` code. Although `run` code doesn't have access to the full range of R functions, it can make use of objects created in the `setup` code.

2.4.1.2 modelValues Objects

Models in **NIMBLE** are made up of nodes – for example, state-space models have nodes for parameters, for data, and for latent states. Each node in a **NIMBLE** model will hold a single value for the variable it represents. However, in many situations we may be interested in considering multiple values for a single node in a model. For example, in the Bootstrap filter of Section 2.2.3, each particle $x_t^{(p)}$ provides a different value for the x_t node in a state space model.

`modelValues` objects provide a container for storing multiple values of model nodes in **NIMBLE**. `modelValues` objects can be built by creating a specification with the `modelValuesSpec` function, which is used to describe the names, types, and sizes of the nodes to be stored. Once a `modelValues` object has been created, the `copy` function is used to copy node values from a model into a `modelValues` object, from a `modelValues` object into a model, or from one `modelValues` object to another. As will be seen in Section 2.4.2, a `modelValues` object is used to store the values of particles at each time point.

2.4.1.3 The calculate and simulate Functions

Two functions in **NIMBLE**'s DSL, named `calculate` and `simulate`, provide much of the functionality necessary to implement an SMC algorithm. The `calculate` function takes as arguments the name of a model, and the name of one or mode nodes in that model. If `calculate` is called on a stochastic node, the function returns the log density of that node. If `calculate` is called on a deterministic node, the function calculates the value of that node and returns 0. The `simulate` function also takes as arguments the name of a model and the name of one or mode nodes in that model. If `simulate` is called on a stochastic node, it draws a random value from that node's distribution. If `simulate` is called on a deterministic node, it is equivalent to the `calculate` function. As an example of the use of these two functions, we provide a very simple model below. We first `simulate` a value for the stochastic `x` node in this model, and then `calculate` the value of the deterministic `y` node.

```
> simCalcExample <- nimbleCode({
+  x ~ dnorm(mean = 0, var = 1);
+  y <- x + 5;
+ })
> simCalcExampleModel <- nimbleModel(code = simCalcExample)
> simulate(simCalcExampleModel, 'x')
> print(simCalcExampleModel$x)

[1] -0.9780563
> calculate(simCalcExampleModel, 'y')

[1] 0
> print(simCalcExampleModel$y)

[1] 4.021944
```

Use of the `calculate` and `simulate` functions allows **NIMBLE** algorithms to interact with models in a generic manner, as no information about the distributions of the nodes to be calculated or simulated needs to be provided to the `calculate` or `simulate` functions. As such, the use of `calculate` and `simulate` plays a central role in **NIMBLE**'s SMC algorithms. In the example SMC algorithm of Section 2.4.2, the `simulate` function is used to simulate particles for latent states at the current time point conditioned on particles from the previous time point, and the `calculate` function is used to calculate weights for these particles.

2.4.2 An Example SMC Algorithm

We now demonstrate programming in **NIMBLE** by providing code for a Bootstrap filtering algorithm. Note that the code shown below is simpler than the actual implementation of the Bootstrap filter available in **NIMBLE** through the `buildBootstrapFilter` function. However, this demonstration code is indeed a fully functional Bootstrap filter – the Bootstrap filter included in the **NIMBLE** package simply has more customization options than the demonstration shown here.

The example Bootstrap filter provided is comprised of two separate **NIMBLE** functions, each with their own `setup` and `run` code. The first function, named `bootstrapFilter`, extracts necessary information from the state space model, including the names and dimensions of the latent nodes to be sampled. It then iterates through time, at each time point t calling the second function, `bootstrapStep`. The `bootstrapStep` function takes information about the latent state at time t, and then conducts Steps 7 through 13 of the Bootstrap filter algorithm given in Section 2.2.3. As the filtering algorithm progresses through each time point, samples from the filtering distribution at that time are saved in a **NIMBLE** `modelValues` object.

Below is the `setup` and `run` code for the `bootstrapFilter` function. Functions in **NIMBLE** are specified using the `nimbleFunction` function. Note that the `setup` code for the `myBootstrapFilter` function is provided with the **NIMBLE** model object and the names of the latent states as arguments. The setup code first defines a function that will initialize the model, and then obtains the names and dimensions of the latent states in the model in chronological order. A `modelValues` object is then created to store samples from the latent states. Note that the `modelValues` object will store two different sets of samples from the latent states: `x`, which will have non-equally weighted samples, and `xEW`, which will have equally weighted samples. Finally, the `setup` code creates a list of `bootstrapStep` functions (called a `nimbleFunctionList`). For each time point $t = 1, \ldots, T$, the list contains one `bootstrapStep` function that will conduct a Bootstrap filtering algorithm at that time.

```
> bootstrapFilter <- nimbleFunction(
+     setup = function(model, latentNodes) {
+         my_initializeModel <- initializeModel(model)
```

```
+        latentNodes <- model$expandNodeNames(latentNodes, sort = TRUE)
+        dims <- lapply(latentNodes, function(n) nimDim(model[[n]]))
+        mvSpec <- modelValuesSpec(vars = c('x', 'xEW'),
+                                  types = c('double', 'double'),
+                                  sizes = list(x = dims[[1]], xEW = dims[[1]]))
+        mv <- modelValues(mvSpec)
+        bootStepFunctions <- nimbleFunctionList(bootstrapStepVirtual)
+        timePoints <- length(latentNodes)
+        for(t in 1:timePoints)
+          bootStepFunctions[[t]] <- bootstrapStep(model, mv, latentNodes, t)
+    },
+    run = function(P = integer()) {
+        my_initializeModel$run()
+        resize(mv, P)
+        for(t in 1:timePoints)
+             bootStepFunctions[[t]]$run(P)
+
+     })
```

The `run` code for the `bootstrapFilter` function takes as its only input argument the number of particles (P) to use for estimation. `run` code requires explicit specification of the type of any input arguments, so here P is specified as an integer object. In general, the type of object to be returned must also be specified, although this function does not return any objects so no specification is necessary. The `run` function first initializes the model (conducting Steps 2 and 3 of the Bootstrap filter algorithm), and then resizes the `modelValues` object so that it can store P particles. After that, the `run` function iterates through each time point, running the `bootstrapStep` function that was defined for that time point in the `setup` code.

Creating a `nimbleFunctionList`, such as the one used in the `setup` code above, requires an additional piece of code that informs **NIMBLE** about the input arguments and return objects of each function in that list. Specifically, the `nimbleFunctionVirtual` function is used to define the attributes that each function in the `nimbleFunctionList` will have. Below, we specify that each element of our `nimbleFunctionList` will have a run function with a single integer input.

```
> bootstrapStepVirtual <- nimbleFunctionVirtual(
+  run = function(P = integer()) {}
+ )
```

`setup` and `run` code for the `bootstrapStep` function is given below. At each time point t, the `setup` function gets the names and deterministic dependencies of the previous and current latent states. The `run` code first declares a length P vector of doubles and a length P vector of integers using the `declare` function. The `run` code then iterates through each particle. For each particle, the code takes the value of the latent state at $t-1$ from the `modelValues` object, uses that value to simulate a value for the latent state at time t, and calculates a weight. Particles are then resampled proportional to their weights and stored in the `mv` object. Note that the `rankSample` function fills the elements of the `ids` vector with the indices of the particles that have been chosen in the resampling procedure.

```
> bootstrapStep <- nimbleFunction(
+    contains = bootstrapStepVirtual,
+    setup = function(model, mv, latentNodes, timePoint) {
+        notFirst <- timePoint != 1
+        prevNode <- latentNodes[if(notFirst) timePoint-1 else timePoint]
+        thisNode <- latentNodes[timePoint]
+        prevDeterm <- model$getDependencies(prevNode, determOnly = TRUE)
+        thisDeterm <- model$getDependencies(thisNode, determOnly = TRUE)
+        thisData   <- model$getDependencies(thisNode, dataOnly = TRUE)
+    },
+    run = function(P = integer()) {
+        ids <- integer(P, 0)
+        wts <- numeric(P, 0)
+
+        for(p in 1:P) {
+            if(notFirst) {
+              copy(from = mv, to = model, nodes = 'xEW',
+                   nodesTo = prevNode, row = p)
+              calculate(model, prevDeterm)
+            }
+            simulate(model, thisNode)
+            copy(from = model, to = mv, nodes = thisNode,
+                 nodesTo = 'x', row = p)
+            calculate(model, thisDeterm)
+            wts[p] <- exp(calculate(model, thisData))
+        }
+        rankSample(wts, P, ids)
```

```
+        for(p in 1:P)
+            copy(from = mv, to = mv, nodes = 'x',
+                 nodesTo = 'xEW', row = ids[p], rowTo = p)
+ })
```

Once the `nimbleFunctions` have been defined, we can build, compile, and run the Bootstrap filter. Samples from the filtering distribution are stored in the `mv` `modelValues` object.

```
> myBootstrap <- bootstrapFilter(exampleModel, 'x')
> cmyBootstrap <- compileNimble(myBootstrap, project = exampleModel)
> cmyBootstrap$run(1000)
> filterSamps <- as.matrix(cmyBootstrap$mv)
> hist(filterSamps[,'xEW[1]'], main = "", xlab = "")
```

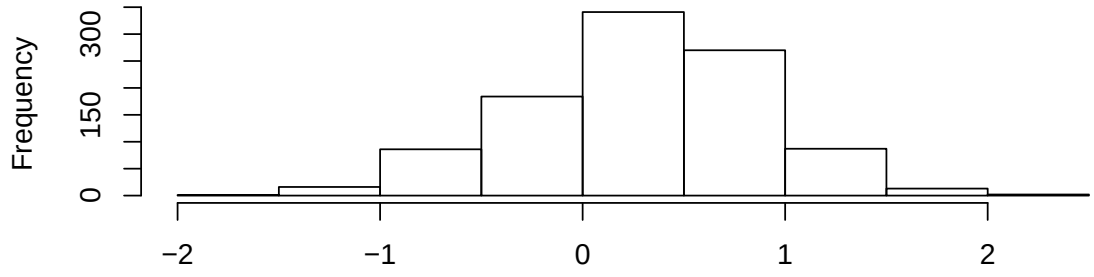

Figure 2.9: Histogram of filtering distribution of x

The Bootstrap filter code provided above demonstrates **NIMBLE**'s ability to program model-generic algorithms. The filter could be used to conduct filtering on any correctly specified state space model. In addition to the generality of the algorithm, it would be relatively straightforward to modify the filter, changing it to an Auxiliary Particle filter, a Liu and West filter, or a filter type not currently included in **NIMBLE**. The ease with which existing algorithms can be modified, along with the generality with which they are written, promotes the development of user-written filters in a manner previously unavailable in R.

2.5 Conclusion

This paper has described **NIMBLE**'s suite of SMC algorithms, which provide a straightforward method of conducting inference on state space models. In addition, **NIMBLE**'s model-generic programmability make it perfectly suited for implementing new SMC algorithms, an example of which was given in Section 2.4.2. **NIMBLE**'s flexible model specification also enables the application of existing algorithms to models that do not fall into the traditional state space model framework. For example, a model could be specified where a number of state space models are set within a larger hierarchical structure. Using **NIMBLE**, SMC algorithms could be used to estimate the individual state space models, while an MCMC algorithm could conduct inference on higher-level parameters.

Additional examples of modeling and inference using **NIMBLE** can be found at `http://r-nimble.org/`.

CHAPTER 3. Evaluating the Use of Biased Data in Disease Forecasting

3.1 Introduction

Seasonal influenza epidemics account for millions of illnesses and thousands of hospitalizations in the United States each year (Thompson et al., 2004). For the 2013 - 2014 season, the rate of hospitalization in the U.S. due to confirmed influenza virus infections was 35.6 per 100,000 people (Epperson et al., 2014). Due to the scale of these yearly influenza epidemics, accurate forecasting of epidemic severity is important in helping hospitals to prepare for a potential influx of patients. In addition, epidemic severity can be used to detect new influenza strains by monitoring for severity patterns not expected to be produced by seasonal epidemics. However, accurately forecasting the course of epidemics has proven challenging, spurring much recent research into outbreak forecasting methods (Chretien et al., 2014).

Many such forecasting methods make use of data from the U.S. Outpatient Influenza-like Illness Surveillance Network (ILINet), provided by the U.S. Centers for Disease Control and Prevention (CDC). ILINet data are obtained using a sentinel surveillance network, in which participating hospitals and medical centers around the country report the proportion of patients with an influenza-like illness. The proportions provided by ILINet are considered the gold standard for influenza data in the United States, as used in Yang et al. (2015), Paul et al. (2014), and others, and provide a good basis upon which to make predictions. Statistical models have been made which use ILINet data to monitor and predict influenza epidemic severity, such as in Goldstein et al. (2011), which predicts the size of epidemics for individual influenza strains by modeling their correlation with other concurrent strains. However, ILINet data are typically reported at a lag of one to two weeks. Thus, a prediction of next week's epidemic severity using only ILINet data would in fact require forecasting two to three weeks beyond the scope of the

data.

More timely indicators of epidemic severity than ILINet data exist, and have also been used in models to predict epidemic severity. Data from Twitter (Paul et al., 2014) and Wikipedia (Hickmann et al., 2015) have both demonstrated an ability to increase predictive capabilities over models with ILINet data alone. Another timely indicator of epidemic severity is Google Flu Trends (GFT), which provides frequently updated outbreak severity data based on the number of searches for certain influenza-related terms in the Google search engine (Ginsberg et al., 2009). Recent attempts at severity prediction using GFT data include Dugas et al. (2013), which found that incorporating GFT data in a GARMA time series model increased prediction accuracy, and Shaman and Karspeck (2012), who presents a method of using GFT data within an SIRS model of influenza epidemics. Shaman and Karspeck (2012) later demonstrate their model's forecasting ability by applying it to the 2012 - 2013 influenza season in New York City (Shaman et al., 2013). Lampos et al. (2015) use a refined set of Google keywords, chosen by an Elastic Net procedure, to provide predictions of future epidemic severity. However, despite its recency as compared to ILINet data, GFT data have been observed (Lazer et al., 2014) to be a biased estimate of influenza severity in some seasons, requiring modelers to use care when using it as a basis for forecasts. Some recent papers (Nsoesie et al., 2013) have used GFT alone to forecast features of seasonal influenza outbreaks. Others (Nishiura et al., 2011) neglect to use GFT or any crowd-sourced data.

In this paper, we propose the use of GFT in concert with ILINet data to forecast epidemic severity. Using multiple data streams which measure the same quantity of interest to produce refined forecasts is known as data fusion. Data fusion methods have been used in fields such as motion capturing (Zhang and Wu, 2006), genomics (Aerts et al., 2006), epidemiology (Berrocal et al., 2010), and meteorology (Barboza et al., 2014). Data fusion has the potential to provide enhanced prediction accuracy. For example, when forecasting influenza outbreaks, the use of GFT data in addition to ILINet data provides an extra two weeks worth of information on outbreak severity. However, the benefit of including additional, more up-to-date data sources like GFT is dependent on their accuracy. If the more recent data to be included are a biased estimate of epidemic severity, care must be taken to account for the bias, or forecasts may

actually be hindered by the use of the additional data.

In Section 3.2, we describe in detail the ILINet and GFT data sources. In Section 3.3, we propose two sets of modeling frameworks, one a DLM framework, and one an SIR framework, which can incorporate biased and unbiased sources of data to forecast the values of a latent state of interest. The models can account for data sources which vary in both accuracy and timeliness. In Sections 4.4 and 3.5, simulation studies are conducted to examine the effect of including a biased data source, and the effect of explicitly modeling the bias in that data source, on forecasting ability. In Section 3.6, the models are applied to influenza forecasting using real ILINet and GFT data. Section 3.7 discusses the results, and makes a case for explicitly modeling sources of bias.

3.2 Data

Our models use both ILINet data provided by the CDC and GFT data provided by Google. In Sections 3.2.1 and 3.2.2, we provide details about how each of these measures of influenza severity are calculated.

3.2.1 CDC ILINet Data

The U.S. Outpatient Influenza-like Illness Surveillance Network (ILINet) is a syndromic surveillance network of more than 3,300 health care providers, spanning all 50 states, which collects weekly information on the proportion of patients who visit for influenza-like illness (Brammer et al., 2011). An influenza-like illness is defined as a fever of greater than 100° Fahrenheit accompanied by a cough or sore throat, and for which no non-influenza cause has been identified (CDC, 2016). Each health care provider which participates in ILINet reports their weekly total number of patient visits, and their weekly number of patient visits for an influenza-like illness. Using this information, the overall proportion of patient visits due to an influenza like illness is calculated for Health and Human Services regions and Census divisions, as well as a proportion for the entire U.S.. IlInet data are typically reported at a lag of one to two weeks. In addition, as the ILINet data are dependent on volunteer submissions, it is also subject to revision by the CDC as more submissions are received.

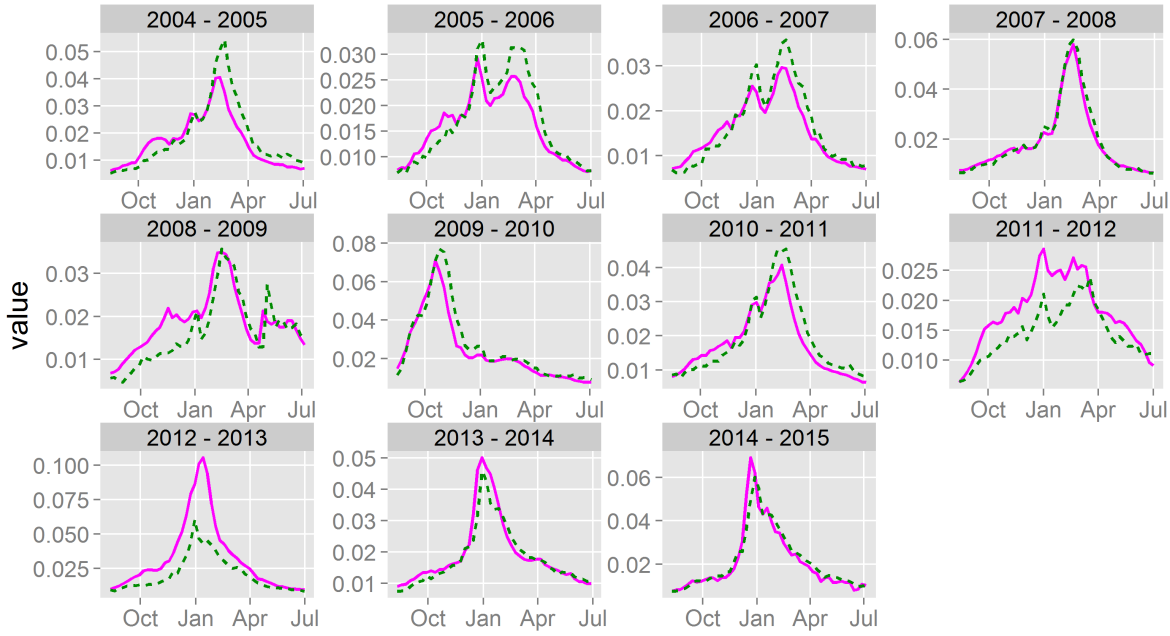

Figure 3.1: Weekly indicators of influenza-like illness in the United States across 11 seasons for the CDC's ILINet sentinel network (pink solid lines) and Google's Google Flu Trends (green dashed lines).

ILINet data are published on a weekly basis, and is recorded along with its Morbidity and Mortality Weekly Report (MMWR) week. Information on obtaining ILINet data and calculating MMWR weeks can be found in Appendix Section A.5. Our data models, presented in Section 3.3, use ILINet data on the national scale for the United States, starting with the 2004 – 2005 influenza season and ending with the 2013 – 2014 season. Similar to Yang et al. (2014), we omit the 2008 – 2009 and 2009 – 2010 seasons, as the data from both seasons show evidence of the A/H1N1 influenza pandemic. In the 2008 – 2009 season, evidence of the pandemic can be seen in the late rise of the proportion infected around week 40. In the 2009 – 2010 season, the pandemic causes the outbreak curve to rise more quickly and peak at an earlier week than a seasonal outbreak would. Figure 3.1 displays the influenza outbreak curves for these seasons, also showing the discrepancy in curve shape for the 2008 – 2009 and 2009 – 2010 seasons. Omitting these seasons seems prudent, as our interest lies in forecasting seasonal influenza outbreaks, rather than pandemics.

3.2.2 Google Flu Trends Data

Google Flu Trends provides estimates of influenza activity around the world by monitoring the frequency of searches for certain influenza-related keywords (Ginsberg et al., 2009). GFT data are provided as the expected number of influenza cases per 100,000 people, and is available on the city, regional, and national scales. We divide this data by 100,000 to put it on the same scale as the ILINet data .

The search keywords used by Google to produce GFT data are shown to be highly correlated with historical ILINet data (Ortiz et al., 2011). In addition, Google Flu Trends estimates are provided in "near real-time" (Google, 2016), making them a more current counterpart to the ILINet data. However, Google Flu Trends is also a less direct indicator of influenza prevalence than the ILINet data. GFT data are dependent upon user searches through the Google search engine, which in turn could be influenced not only by users experiencing influenza like symptoms, but also by events without a direct relation to a season's influenza outbreak - for example, excess media coverage, or symptoms from non-influenza diseases could both inflate the estimate provided by GFT. Evident in Figure 1, there are a few seasons where GFT data and ILINet data provide markedly different curves. For example, GFT provides a peak infected proportion more than twice as large as ILINet's in the 2012–2013 season. Lazer et al. (2014) discuss some possible reasons for the observed inaccuracy in GFT data, including the possibility that Google has modified its search algorithm over time, guiding users towards certain search terms, which could necessitate a recalibration of their GFT algorithm. Since Google has not published the keywords they use to derive the GFT index, it is difficult to know whether these search engine changes have been accounted for.

3.3 Models for Forecasting Epidemics

State space models are commonly used to model time series data and other data that arrives sequentially. State space models assume a latent state, labeled x_t at time t, which evolves over time according to a stochastic evolution equation $f(x_{t+1}|x_t, \theta)$, where θ is a vector of parameters. Data at each time point, labeled y_t, is related to the unobserved latent state

through a stochastic observation equation $f(y_t|x_t, \theta)$. State space models also assume a Markov property for the latent states, where $f(x_t|x_{1:t-1}, \theta) = f(x_t|x_{t-1}, \theta)$. When modeling disease epidemic data, interest often lies in the predictive distribution $f(x_{t+k}|y_{1:t}, \theta)$ and the forecasting distribution $f(y_{t+k}|y_{1:t}, \theta)$ for some $k > 0$.

Oftentimes compartmental state space models, such as the stochastic SIR or SEIR models, are used to conduct inference on epidemics. Compartmental models conceptualize a population as being partitioned into some number of distinct compartments. For example, an SIR model partitions a population into an S group, in which individuals are susceptible to contracting a disease, an I group, in which individuals have contracted a disease and are capable of spreading it, and an R group, for individuals who have recovered after having the disease. A broader overview of epidemiological compartmental models can be found in Brauer (2008). Compartmental models are widely used as they provide a biologically interpretable conceptualization of disease outbreaks.

A less common approach for analyzing epidemic data are dynamic linear models (DLMs). DLMs are a special subset of state space models for which both the observation and evolution equations are taken to be linear with a Gaussian error term. DLMs are specified by:

$$\begin{aligned} x_0 &= \mu_0 + w_0 \quad w_0 \sim N(0, W_0) \\ x_t &= M_t x_{t-1} + w_t \quad w_t \sim N(0, W_t) \\ y_t &= F_t x_t + v_t \quad v_t \sim N(0, V_t) \end{aligned} \tag{3.1}$$

where M_t and F_t are known, fixed matrices, and where $N(A, S)$ denotes a multivariate normal distribution with mean vector A and covariance matrix S.

DLMs are commonly used to analyze time series data, in subjects as diverse as meteorology (Aberson, 2003) and power systems (Douglas et al., 1998). On the other hand, DLMs are not often applied to epidemic data, as they lack the conceptual epidemiological underpinnings provided by compartmental models. However, as compared to compartmental models, DLMs have a number of desirable mathematical properties. Specifically, for dynamic linear models where μ_0, W_0, W_t, and V_t are known for all t, the Kalman filter (Kalman, 1960) provides an analytic solution to the predictive and forecasting distributions, and also provides an exact

calculation of the likelihood $f(y_{1:t}|\theta)$. Using DLMs and having access to these analytic solutions will allow the benefits of both incorporating biased data sources, and of explicitly modeling the bias in these sources, to be evaluated exactly.

In Section 3.3.1 we introduce a set of three DLMs with known parameters which incorporate two data streams – one unbiased data stream, and one biased data stream. The three DLMs differ in their treatment of the biased data stream. In Section 3.3.2, we define a similar set of three SIR models which are used to model one biased and one unbiased data stream. Examining both DLM and SIR models provides a comprehensive overview of the effect that bias modeling can have on predictive ability for epidemics; the DLM models, though not specifically designed for disease outbreak data, provide exact results, while the SIR models provide results in a conceptual epidemiological framework.

3.3.1 DLM Models for Biased Data

The three DLM models presented in this section represent different methods for modeling biased and unbiased data streams. For simplicity of exposition, we present models which incorporate a single biased and a single unbiased data stream. However, we note that it is trivial to add additional biased or unbiased sources of information to this model – see Appendix Section A.1 for a generic model. The first of the three models presented excludes the biased data stream entirely, using only the unbiased stream. The second model presented uses both data streams, but naively treats the biased data stream as a second unbiased source of data. Finally, the third model includes an evolving bias offset for relating the biased data stream to the latent quantity of interest.

Define the latent epidemic severity at time t as the proportion of the total population at time t that is infected with an ILI. Label the logit of the latent epidemic severity at time t as s_t. We begin by modeling the evolution of s_t as a local level model, defined by

$$s_t = s_{t-1} + w_{t,s} \qquad w_{t,s} \sim N(0, \sigma_s^2) \quad 1 \leq t \leq T \tag{3.2}$$

$$s_0 = \mu_s + w_{0,s} \qquad w_{0,s} \sim N(0, \sigma_s^2) \tag{3.3}$$

where $N(\mu, \sigma^2)$ represents a normal distribution with mean μ and variance σ^2.

The first data model, labeled the "ILINet Only Model", makes use of only the ILINet data stream. Label the logit of ILINet data at time t as ILI_t. As noted in Section 3.2.1, ILINet data are assumed to be an unbiased source of data on latent epidemic severity. Furthermore, ILINet data are reported at a time lag. Suppose that ILINet data are reported at a lag of δ time points, so that at time T ILINet data are available from time 1 up to time $T - \delta$. ILI_t is modeled as

$$ILI_t = s_t + v_{t,I} \qquad v_{t,I} \sim N(0, \sigma_I^2) \quad 1 \leq t \leq T - \delta \tag{3.4}$$

Thus the logit of ILINet data at time t is modeled as a noisy, unbiased estimate of logit severity at time t. Equations (3.2) – (3.4) define the "ILINet Only Model".

The next model, labeled the "ILINet and GFT Unbiased Model", uses both ILINet and GFT data, but treats GFT data as an unbiased data source. GFT data, described in Section 3.2.2, can be a biased source of data on epidemic severity. However, GFT data are reported in near real-time. Thus, we assume that at time T, GFT data are available from time 1 up to time T. Label the logit of GFT data at time t as GFT_t. The "ILINet and GFT Unbiased Model" models GFT data as

$$GFT_t = s_t + v_{t,G} \qquad v_{t,G} \sim N(0, \sigma_G^2) \quad 1 \leq t \leq T \tag{3.5}$$

Equations (3.2) – (3.5) define the "ILINet and GFT Unbiased Model".

The final DLM we present, labeled the "ILINet and GFT Biased Model", explicitly models the bias in GFT data. We again assume that GFT data are available from time 1 up to time T, and model the logit of GFT data as

$$GFT_t = s_t + \beta_t + v_{t,G} \qquad v_{t,G} \sim N(0, \sigma_G^2) \quad 1 \leq t \leq T \tag{3.6}$$

$$\beta_t = \beta_{t-1} + w_{t,\beta} \qquad w_{t,\beta} \sim N(0, \sigma_\beta^2) \quad 1 \leq t \leq T \tag{3.7}$$

$$\beta_0 = \mu_\beta + w_{0,\beta} \qquad w_{0,\beta} \sim N(0, \sigma_\beta^2) \tag{3.8}$$

Here β_t is a latent bias term which itself evolves over time according to a local level model. The logit of GFT data is modeled as a noisy, biased estimate of logit latent severity, where the additive bias at time t is equal to β_t. Modeling the bias as an evolving latent variable allows the

two data sources to start at similar values, as seen in Figure 3.1. As the season progresses, the bias is able to change to reflect the consistent overestimation (seen, for example, in the 2012–2013 influenza season) or underestimation (seen in the 2010–2011 influenza season) present in the GFT data. Equations (3.2) – (3.4) and (3.6) – (3.8) define the "ILINet and GFT Biased Model".

Comparing the forecasting ability of the "ILINet Only Model" to the two models which include GFT data will allow us to determine what benefit, if any, the use of biased data confers. Comparing the forecasts produced by the "ILINet and GFT Unbiased Model" to those produced by the "ILINet and GFT Biased" model will further allow us to evaluate whether and under what conditions explicitly modeling the bias leads to more accurate forecasts.

3.3.2 SIR Models for Biased Data

This section present three compartmental SIR models for modeling two data sources, one of which is biased, and one of which is unbiased. We begin by presenting a model for the latent epidemic states, and then define three data models to relate observed ILINet and GFT data to the latent states. Let S_t, I_t, and R_t be the susceptible, infectious, and recovered compartments respectively at time t. Further, assume a known and fixed population size $N = S_t + I_t + R_t$ for all t. The Chain Binomial model models movements between compartments as

$$\begin{aligned}
S_t &= S_{t-1} - \Delta_{1,t} \\
I_t &= I_{t-1} + \Delta_{1,t} - \Delta_{2,t} \\
R_t &= R_{t-1} + \Delta_{2,t} \\
\Delta_{t,1} &\sim \text{Binom}\left(S_{t-1}, 1 - e^{-\beta \frac{I_{t-1}}{N}}\right) \\
\Delta_{t,2} &\sim \text{Binom}\left(I_{t-1}, 1 - e^{-\gamma}\right)
\end{aligned}$$

with initial states

$$\begin{aligned}
S_0 &= N - I_0 \\
I_0 &\sim \text{Binom}(N, p_0) \\
R_0 &= 0
\end{aligned}$$

where p_0 is a fixed proportion. As in Lekone and Finkenstädt (2006), we note that the Binomial compartment transitions of the Chain Binomial model can be thought of as the sum of independent Bernoulli random variables as follows: each of the S_t people in the S compartment at time t will either stay in S at time $t+1$, or transition to I. Suppose that the length of time each individual in S_t remains in S_t is independently exponentially distributed with rate parameter $\beta\frac{I_t}{N}$. Then we see that the probability an individual in S_t will remain in S at time $t+1$ is equal to $exp\left\{-\beta\frac{I_t}{N}\right\}$, and thus that the probability that an individual will transition from S_t to I_{t+1} is equal to $1-exp\left\{-\beta\frac{I_t}{N}\right\}$. Thus the Binomial distribution for $\delta_{t+1,1}$ arises from the sum of the independent Bernoulli RV's for each individual in S_t.

The choice of $\beta\frac{I_t}{N}$ as the rate parameter for the exponential distribution corresponds to a frequency-dependent transmission model, which assumes that individuals in our population mix entirely at random, and thus the proportion of infectious individuals a susceptible individual will come into contact with at time t is $\frac{I_t}{N}$ (McCallum et al., 2001). The β parameter can be interpreted as the average number of contacts sufficient to transmit infection made by an individual in S per unit time t (Chowell et al., 2009). As such, the rate parameter $\beta\frac{I_t}{N}$ is equal to the rate at which individuals in S come into transmission-sufficient contact with individuals in I.

The transition from I_t to R_{t+1} is likewise modeled as the sum of Bernoulli RV's. However, we note that the rate parameter for the exponential distribution governing the amount of time an individual remains in I is no longer frequency dependent, but rather depends only on the γ parameter. This seems intuitively reasonable, as the speed with which an individual recovers from influenza should be independent of the number of people who have influenza.

This model of compartment transitions assumes that the number of individuals in S_t, I_t, and R_t does not change dramatically over a single time interval $(t, t+1)$. If this assumption were violated, the rate parameter $\beta\frac{I_t}{N}$ would no longer be a valid approximation to the transmission-sufficient contact rate from time t to time $t+1$, as $\frac{I_t}{N}$ would no longer well approximate the infectious proportion of the population for the time interval. Since our data are only available at weekly intervals, we think there would be little gained from updating transmission parameters at a finer time resolution. However, we acknowledge that if finer-grain data were available,

forecast accuracy could benefit.

The Chain Binomial SIR model presented above also assumes that the population has no individuals entering or exiting over the course of the epidemic. This is obviously an incorrect assumption for modeling large populations, such as the entire U.S., as people are being born and dying every minute. However, given that the course of seasonal influenza infections usually lasts 4–6 months, over which time we do not expect drastic fluctuations in the population of interest, we regard the assumption as unlikely to have a meaningful influence on forecasting or parameter inference. The Chain Binomial model also assumes that the β and γ parameters are constant for all time points t and all individuals in the population. More complex models can further decompose a population by location (Jewell et al., 2009), age (Hethcote, 2000), and other factors.

We propose three data models for relating our ILINet data and GFT data to latent epidemic severity. As with the DLM models, define ILI_t and GFT_t as the logit of ILINet data and GFT data at time t, respectively. The first model is the "SIR ILINet Only Model", specified by

$$ILI_t = \text{logit}\left(\frac{I_t}{N}\right) + \epsilon_{t,ILI} \qquad\qquad \epsilon_{t,ILI} \sim N(0, \sigma^2_{ILI}) \tag{3.9}$$

where I_t is the latent number of people in the I compartment at time t, and N is our total population size. The "SIR ILINet Only Model" models the logit of ILINet data as being a noisy, unbiased estimate of the logit of the proportion of the population that is currently infectious. A correspondence can readily be seen between this data model and the "DLM ILINet Only Model".

We next specify the "SIR ILINet and GFT Biased Model". The "SIR ILINet and GFT Biased Model" uses equation 3.9 as the observation equation for ILINet data, and further models GFT data as as as

$$GFT_t = \text{logit}\left(\frac{I_t}{N}\right) + \rho\left(GFT_{t-1} - \text{logit}\left(\frac{I_{t-1}}{N}\right)\right) + \epsilon_{t,GFT} \tag{3.10}$$

where $\epsilon_{t,GFT} \sim N(0, \sigma^2_{GFT})$. The "SIR ILINet and GFT Biased Model" treats GFT data as a biased source of information on latent severity. Notably, the bias in week w is set equal to the observed difference between GFT data and logit latent severity in the previous week,

multiplied by a shrinkage parameter ρ. The third model, labeled the "SIR ILINet and GFT Unbiased Model", is specified by setting $\rho = 0$ in equation 3.10.

We note that the bias specification in Equation 3.10 contrasts with the bias specification in the "DLM ILINet and GFT Biased Model", which had the bias evolving via a local linear trend process. Although we tried to fit a stochastic bias term β to the SIR model to more closely mirror the "DLM ILINet and GFT Biased Model", doing so resulted in unidentifiablity between the σ_G and σ_β parameters. Namely, the model was unable to correctly partition the total variance between σ_G and σ_β, instead providing very large estimates for σ_G and very small estimates for σ_β. In turn, the large σ_G estimates resulted in forecasts that were based almost entirely on the lagged ILINet data, which tended to do a poor job. As such, in the "SIR ILINet and GFT Biased Model" we treat the bias as a deterministic function of ρ, GFT_{t-1}, and $logit\left(\frac{I_{t-1}}{N}\right)$ to reduce the number of variance parameters.

3.4 DLM Simulation Studies

In Section 3.4.1, data are simulated from the "DLM Combined Model with Bias" presented in Section 3.3.1. Predictions using this data are evaluated, first assuming known values of generating parameters in Section 3.4.2, and then in a fully Bayesian setting with no known parameters in Section 3.4.3. In both sections, it is found that the inclusion of biased data has the potential to greatly increase predictive ability for simulated data.

3.4.1 Simulating Data from a DLM

Data is simulated as coming from the DLM "Combined Model with Bias". We simulate N sets of data as coming from the "Combined Model with Bias" for all combinations of $(\sigma_\beta, \sigma_I, \sigma_G) \in (.1, 1, 10)^3$. σ_s was fixed at a value of 1. Data sets are simulated as follows: for each combination of values chosen for σ_I and σ_G, I data sets consisting of $s_{1:T}$, $\beta_{1:T}$, $ILI_{1:T}$, and $GFT_{1:T}$ are simulated using $\mu_S = 0$, $\mu_\beta = 0$, and $\sigma_\beta = 0.1$. The $\beta_{1:T}$ and $GFT_{1:T}$ observations from these data sets are then rescaled to produce two new sets of I observations, one with $\sigma_\beta = 1$ and one with $\sigma_\beta = 10$. Thus, for given σ_I and σ_G generative values, data sets with

different σ_β values are directly comparable to each other. Values of $T = 20$ and $I = 100$ were chosen.

Using different combinations of values for the σ_I, σ_G, and σ_β parameters allows the prediction performance of the models to be evaluated under a variety of conditions that could correspond to those encountered when using real data. For example, highly accurate ILINet and GFT data could be available, corresponding to low vales for σ_I and σ_G. However, the bias term β may be highly variable from week to week, giving a relatively large σ_β term.

3.4.2 Simulation Study for DLM Models with Known Parameters

Define $\theta^{(j)} = \left(\sigma_I^{(j)}, \sigma_G^{(j)}, \sigma_\beta^{(j)}\right)$ as the j'th combination of parameter values for $(\sigma_I, \sigma_G, \sigma_\beta)$ where $j = 1, \ldots, 27$. Define $s^{(i)}_{\theta^{(j)},t}$ as the latent severity value at time t from the i'th set of data generated using the parameter values $\theta^{(j)}$ for $i = 1, \ldots, I$, $j = 1, \ldots, 27$, and $t = 1, \ldots, T$.

To compare the different models, we evaluate their ability to accurately predict the latent state $s^{(i)}_{t,\theta^{(j)}}$. Specifically, interest lies in the one week ahead predictive distributions $f_M(s^{(i)}_{\theta^{(j)},t+1} | y^{(i)}_{\theta^{(j)},1:t}, \theta^{(j)})$ for different data sets i and parameter sets j, where M denotes the model used to conduct predictions (chosen from the "Combined Model with Bias" and the "ILINet Only Model"), and where $y^{(i)}_{\theta^{(j)},t}$ is the observed data values at time t from the i'th set of data generated using the j'th set of parameter values. Conditioned upon the known value of θ (the parameters used to generate each data set) these predictive distributions can be obtained recursively using the Kalman filter (Kalman, 1960).

We note that the "ILINet and GFT Unbiased Model" was not included for analysis in this section. This choice was made because using the "ILINet and GFT Unbiased Model" to produce predictions for biased data requires specification of the σ_G parameter. If the value of σ_G used to generate the data was also used for predictions with the "ILINet and GFT Unbiased Model", the forecast variance would be greatly understated, as the "ILINet and GFT Unbiased Model" does not account for the bias variance σ_β. Rather than arbitrarily inflating the σ_G value provided to the "ILINet and GFT Unbiased Model", we excluded it from comparison in this section.

With respect to the models specified in Section 3.3, a value of $\delta = 2$ was chosen since ILINet data are commonly reported at a lag of 2 weeks. Thus, for each data set i simu-

lated from each combination of parameters j, each of the two competing models M are provided simulated ILINet data $ILI^{(i)}_{\theta^{(j)},1:t-2}$ for $t \in t_0, \ldots, T$, where t_0 is the earliest point for which we are interested in predictive density heights. The "ILINet and GFT Biased Model" is additionally provided $GFT^{(i)}_{\theta^{(j)},1:t}$, so that $y^{(i)}_{\theta^{(j)},1:t} = \left(ILI^{(i)}_{\theta^{(j)},1:t-2}\right)$ for the "ILINet Only Model", and $y^{(i)}_{\theta^{(j)},1:t} = \left(ILI^{(i)}_{\theta^{(j)},1:t-2}, GFT^{(i)}_{\theta^{(j)},1:t}\right)$ for the "ILINet and GFT Biased Model". Using this data, predictive distributions for the latent severity s_{t+1} are calculated for model M as $f_M(s_{t+1}|y^{(i)}_{\theta^{(j)},1:t}, \theta^{(j)})$. For each predictive curve, label the height of the predictive curve evaluated at the known simulated latent severity value as $H^{(i)}_{M,\theta^{(j)},t+1}$, so that

$$H^{(i)}_{M,\theta^{(j)},t+1} = f_M(s^{(i)}_{\theta^{(j)},t+1}|y^{(i)}_{\theta^{(j)},1:t}, \theta^{(j)})$$

. Values of $H^{(i)}_{M,\theta^{(j)},t+1}$ are calculated for $i = 1, \ldots, I$, providing I predictive heights for each time point $t = t_0, \ldots, T$ for each combination of parameters $\theta^{(j)}$.

We are interested primarily not in the absolute predictive heights, but rather in comparing different models to each other. Label the two models under consideration as M_1 and M_2. Then $H^{(i)}_{M_1,\theta^{(j)},t+1}$ and $H^{(i)}_{M_2,\theta^{(j)},t+1}$ constitute paired data, as they are both height measurements at the same time point, from the same data set. We calculate the difference between the heights of the two models as $D^{(i)}_{\theta^{(j)},t+1} = H^{(i)}_{M_1,\theta^{(j)},t+1} - H^{(i)}_{M_2,\theta^{(j)},t+1}$, and then average over time for given values of i and j to obtain

$$D^{(i)}_{\theta^{(j)}} = \frac{1}{T - t_0 + 1} \sum_{t=t_0}^{T} D^{(i)}_{\theta^{(j)},t+1}$$

For a fixed parameter set j, the mean and standard error of $D^{(i)}_{\theta^{(j)}}$ can calculated as

$$D_{\theta^{(j)}} = \frac{1}{I} \sum_{i=1}^{I} D^{(i)}_{\theta^{(j)}} \tag{3.11}$$

$$SE(D_{\theta^{(j)}}) = \frac{1}{\sqrt{I}} \sqrt{\frac{\sum_{i=1}^{I} \left(D^{(i)}_{\theta^{(j)}} - D_{\theta^{(j)}}\right)^2}{I - 1}} \tag{3.12}$$

The above procedure was conducted using values of $I = 100$, $t_0 = 5$, and $T = 20$.

Figure 3.2 displays $D_{\theta^{(j)}} \pm 1.96 * SE(D_{\theta^{(j)}})$ faceted by the $\sigma^{(j)}_I$ and $\sigma^{(j)}_G$ parameter values used to generate the data. Within each facet, mean predictive density heights and bars extending to 1.96 standard errors above and below the mean are displayed for each model for different values of $\sigma^{(j)}_\beta$.

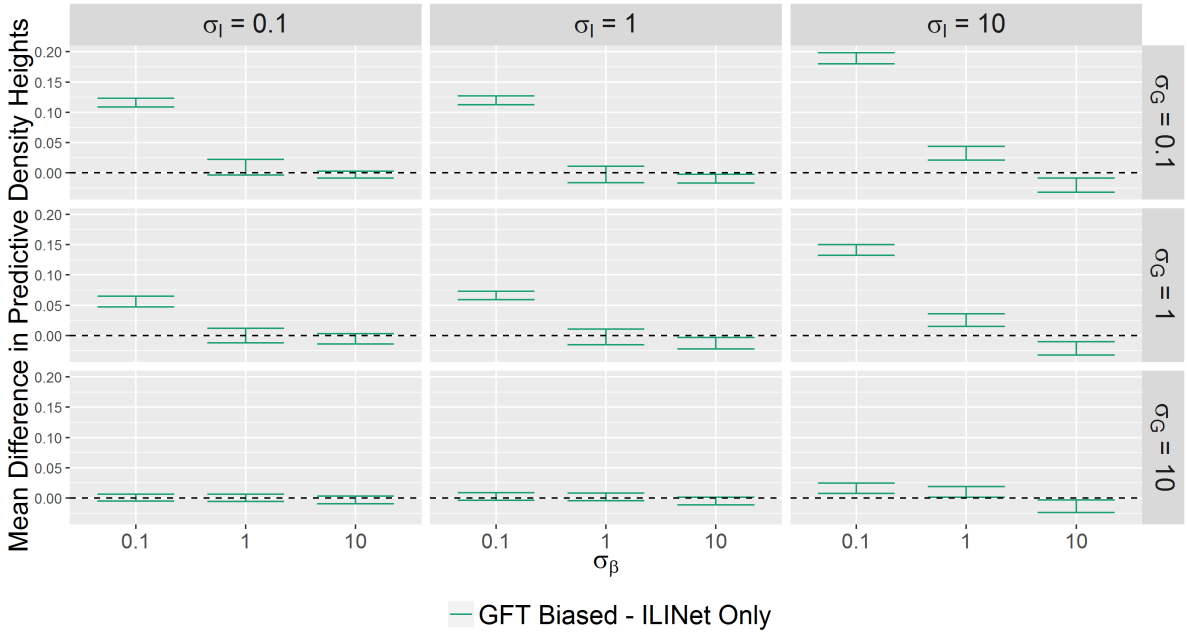

Figure 3.2: Mean difference in one time-point ahead predictive density heights between the "ILINet and GFT Biased Model" and the "ILINet Only Model" for different combinations of generating parameters σ_I, σ_G, and σ_β. Means are taken over all simulated data sets and time points for each combination of parameters. Bars extend to 1.96 standard errors above and below the mean difference. The horizontal dotted line at 0 represents no difference in average heights between the two models.

Unsurprisingly, the mean differences between the "ILINet and GFT Biased Model" and the "ILINet Only Model" seen in Figure 3.2 are highest for low values of σ_β. At σ_β values of 0.1, the average difference is far above 0 for small and moderate values of σ_G - in other words, when the bias is relatively consistent from one time point to the next, the Bias model predictions benefit greatly from incorporating the biased data. The difference in average heights is most apparent at larger values of σ_I and smaller values of σ_G, seen in the upper right facets of the plot. In these facets, the GFT data stream provided to the "ILINet and GFT Biased Model" is not only more timely than the ILINet data stream, it's also more accurate.

We also note that as σ_β increases, the difference between the two models decreases. Intuitively, as the bias in the GFT data becomes more variable, the predictions made using the "ILINet and GFT Biased Model" will depend more on the lagged, but unbiased, ILINet data. Similarly, for fixed values of σ_I and σ_β, increasing σ_G again causes the difference in heights

between the two models to disappear.

To further examine the predictive performance of our two models, we look at the variance of the predictive distributions. Predictive distributions for DLMs are Gaussian, and thus defined by their mean and variance. The Kalman filter provides both the mean and variance of the predictive distributions, allowing a comparison of predictive distribution variances between models and parameter values. Figure 3.3 displays the average variance of the one week predictive densities. Notably, the "ILINet and GFT Biased Model" predictive variance is never higher than the "ILINet Only Model". The difference in variances is most evident for large values of σ_I and small values of σ_β and σ_G. For these parameter combinations, the "ILINet Only Model" predictions are based on highly variable, lagged ILINet data, in turn causing the predictive density variance to be large. The "ILINet and GFT Biased Model", on the other hand, has access to accurate and up-to-date GFT data, leading to a low predictive variance. Again, we also note that as σ_β and σ_G increase, the predictive variance of the "ILINet and GFT Biased Model" converges to that of the "ILINet Only Model".

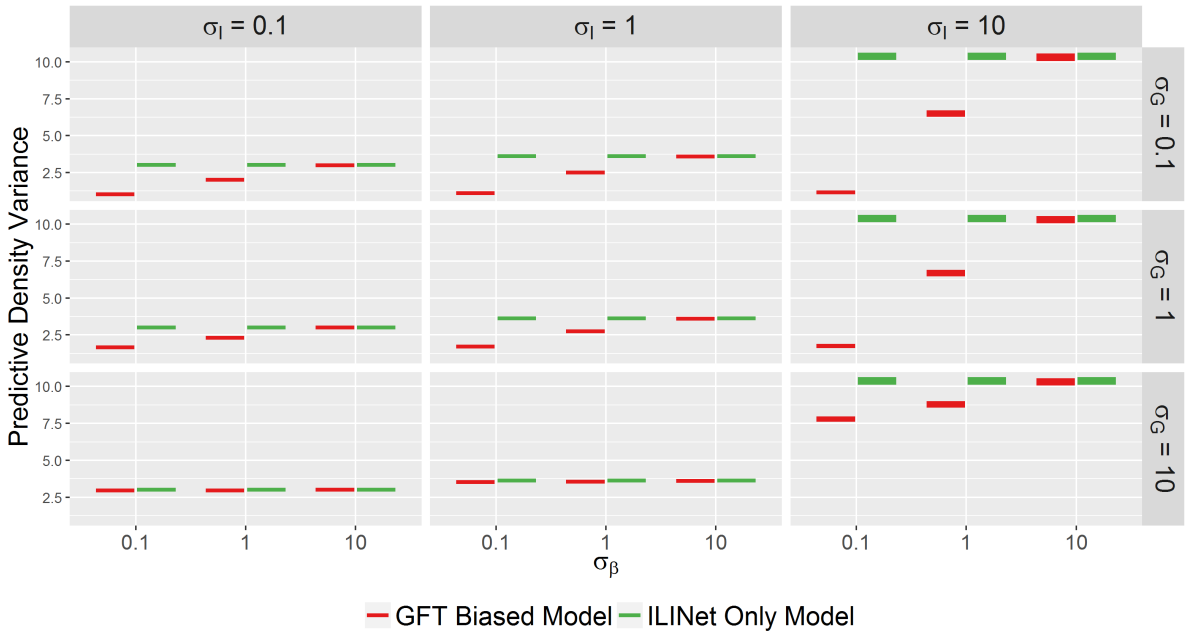

Figure 3.3: Average variance of one week ahead predictive distributions for the "ILINet and GFT Biased Model" (green lines) and the "ILINet Only Model" (red lines). Averages are taken over all simulated data sets and time points for each combination of parameters. Bars extend to 1.96 standard errors above and below the mean variance.

Figure 3.4 plots predictive distributions produced by the two models for the first set of simulated data for each combination of σ_I and σ_G. Note that σ_β is fixed at 0.1 in all facets. The predictive densities produced by the "ILINet Only Model" are, in general, centered near the latent state of interest, as are those produced by the "ILINet and GFT Biased Model" – it can be shown that predictive distributions generated by these two models are unbiased. The major difference between the distributions produced by the two models instead arises from the variance of the predictive distributions. For smaller values of σ_G, the "ILINet and GFT Biased Model" produces consistently narrower predictive distributions centered near the true latent state. For large values of σ_G, the predictive distributions are indistinguishable between the two models.

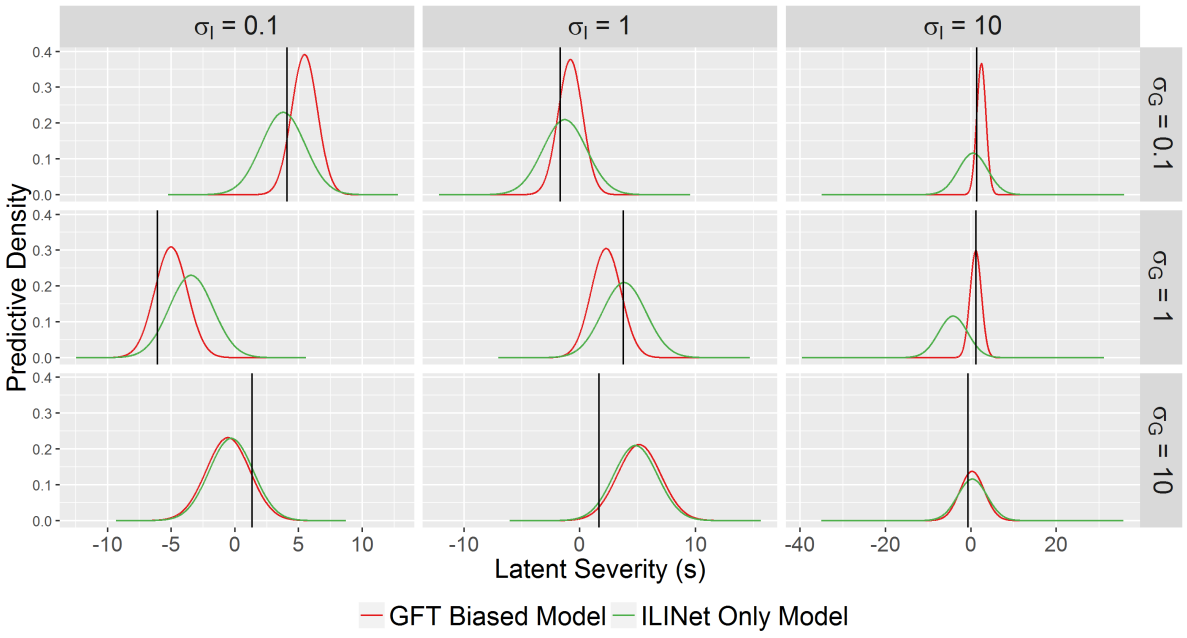

Figure 3.4: One week ahead Predictive density curves for the latent state s for both the "ILINet and GFT Biased Model" and the "ILINet Only Model", along with the true value of s (vertical line). Prediction was for week 15 of the first simulated data set for each parameter combination, with $\sigma_\beta = 0.1$

Figure 3.5 shows one week ahead predictive intervals for our competing models. The patterns seen in Figure 3.5 are the same as those from Figure 3.4. The "ILINet and GFT Biased Model" produces noticeably narrower forecasts for parameter values in the upper-right facets

of the figure. For large σ_G values, the forecasts produced match almost exactly.

Figure 3.5: Latent states s for times 5 – 20 (purple square). For each time t, we provide a 95% prediction interval for the latent state at time t produced by the "ILINet and GFT Biased Model" and the "ILINet Only Model". Predictions were conducted for the first simulated data set for each parameter combination, with $\sigma_\beta = 0.1$.

3.4.3 DLM Simulation Study With Unknown Parameters

In Section 3.4.3.1, we describe the priors placed on the parameters in our DLM models, and the MCMC algorithm used to conduct inference on these parameters. In Section 3.4.3.2, we analyze the predictive performance of all three DLM models and compare the results to those of Section 3.4.2.

3.4.3.1 Priors and Inference Methods

To provide a study which more closely resembles a real-world analysis, the data simulated in Section 3.4.1 were also used in a fully Bayesian simulation study, in which no parameters were assumed to be known. To conduct the Bayesian simulation study, priors were placed on the unknown parameters. Prior distributions for all standard deviation parameters

$(\sigma_I, \sigma_G, \sigma_s, \sigma_\gamma, \sigma_\beta, \sigma_\lambda)$ were set to

$$\sigma \sim t_m^+(M, S^2)$$

where $t_m^+(M, S^2)$ represents the shifted, scaled half-t distribution with center parameter M and scale parameter S^2. Values of $M = 0$, $S^2 = .5$, and $m = 10$ were used for all half-t priors.

Prior distributions for mean parameters $(\mu_s, \mu_\gamma, \mu_\beta, \mu_\lambda)$. were specified as

$$\mu \sim t_m(M, S)$$

M was set to $M = -4$ for μ_s and to $M = 0$ for all other μ parameters. Values of $S = 1$ and $m = 10$ were used for all μ parameters.

Recall from Section 3.4.1 that data sets were simulated from the "ILINet and GFT Biased Model" for all combinations of $(\sigma_\beta, \sigma_I, \sigma_G) \in (.1, 1, 10)^3$. Define $\theta^{(j)} = \left(\sigma_I^{(j)}, \sigma_G^{(j)}, \sigma_\beta^{(j)}\right)$ as the j'th set of parameter values for $(\sigma_I, \sigma_G, \sigma_\beta)$ where $j = 1, \ldots, 27$. For each combination of parameters $\theta^{(j)}$, the first 16 data sets simulated from that combination of parameters were used for prediction. For each of the 16 data sets from each parameter combination j, predictions were conducted for times $t = t_0, \ldots, T$ using each of the three competing data models. Following Section 3.4.1, if interest lies in prediction for time point $t + 1$, all three models ("ILINet and GFT Biased Model", "ILINet and GFT Unbiased Model", "ILINet Only Model") were provided simulated ILINet data from time 1 to time $t - 2$. The two GFT models were additionally provided GFT data up tom time t.

We ran separate MCMC algorithms for each of the three models, using a Metropolis-Hastings sampler with a block multivariate normal proposal distribution to obtain samples from the posterior distributions of the model parameters. The likelihood for each Metropolis-Hastings step was calculated using the Kalman filter. The MCMC algorithm was run for an initial set of 12,000 iterations, with the first 6,000 discarded as a burn-in period. A thinning interval of 3 was applied to the remaining 6,000 samples, leaving a total of 2,000 posterior samples. A Geweke diagnostic was applied to the posterior samples, using a Bonferroni correction for multiple testing. If a lack of convergence was indicated by the diagnostic, additional runs were conducted until convergence was achieved.

Using the posterior samples of the parameters, forecasting was conducted via a Kalman filter. Specifically, label the k'th posterior sample of all top level σ and μ parameters from a given MCMC run as $\tilde{\theta}^{(k)}$ for $k = 1, \ldots, K$, where $\tilde{\theta}$ is a sampled vector of parameters. For each posterior sample k, the Kalman filter was used to obtain the one time step ahead predictive distribution for the latent state s, defined as $f_M(s^{(i)}_{\theta^{(j)},t+1}|y^{(i)}_{\theta^{(j)},1:t}, \tilde{\theta}^{(k)})$.

3.4.3.2 Simulated Data Analysis

From the one time step ahead predictive distributions, predictive densities can be evaluated in a manner similar to Section 3.4.2. We average over posterior samples to calculate the height for model M and generating parameter set j as

$$\begin{aligned} H^{(i)}_{M,\theta^{(j)},t+1} &= \int_{\tilde{\theta}} f_M(s^{(i)}_{\theta^{(j)},t+1}|y^{(i)}_{\theta^{(j)},1:t}, \tilde{\theta}) dt d\tilde{\theta} \\ &\approx \frac{1}{K} \sum_{k=1}^{K} f_M(s^{(i)}_{\theta^{(j)},t+1}|y^{(i)}_{\theta^{(j)},1:t}, \tilde{\theta}^{(k)} \end{aligned}$$

where K is the number of MCMC samples for parameter set j, simulated data set i, and model M. From these heights, we calculate height differences between two models as $D^{(i)}_{\theta^{(j)},t+1} = H^{(i)}_{M_1,\theta^{(j)},t+1} - H^{(i)}_{M_2,\theta^{(j)},t+1}$. We then average differences over times and data sets to produce mean hight differences and standard errors of height differences as in Equations 3.11 and 3.12. Similar to Figure 3.2. Figure 3.6 displays mean differences between all sets of models for different parameter combinations j, with bars extending to 1.96 standard errors above and below the mean.

As compared to the mean differences seen in Section 3.4.2, the widths of the standard error bars have increased. This is partly because the standard errors for each combination of parameters and models are now calculated using only 128 samples (16 data sets, each with 8 weeks), and partly because our predictions now incorporate the uncertainty in the model parameters. As such, many of the intervals for the mean difference produced contain 0. Nonetheless, some interesting conclusions can be drawn from the simulation study results.

Looking at the differences between "ILINet and GFT Biased Model" and the "ILINet Only Model", we notice a similar trend to that seen in 3.2. For lower values of σ_G and σ_β, the "ILINet and GFT Biased Model" still greatly outperforms the "ILINet Only Model". As σ_G

or σ_β increase, the average predictive heights converge. Additionally, the "ILINet and GFT Biased Model" never produces significantly lower average heights than the "ILINet Only Model". Thus a modeler should feel comfortable that if possibly biased data are included, and the bias is modeled explicitly, the forecasts obtained will never be worse on average than if only the unbiased data had been used.

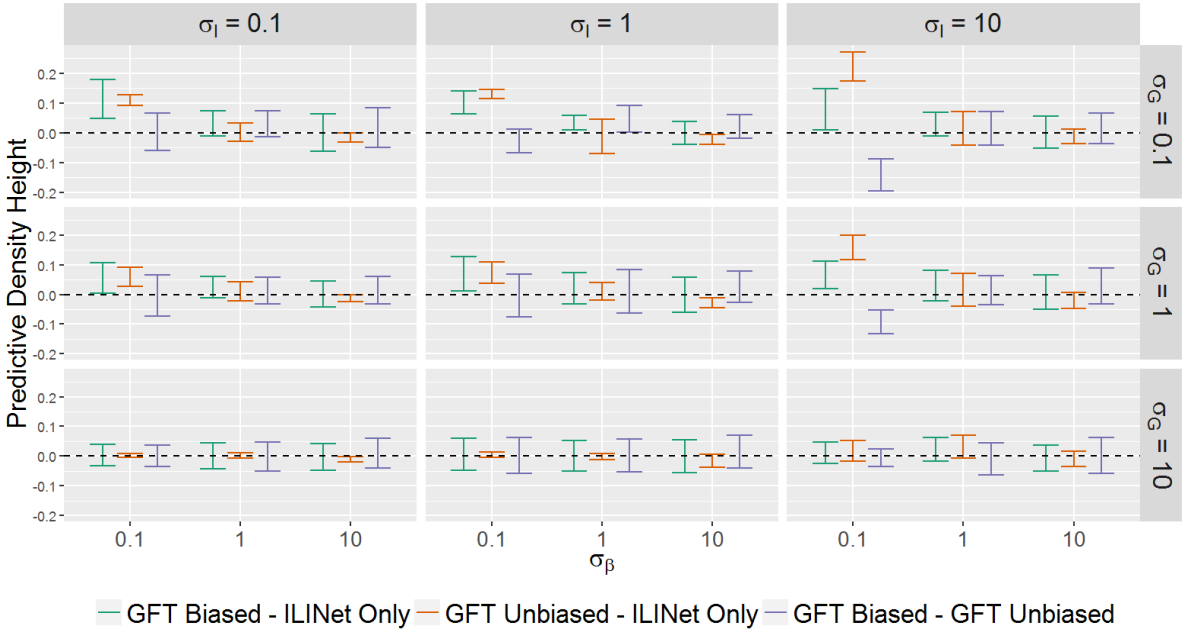

Figure 3.6: Average difference in one week ahead predictive density height for all pairs of models uder different generating parameter combinations. For each combination of parameters, averages are taken over simulated data sets and weeks. Bars extend to 1.96 standard errors above and below the mean differences. The horizontal dashed line at 0 denotes no difference in predictive density heights between a pair of models.

Looking at the mean height difference between the "ILINet and GFT Unbiased Model" and the "ILINet Only Model" in Figure 3.6, we see that the "ILINet and GFT Unbiased Model" produces higher average heights for low values of σ_β, and low to moderate values of σ_G, than the "ILINet Only Model". Note that the bias term β was initialized as coming from a $N(0, \sigma_\beta^2)$ distribution. Thus, when $\sigma_\beta = 0.1$, the bias term is starting at a very small value, and is evolving very little at each time point, while the latent state of interest changes much more rapidly. For smaller values of σ_β, modeling the GFT data as an unbiased data source produces much improved predictions over using ILINet data only.

On the other hand, for moderate and large values of σ_β, the GFT data stream becomes too variable to impart much information to the predictive distributions, so predictions for these parameter values rely almost entirely on the lagged ILINet data. In fact, for very high σ_β values, the "ILINet and GFT Unbiased Model" can perform noticeably worse than the "ILINet Only Model" for small and moderate σ_G values. This should encourage researchers to use caution when treating possibly biased data as unbiased for the purposes of forecasting.

Deciding which biased data model to use is somewhat less clear cut. Looking at the average height differences between the "ILINet and GFT Biased Model" to the "ILINet and GFT Unbiased Model" reveals that the Unbiased model tends to perform best when $\sigma_\beta = 0.1$. Specifically, when $\sigma_\beta = 0.1$ and $\sigma_I = 10$, the average predictive height for the biased model falls noticeably below that of the unbiased model. On the other hand, the biased model tends to give higher average heights when $\sigma_\beta = 1$ and $\sigma_\beta = 10$. The reason for this is that for relatively small values of σ_β, modeling the bias explicitly does not provide additional predictive information, as the bias variance is swamped by the variance of the latent state of interest s. For higher values of σ_β, modeling the bias has a more pronounced beneficial effect on predictive accuracy.

To summarize, we found that using biased data via the "ILINet and GFT Biased Model" will never lead to worse predictive performance than using lagged, unbiased data, and will often lead to better performance. Treating the biased data as unbiased may actually produce superior predictions, out of the models considered, if the variance in the bias term is very low. On the other hand, if the variance in the bias term is comparable to or higher than the latent state variance, treating the biased data as unbiased risks producing worse forecasts than would be obtained if only the lagged, unbiased data were used. In such cases, explicitly modeling the bias term will provide the highest average predictive height.

3.5 Simulation Study for SIR Models

In Section 3.5.1 we describe our method of simulating data from an SIR model. In Section 3.5.2, we detail the Particle MCMC method used to conduct inference on parameters in our SIR models, and conduct a fully Bayesian analysis of the predictive abilities of the SIR models using the simulated data.

3.5.1 Simulating Data from an SIR Model

Data was simulated as coming from the "SIR ILINet and GFT Biased Model" of Section 3.3.2. We started by generating 16 sets of $S_{1:T}$, $I_{1:T}$ and $R_{1:T}$ using fixed values of $\beta = 0.4$, $\gamma = 0.32$, and $p_0 = 0.02$. For each of these 16 generated data sets, we then generated a single set of ILINet and GFT observations for all combinations of $\sigma_I, \sigma_G \in (0.1, 0.3)$ and $\rho \in (0.2, 0.8)$. Thus for each combination of σ_I, σ_G, and ρ, we have 16 sets of observations generated from the same 16 sets of latent S, I, and R states.

In addition to varying the σ_I and σ_G values used to generate the data, using different ρ values for our simulated data sets will allow us to determine the benefit of including biased GFT data for a range of different bias magnitudes – a ρ of 0.2 corresponds to a low level of bias being carried over from week to week, while $\rho = .8$ corresponds to most of the bias carrying over from week to week.

3.5.2 Priors and Inference Methods

Inference was conducted using a PMCMC algorithm (Andrieu et al., 2010) which jointly sampled from all unknown parameters at each iteration. To obtain estimates of the data likelihood, as well as samples from the latent S, I, R, and Δ states, a bootstrap filter was used, and is provided in Appendix Section A.2. The following priors were placed on the parameters of the SIR models:

$$\begin{aligned} \beta \sim \text{Gamma}(K, O) \quad & \lambda \sim \text{Gamma}(K, O) \\ \sigma_I \sim t_m^+(M_\sigma, S_\sigma) \quad & \sigma_G \sim t_m^+(M_\sigma, S_\sigma) \\ \rho \sim \text{N}(M_\rho, S_\rho^2) \quad & p_0 \sim \text{Beta}(A, B) \end{aligned}$$

where K and O denote the shape and rate parameters of a gamma distribution, A and B denote the two shape parameters of the beta distribution, M_σ and S_σ denote the center and scale parameter of a half-t distribution with m degrees of freedom, and M_ρ and S_ρ^2 denote the mean and variance of a normal distribution. Values of $K = 5$, $O = 10$, $m = 10$, $M_\sigma = 0$, $S_\sigma = .5$, $M_\rho = 1$, $S_\rho = .5$, $A = 1$, and $B = 20$ were used.

The PMCMC algorithm was run for an initial set of 12,000 iterations, with a thinning interval of 3. At each iteration, 3000 particles were used to estimate the data likelihood. Additionally, an adaptive procedure was applied which recalculated the proposal covariance matrix every 2,000 iterations to achieve a more optimal acceptance rate. Convergence was diagnosed using a Geweke diagnostic. If a departure from convergence was found for any parameters, additional sets of 3,000 iterations were run until convergence was achieved.

3.5.3 Simulation Study Results

For each set of data generated from each combination of parameter, one week ahead forecasts were conducted for times $t = 11, \ldots, 14$. Table 3.1 displays one week ahead predictive interval coverage proportions and average interval length for all combinations of models and generating parameters. Predictive interval coverage proportions are defined as the proportion of times a one week ahead 50% predictive interval produced by a model covers the true latent state for the next week.

			$\sigma_I = 0.1$		$\sigma_I = 0.3$	
	GFT Data?	Bias?	$\rho = 0.2$	$\rho = 0.8$	$\rho = 0.2$	$\rho = 0.8$
$\sigma_G = 0.1$	Yes	Yes	0.87 (0.025)	0.82 (0.024)	0.87 (0.026)	0.88 (0.027)
	Yes	No	0.92 (0.019)	0.73 (0.020)	0.91 (0.019)	0.72 (0.019)
	No	-	0.92 (0.033)	0.92 (0.033)	0.84 (0.038)	0.84 (0.038)
$\sigma_G = 0.3$	Yes	Yes	0.85 (0.025)	0.73 (0.029)	0.82 (0.027)	0.72 (0.032)
	Yes	No	0.75 (0.022)	0.39 (0.026)	0.69 (0.023)	0.36 (0.025)
	No	-	0.92 (0.033)	0.92 (0.033)	0.84 (0.038)	0.84 (0.038)

Table 3.1: 50% one week ahead predictive interval coverage proportions (average interval length provided in parentheses) for each of the three SIR models and each combination of generating parameters. For a given set of generating parameters, coverage proportions and average interval lengths are calculated over all simulated data sets and time points.

We first focus on comparing the "SIR ILINet and GFT Biased Model", seen in rows 1 and 4, to the "SIR ILINet Only Model", seen in rows 3 and 5. Surprisingly, the model using only ILINet data produces intervals with the highest coverage proportions for almost all combinations of generating parameters. However, it also produces very wide intervals, especially when $\sigma_I = 0.3$. The biased model, on the other hand, produces narrower intervals with slightly decreased coverage proportions.

Comparing the "SIR ILINet and GFT Unbiased Model" to the "ILINet Only Model", we notice that the unbiased model tends to produce the smallest average interval lengths. However, smaller lengths come at the expense of adequate coverage proportion. Especially for $\sigma_G = 0.3$ and $\rho = 0.8$, the intervals produced by the unbiased model display under-coverage, which is not a desirable trait in forecast intervals.

When comparing the two models which use GFT data to each other, we note a similar trend to that seen in Section 3.4.3. Namely, for the smallest values of σ_G and ρ, the unbiased model produces narrower intervals witch cover more of the true latent states of interest. We again see that if the biased data source is, in a sense, "only slightly biased", treating it as an unbiased source can lead to superior forecasts. However, as soon as either ρ or σ_G increase, we see that the forecast coverage proportions dip markedly.

3.6 Forecasting With Real ILINet and GFT Data

In Section 3.6.1, the DLM models described in Section 3.3 are used to forecast influenza severity for the 9 seasons of ILINet and GFT data we have available. We focus primarily on comparing the forecasts produced by the two models which include GFT to those produced by the "ILINet Only Model", although comment on the relative performance of the two GFT models as well.

In Section 3.6.2, we compare forecasts produced by the two SIR models which include GFT data to those from the "SIR ILINet Only Model". Forecasts produced by the SIR models are also compared to those produced by the DLM models of Section 3.6.1.

3.6.1 DLM Forecasts

The DLMs described in Section 3.3.1 were fit to the real data detailed in Section 3.2. Inference and forecasting were conducted using the MCMC scheme described in 3.4.3. Nine seasons of data are available for both the ILINet and GFT data streams.

For each model, we obtained samples from the posterior distributions of θ, where θ is defined as the vector of all top level σ and μ parameters. For each posterior sample $\theta^{\tilde{(k)}}$, the Kalman filter was used to obtain the one week ahead forecasting distribution, defined as

$f(y_{s,w+1}|y_{s,1:w}, \tilde{\theta^{(k)}})$ for week w and season s. Note that in using real ILINet and GFT data, as opposed to simulated data, we no longer know the true value of the latent epidemic severity state $s_{s,w+1}$ for season s and week $w+1$. Thus, forecasts are no longer evaluated by their ability to predict the latent severity in week $w+1$, but rather by their ability to predict the observed value of ILINet data in week $w+1$.

Posterior distributions for the DLM model parameters can be found in Appendix Section A.3. Using these posterior distributions, we can attempt to discern how effective the inclusion of GFT data will be by referencing the posterior distributions to the average forecast heights seen in Figure 3.6. Posterior distributions obtained from the "ILINet and GFT Biased Model" for the 9 seasons of ILINet and GFT data show that median posterior values for σ_z were in general about twice as large as σ_I and σ_β values, and an order of magnitude larger than σ_G values. Based on these relationships between our parameters, placing us roughly in the middle of the upper middle facet of Figure 3.6, we can roughly expect the "ILINet and GFT Biased Model" to outperform both the "ILINet and GFT Unbiased Model" and the "ILINet Only Model". We further expect there to be little difference between the "ILINet and GFT Unbiased Model" and the "ILINet Only Model".

Following Section 3.4.3.2, we investigate forecasting performance by looking at the difference in average one week ahead forecast heights between pairs of models. Of primary interest in influenza seasons is forecasting from the point when epidemic severity starts to increase rapidly until severity reaches its peak. For this reason, we only consider forecasts for weeks $w = 13, \ldots, 33$, where $w = 0$ corresponds to MMWR week 32 of a given influenza season. This time frame captures the peak of the seasonal epidemics for all seasons we have data for.

Calculation of the standard error for the difference in forecast heights differs somewhat here as compared to Equation 3.12. We first calculate forecast heights for week w in season s as $f_M(y_{s,w+1}|y_{s,1:w}) \approx \frac{1}{K}\sum_{k=1}^{K} f_M(y_{s,w+1}|y_{s,1:w}, \tilde{\theta^{(k)}})$. Differences between two models of interest are then calculated as $D_{s,w+1} = f_{M_1}(y_{s,w+1}|y_{s,1:w}) - f_{M_2}(y_{s,w+1}|y_{s,1:w})$. From these weekly differences, the average difference for season s is then $D_s = \frac{1}{W-w_0+1}\sum_{w_0}^{W} D_{s,w+1}$.

Standard errors taken across a single season are calculated using a block bootstrap method (Efron and Tibshirani, 1986) with 1000 repetitions and a block size of 5. Bootstrapping was

conducted using the `boot` R package.

	Season				
Difference	1	2	3	4	5
Biased - ILINet	0.36 (0.23)	0.42 (0.35)	0.25 (0.17)	0.46 (0.37)	0.23 (0.18)
Unbiased - ILINet	0.23 (0.13)	0.23 (0.13)	0.42 (0.12)	0.55 (0.17)	0.57 (0.20)
Biased - Unbiased	0.13 (0.16)	0.19 (0.24)	-0.18 (0.21)	-0.09 (0.44)	-0.35 (0.14)

Difference	6	7	8	9
Biased - ILINet	0.25 (0.48)	0.33 (0.34)	0.49 (0.37)	0.4 (0.31)
Unbiased - ILINet	-0.18 (0.14)	-0.26 (0.17)	0.44 (0.15)	0.31 (0.16)
Biased - Unbiased	0.44 (0.53)	0.59 (0.20)	0.05 (0.38)	0.09 (0.42)

Table 3.2: Mean one week ahead seasonal mean forecast height differences between the three DLM models (Standard errors for height differences are provided in parentheses). For a given season, each mean is calculated as the average forecast height from week 13 to week 33.

Table 3.2 displays seasonal mean forecast height differences for all pairs of models under comparison. Again, we note that standard errors are fairly large, in part because we are only considering 21 weeks of data for each season. We first focus on the differences between the "ILINet and GFT Biased Model" and the "ILINet Only Model". The average forecast height differences between these two models are all positive, suggesting that the "ILINet and GFT Biased Model" does a superior job of prediction in every season considered. The highest mean height differences occur in seasons 2, 4, 8, and 9. These seasons all display sudden increases in influenza epidemic severity as compared to the other seasons, making the inclusion of the more timely GFT data have a large beneficial impact on forecasting.

A similar trend is seen in the difference between the "ILINet and GFT Unbiased Model" and the "ILINet Only Model". The average difference between the "ILINet and GFT Unbiased Model" and the "ILINet Only Model" is also greatest for seasons 4, 8, and 9. The GFT data for these seasons displayed little bias, in turn leading the unbiased model to produce highly accurate forecasts. However, the unbiased model also produces negative differences for seasons 6 and 7, when GFT data was highly biased. This is reflective of Figure 3.6, where we noted that highly biased data can actually lead the "ILINet and GFT Unbiased Model" to perform worse than the "ILINet Only Model".

We also note that the standard errors of the difference in mean seasonal forecast heights between the "ILINet and GFT Unbiased Model" and the "ILINet Only Model" tend to be lower

than the standard errors for other comparisons, a phenomenon also seen in Figure 3.6. This occurs because forecast heights produced by the unbiased model tend to be less variable than those produced by the biased model. As will be seen in Table 3.3, the unbiased model gives rise to wider, less peaked forecast distributions than the biased model, which in turn produce more consistent forecast heights.

Comparing the "ILINet and GFT Biased Model" to the "ILINet and GFT Unbiased Model", we see that the biased model produces higher average forecast heights for 6 of the 9 seasons considered. Unsurprisingly, the highest average differences arise from seasons 6 and 7, the two seasons when GFT data was heavily biased. Given the results in Table 3.3, for any future forecasts we would recommend an examination of the bias currently present in the data before deciding which model to use – if the GFT data shows little bias, the unbiased model could produce superior results, while if a noticeable bias is present, modeling the bias explicitly can lead to better forecasts. Alternatively, choosing to always model the bias can be considered a safe choice, in that forecasts will not be worse than using the unbiased data alone.

To further compare forecasting ability between our models, we produce one week ahead forecast intervals. Samples $\tilde{y}^{(k)}_{M,s,w+1}$ were drawn from the forecasting distribution $f_M(y_{s,w+1}|y_{s,1:w}, \tilde{\theta}^{(k)})$ for each season s, week w, and model M, and the collection of all forecast samples from each time point $(\tilde{y}^{(k)}_{M,s,w+1})_{k=1}^{K}$ was used as a discrete representation of our forecasting distribution. The 2.5%th and 97.5%th quantiles of these samples provide 95% forecast intervals.

Figure 3.7 displays 95% forecast intervals for weeks 15 – 30 for each influenza season for the "ILINet and GFT Biased Model" and the "ILINet Only Model".

Notably, the forecast intervals produced by the "ILINet and GFT Biased Model" tend to be more accurate than those from the "ILINet Only Model". In seasons 8 and 9, for example, the bias model is able to capture the observed ILINet data as the epidemic starts to ramp up, while the "ILINet Only Model" consistently underestimates the observed data in this period. In season 3, the biased model produces similar but noticeably narrower forecasts than the "ILINet Only Model". It can also be observed that both models have a tendency to under-predict as the epidemic begins to ramp up, although this is somewhat less present in the "ILINet and GFT Biased Model". As both models treat latent severity as a local level process, they have

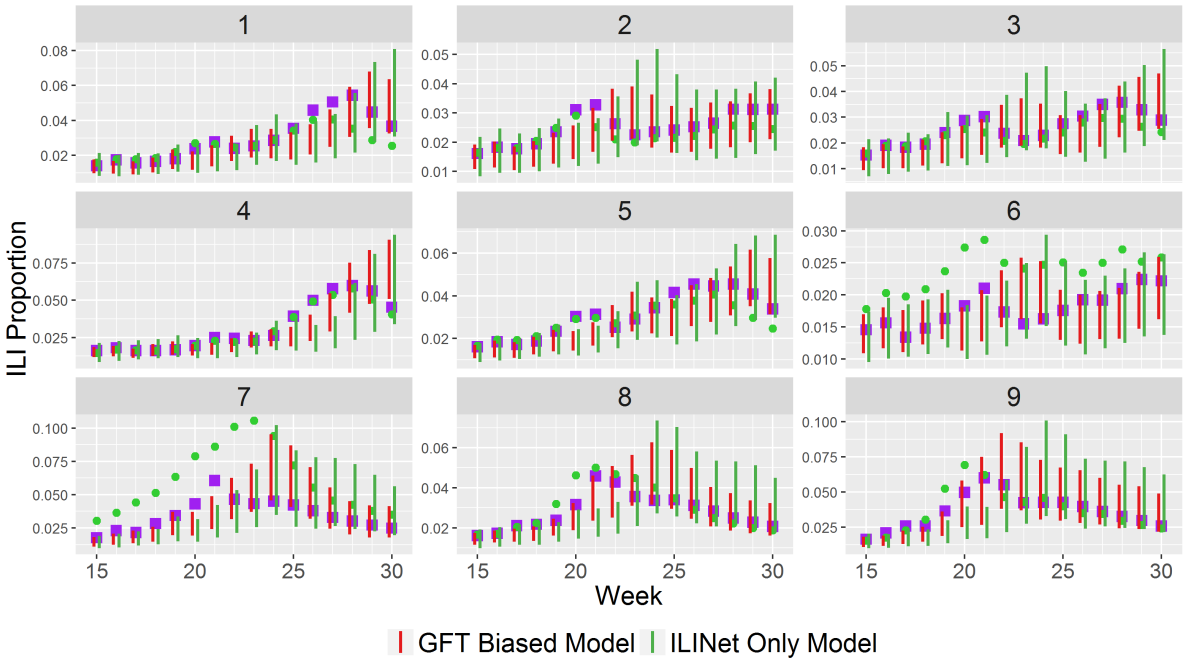

Figure 3.7: For both the "ILINet and GFT Biased Model" and the "ILINet Only Model", one week ahead 95% forecast intervals are plotted for ILINet data for all 9 seasons of data. Observed ILINet data (purple squares) and GFT data (green dots) are included. The forecast intervals produced by the biased model tend to do a better job of covering the observed ILINet data.

no structure to inform their predictions about the general shape of epidemic curves. This issue is resolved in Section 3.6.2, where SIR models are used for forecasting.

Figure 3.8 displays 95% forecast intervals for the "ILINet and GFT Unbiased Model" and the "ILINet Only Model".

For many seasons,there is little visual difference between the "ILINet and GFT Unbiased Model" and the "ILINet Only Model". Season 3, for example, has both models producing very similar intervals, with the GFT model's intervals being slightly narrower on average. However, some seasons produce noticeably different forecasts. For example, in seasons 6 and 7 – the two seasons when GFT data exhibited prolonged bias of a large magnitude – we see large variations in forecast interval length for the "ILINet and GFT Unbiased Model". The drastic changes in interval length occur because the unbiased model has difficulty in correctly estimating the variance in the ILINet and GFT data. For example, the prediction for week 21 of season 7 has a posterior median of 0.5 for $sigma_G$, as compared to a posterior median of around 0.03 for σ_I.

Figure 3.8: For both the "ILINet and GFT Unbiased Model" and the "ILINet Only Model", one week ahead 95% forecast intervals are plotted for ILINet data for all 9 seasons of data. Observed ILINet data (purple squares) and GFT data (green dots) are included. In seasons with large bias, the widths of the intervals produced by the unbiased model vary widely.

The high variance in the GFT data causes it to have almost no effect on the forecast for this week, resulting in a forecast interval for the "ILINet and GFT Unbiased Model" that matches the "ILINet Only Model" almost exactly. On the other hand, the prediction for week 22 of season 7 gives a posterior median of 0.5 for σ_I, and 0.02 for σ_G, in turn causing the predictions to be based almost entirely on the GFT data. A similar phenomenon occurs in season 6. Due to the inconsistent nature of the intervals produced by the "ILINet and GFT Unbiased Model" for seasons in which GFT data exhibits large bias, we recommend a careful analysis of the bias present in a data source before treating it as unbiased with a DLM model.

3.6.2 SIR Model Forecasting

In an attempt to more realistically model the process underlying disease outbreaks, we next turn to forecasting using the SIR models defined in Section 3.3.2. Note that when referring to the DLM models of the previous section, we will now describe them as the "DLM Combined Model with Bias", "DLM Combined Model without Bias", and "DLM ILINet Only Model", so

as to distinguish them from the SIR models presented in this section.

Forecasting was conducted using the PMCMC algorithm specified in Section 3.5.2. When using a latent SIR mode, one week ahead forecast density heights are no longer available analytically, so that a table in the style of Table 3.2 is no longer possible. Instead, we turn to visual assessments of prediction performance. In Figure 3.9, forecast intervals for the "SIR ILINet and GFT Biased Model" and the "SIR ILINet Only Model" are displayed. As compared

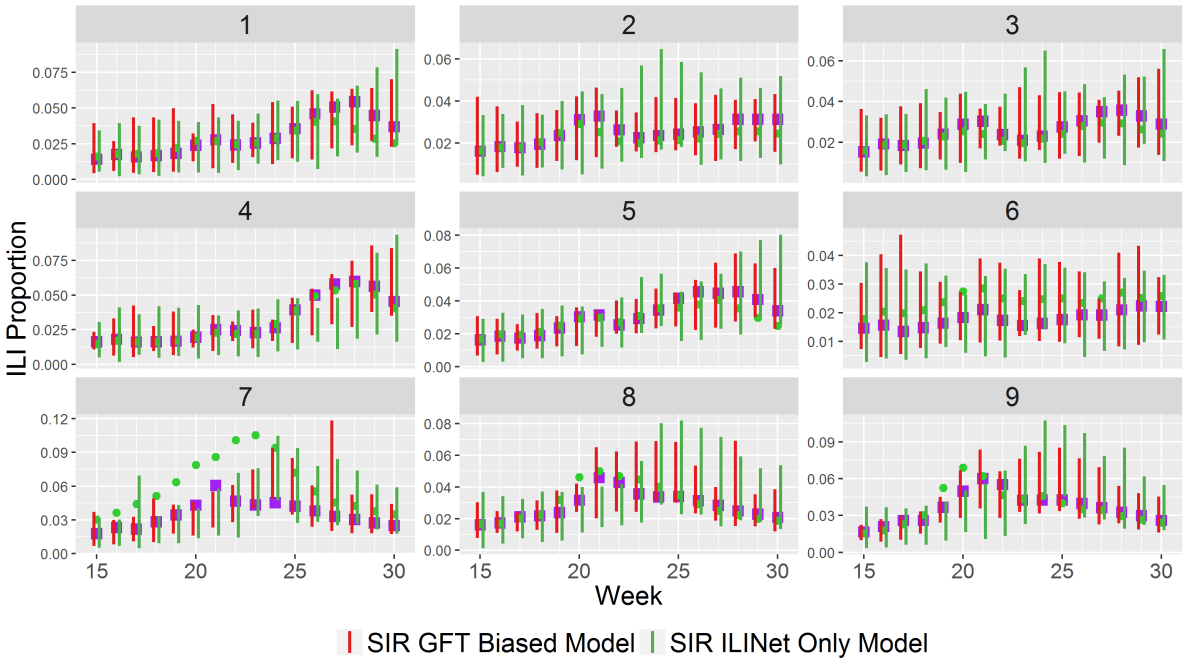

Figure 3.9: For both the "SIR ILINet and GFT Biased Model" and the "SIR ILINet Only Model", one week ahead 95% forecast intervals are plotted for ILINet data for all 9 seasons of data. Observed ILINet data (purple squares) and GFT data (green dots) are included. The biased model does a noticeably better job at covering the observed ILINet data during periods when severity ramps up quickly.

to the "SIR ILINet Only Model", the "SIR ILINet and GFT Biased Model" produces intervals that do a better job of covering the observed ILINet data. This is especially true for periods where the epidemic is ramping up quickly. In seasons 4, 7, 8, and 9 we see that the "SIR ILINet and GFT Biased Model" produces more accurate forecasts than the "SIR ILINet Only Model" for the weeks just before the epidemic reaches its peak. We also note that both SIR models do a better job of capturing the shape of the epidemic curve than the DLM models were able to.

Figure 3.10 compares the "SIR ILINet and GFT Unbiased Model" to the "SIR ILINet Only

Model". For seasons in which GFT data displays little bias, we see that the unbiased GFT model tends to produce more accurate, narrow forecast intervals than the ILInet only model. However, for seasons 6 and 7, the unbiased model produces highly inaccurate forecasts – in both seasons, the unbiased model is strongly influenced by the consistent overestimates provided by the GFT data, and in turn produces forecast intervals that are much higher than the ILINet observations.

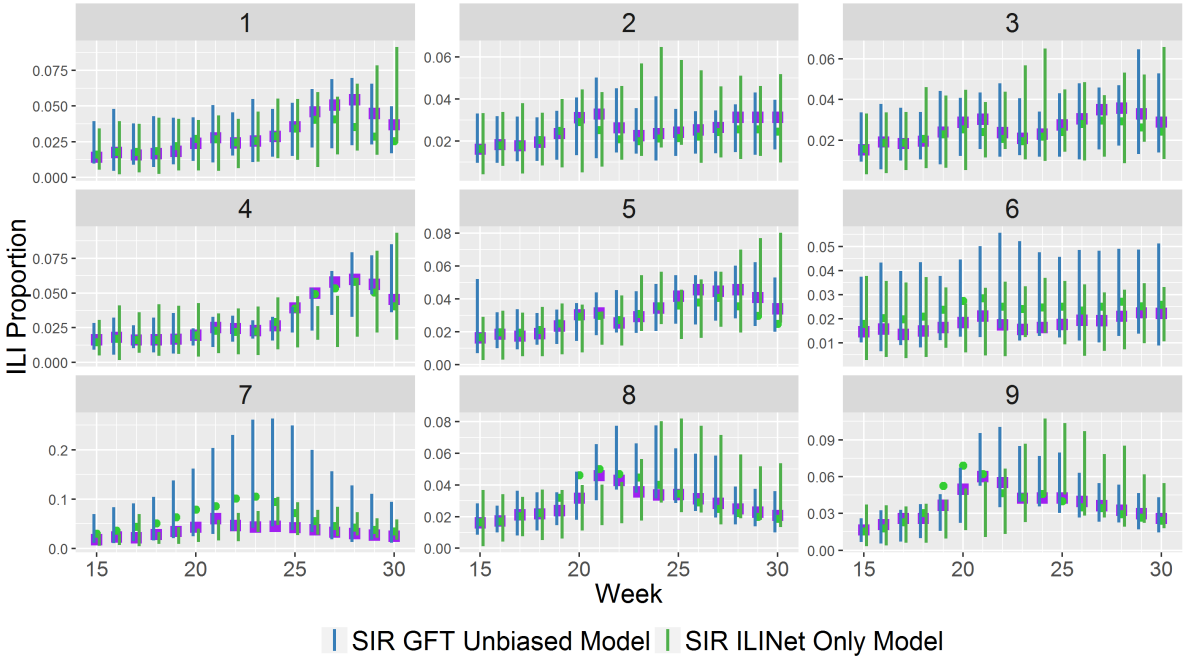

Figure 3.10: For both the "SIR ILINet and GFT Unbiased Model" and the "SIR ILINet Only Model", one week ahead 95% forecast intervals are plotted for ILINet data for all 9 seasons of data. Observed ILINet data (purple squares) and GFT data (green dots) are included. In seasons with large bias, the widths of the intervals produced by the unbiased model tend to be undesirably large.

We finally provide a comparison of all models (DLM and SIR) across all seasons, by computing their forecast coverage proportions– that is, the proportion of the one week ahead forecast intervals produced by each model that cover the true data point. We also calculate the average forecast interval width produced by each model, and the standard deviation of the forecast width. In Table 3.3, we compare the models based on coverage proportions for 50% one week ahead forecast intervals taken over weeks 13 to 33 for all 9 seasons of data.

Table 3.3 quantifies some features of predictions provided by the different models that also

DLM / SIR	GFT Data?	Bias?	Coverage Proportion	Mean Length	SD of Lengths
DLM	Yes	Yes	0.31	0.006	0.003
DLM	Yes	No	0.48	0.009	0.009
DLM	No	–	0.25	0.008	0.005
SIR	Yes	Yes	0.71	0.010	0.004
SIR	Yes	No	0.62	0.012	0.011
SIR	No	–	0.50	0.012	0.004

Table 3.3: 50% one week ahead forecast interval coverage proportions, mean interval lengths, and standard deviations of interval lengths for all models, based on whether the latent model was a DLM or SIR, whether GFT data was included in the model or not, and whether a bias term was included. Note that models without GFT data can not have a bias term. The forecasts produced by the DLM models tended to exhibit under-coverage, while the intervals produced by the SIR models tended to exhibit over-coverage.

had been observed in the previous figures. The DLM models produce narrower intervals, on average, than any of the SIR models. However, the coverage probabilities for the DLM models are all below the nominal level, with only the "DLM ILINet and GFT Unbiased Model" close to 50% coverage. The SIR models fare much better at providing at least nominal coverage, with the two SIR models which use GFT data actually providing greater than nominal coverage. The SIR models also tend to produce somewhat wider intervals than the DLM models. For both DLM and SIR models, the standard deviation of intervals lengths is highest when GFT is treated as an unbiased data source, owing to seasons 6 and 7 when the unbiased model produced very wide intervals.

Based on the visual and numeric forecast summaries, we feel the "SIR ILINet and GFT Biased Model" produces the "best" forecasts, in terms of coverage and length. Intervals produced by the "SIR ILINet and GFT Biased Model" are conservative, which is not necessarily a serious problem when forecasting. However, this raises the possibility that other forms of bias specification could potentially create even narrower intervals that retain nominal coverage.

3.7 Discussion

The simulation study provided a window into the dangers of treating biased data as unbiased. Even when the simulated data was only "lightly" biased – say, when σ_G was very large, so that the noise in the GFT data swamped the variance of the bias term – the "Combined Model

without Bias" performed no better, on average, than the "Combined Model with Bias". On the other hand, for highly accurate GFT data, the model with a bias term provided greatly better forecast ability than the unbiased model. It should be noted that this is not to say that the "Combined Model with Bias" produced forecasts with pinpoint accuracy, as in the simulation study with unknown parameters it often performed no better than the "ILINet Only Model". Rather, the forecasts produced by the "Combined Model without Bias" were so poor as to be unusable in any practical setting.

Perhaps unsurprisingly, when real disease outbreak data was used, SIR models proved better able to forecast the course of seasonal influenza epidemics than DLM models. However, for both DLM and SIR models, using a combined data model with a bias term resulted in superior forecasting for the rise and peak of epidemic curves as compared to a combined model without a bias term. The "SIR Combined Model with Bias" was the best performing model by both visual and numerical metrics, demonstrating the importance of incorporating bias correction into any data streams that may have bias regardless of the underlying latent epidemic model.

Further work on different, possibly adaptive bias specification methods could provide additional gains in accuracy when multiple data sources are used. Further, the use of three or more data sources could lead to further gains in forecasting accuracy, and should be explored given the wealth of additional real time influenza indicators now available through such sites as Twitter and Wikipedia. The framework laid out in this paper will allow modelers to include these new data sources while avoiding biasing their forecasts.

CHAPTER 4. A Hierarchical Model for Seasonal Disease Outbreaks

4.1 Introduction

There is great benefit to be gained from accurately monitoring and predicting the course of influenza outbreaks. Forecasting the trajectory of an outbreak over time can provide an early warning to health care centers that a large influx of influenza infected patients may arrive soon. Additionally, if the shape of an outbreak curve is forecast to look significantly different from curves in previous seasons (whether in terms of peak timing or peak intensity), this could provide early evidence that a non-seasonal influenza pandemic is occurring. However, prediction is challenging, because of both annual differences in the shapes of influenza epidemic curves and weekly variation in influenza severity indicators. Peak influenza activity in the United States, as measured by the proportion of laboratory samples which test positive for influenza virus infection, typically occurs between December and February (see Figure 3.1), but exhibits significant yearly variability. Even seasons which have similar peak times exhibit different levels of peak severity, and different rates of increase to and decrease from peak severity.

Many recent papers studying disease outbreak forecasting have used single-season models of disease outbreaks to conduct forecasts. These models rely on one or more data sources of disease severity from the current season, without considering data from previous seasons. In some disease outbreak situations, considering only data from the current outbreak is a logical choice. For example, if a new disease emerges for which no previous data are available, or if a disease outbreak occurs in a location where it has not previously been observed, no appropriate historical data may be available. However, for diseases which have been observed over multiple years in similar settings, and especially for diseases which occur seasonally, we will show that incorporating historical data can provide a discernible benefit to forecasting and parameter

estimation.

Much of the previous work in hierarchical disease modeling has focused on spatio-temporal modeling of disease spreads. One such example is Knorr-Held and Richardson (2003), who introduce a spatio-temporal model for detecting hyperendemic periods in meningococcal diseases. Lawson (2013) provides a survey of many topics and issues common to spatial disease models. Cauchemez et al. (2011) models the transmission of influenza-like illnesses (ILIs) among a population of elementary school students, incorporating information on the structure of each student's social network. For models which operate only on a temporal scale, Conesa et al. (2015) construct a hidden Markov model which uses ILINet data obtained from sentinel surveillance networks to detect the onset of seasonal influenza epidemics. In Black and McKane (2010), the effects of a seasonal forcing term on outbreak trajectories produced by a stochastic SIR model are examined.

We propose a model for seasonal disease outbreaks wherein the parameters governing the outbreak curve for each season are drawn from higher-level, non-season specific distributions. Using this temporal hierarchical modeling framework allows forecasts and parameters for the current season to be informed by observed outbreaks from previous seasons. Thus, for example, predictions that our model makes for epidemic severity one week in the future are based not only on data from the current influenza season, but from the shapes of epidemic curves from previous seasons as well. The primary purpose of our model is both to predict influenza severity one week into the future and to estimate relevant seasonal outbreak parameters as the season progresses.

We use ILINet and GFT data to inform our model. Using multiple data sources lets us benefit from the accuracy of the ILINet data, and the timeliness of the GFT data, by explicitly accounting for the potential bias of GFT data while allowing it to inform predictions as in Chapter 3. Our model uses both data sources to produce weekly estimates of latent epidemic severity for a given outbreak year by fitting a parametric outbreak curve for that year. Our model is parameterized in terms of parameters that have real-world interpretations in terms of peak timing, outbreak severity, outbreak duration, and baseline infected rate.

The data we use is described in detail in Section 2 of this paper. Section 3 first describes a

data model for a single season, which relates our data streams to latent epidemic severity, and a latent severity model for a single season, which relates latent epidemic severity to influenza outbreak parameters. It then provides a hierarchical model for relating outbreak and data parameters across seasons. Section 4 presents the results of a simulation study, in which our hierarchical multi-season model is compared to a single-season model. In Section 5, we apply our model to forecasting for the 2004-2005 through 2014 – 2015 influenza seasons in the United States. In Section 6 we discuss the results from our model and potential modifications which could be made to it.

4.2 Data

Our model uses both ILINet data provided by the CDC and GFT data provided by Google. In Sections 4.2.1 and 4.2.2, we provide details about how each of these measures of influenza severity are calculated.

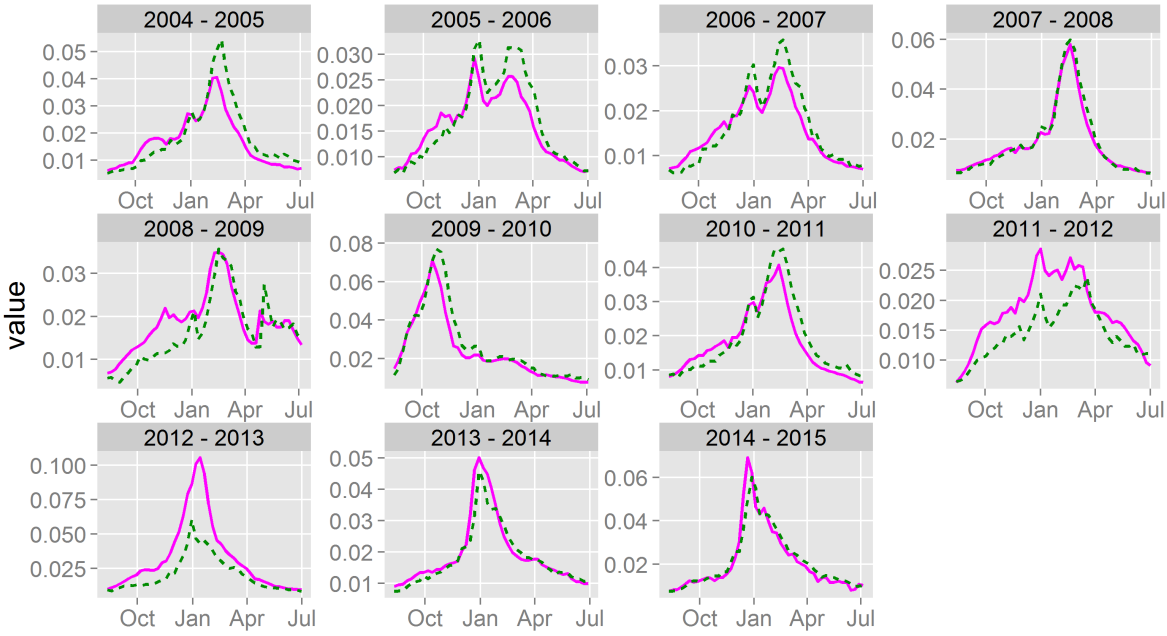

Figure 4.1: Weekly indicators of influenza-like illness in the United States across 11 seasons for the CDC's ILINet sentinel network (pink solid lines) and Google's Google Flu Trends (green dashed lines).

4.2.1 CDC ILINet Data

The U.S. Outpatient Influenza-like Illness Surveillance Network (ILINet) is a syndromic surveillance network of more than 3,300 health care providers, spanning all 50 states, which collects weekly information on the proportion of patients who visit for influenza-like illness (Brammer et al., 2011). An influenza-like illness is defined as a fever of greater than 100° Fahrenheit accompanied by a cough or sore throat, and for which no non-influenza cause has been identified (CDC, 2016). Each health care provider which participates in ILINet reports their weekly total number of patient visits, and their weekly number of patient visits for an influenza-like illness. Using this information, the overall proportion of patient visits due to an influenza like illness is calculated for Health and Human Services regions and Census divisions, as well as a proportion for the entire U.S.. ILINet data are typically reported at a lag of one to two weeks. In addition, as the ILINet data are dependent on volunteer submissions, it is also subject to revision by the CDC as more submissions are received.

ILINet data is published on a weekly basis, and is recorded along with its Morbidity and Mortality Weekly Report (MMWR) week. Information on obtaining ILINet data and calculating MMWR weeks can be found in Appendix Section A.5. Our data model, presented in Section 4.3.2, uses ILINet data on the national scale for the United States, starting with the 2004 – 2005 influenza season and ending with the 2013 – 2014 season. Similar to Yang et al. (2014), we omit the 2008 – 2009 and 2009 – 2010 seasons, as the data from both seasons show evidence of the A/H1N1 influenza pandemic. In the 2008 – 2009 season, evidence of the pandemic can be seen in the late rise of the proportion infected around week 40. In the 2009 - 2010 season, the pandemic causes the outbreak curve to rise more quickly and peak at an earlier week than a seasonal outbreak would. Figure 4.1 displays the influenza outbreak curves for these seasons, also showing the discrepancy in curve shape for the 2008 – 2009 and 2009 – 2010 seasons. Allowing data from these two pandemic seasons to inform our upper-level hierarchical distributions could detract from our model's ability to monitor and predict influenza infections due to seasonal outbreaks, rather than pandemics.

4.2.2 Google Flu Trends Data

Google Flu Trends provides estimates of influenza activity around the world by monitoring the frequency of searches for certain influenza-related keywords (Ginsberg et al., 2009). GFT data are provided as the expected number of influenza cases per 100,000 people, and is available on the city, regional, and national scale. We divide this data by 100,000 to put it on the same scale as the ILINet data .

The search keywords used by Google to produce GFT data are shown to be highly correlated with historical ILINet data (Ortiz et al., 2011). In addition, Google Flu Trends estimates are provided in "near real-time" Google (2016), making them a more current counterpart to the ILINet data. However, Google Flu Trends is also a less direct indicator of influenza prevalence than the ILINet data. GFT data are dependent upon user searches through the Google search engine, which in turn could be influenced not only by users experiencing influenza like symptoms, but also by events without a direct relation to a season's influenza outbreak - for example, excess media coverage, or symptoms from non-influenza diseases could both inflate the estimate provided by GFT. Evident in Figure 1, there are a few seasons where GFT data and ILINet data provide markedly different curves. For example, GFT provides a peak infected proportion more than twice as large as ILINet's in the 2012–2013 season. Lazer et al. (2014) discuss some possible reasons for the observed inaccuracy in GFT data, including the possibility that Google has modified its search algorithm over time, guiding users towards certain search terms, which could necessitate a recalibration of their GFT algorithm. Since Google has not published the keywords they use to derive the GFT index, it is difficult to know whether these search engine changes have been accounted for.

4.3 Models for Latent Epidemic Severity and Data

We first introduce a general model for latent influenza outbreak severity for a single season in Section 4.3.1. Next, in Section 4.3.2, we relate observed ILINet data to the latent outbreak severity. A framework is also provided for modeling additional data sources which may be biased estimates of latent severity, and that framework is applied to a model for GFT data.

In Section 4.3.3 we provide a hierarchical model which allows latent outbreak severity from previous seasons to inform outbreak parameters for future seasons. Additionally, data variance parameters from previous seasons can help to inform predictions for parameters of the current season. Prior distributions and model inference using an MCMC algorithm are discussed in Section 4.3.4. Finally, we describe a prediction method to assess the effect of hierarchical modeling in Section 4.4.1.

4.3.1 Latent Epidemic Model For a Single Season

The latent epidemic model estimates the latent epidemic severity curve in a population on the logit scale. Latent epidemic severity is defined in the same units of measurement as the ILINet data, that is, the proportion of all health care center patients who tested positive for an ILI. Notably, latent severity here does not represent the proportion of all U.S. citizens with an ILI, but rather only the proportion of people who visited a health care center and who tested positive. Shaman and Karspeck (2012) provide a method for expanding the scope of inference from health center patients to the population of all U.S. citizens. However, as our model is primarily interested in forecasting ILINet data in future weeks, we model latent severity on the scale of the ILINet data.

We let $x_{s,w}$ be the logit of the latent severity in season s and week w. We model $x_{s,w}$ for a season s and week w using a parametric curve which is a function of the week w and has season specific parameter γ_s as in the first equality of equation (4.1).

$$x_{s,w} = g(w, \boldsymbol{\gamma}_s) = e^{\gamma_{s,1}} \phi\left(w, e^{\gamma_{s,2}}, e^{\gamma_{s,3}}\right) + \gamma_{s,4} \tag{4.1}$$

For influenza, we investigate the use of normal and log-normal probability density functions (pdfs) with a baseline. Thus, in the second equality of equation (4.1) we have ϕ as the pdf of either a normal or log-normal distribution with mean parameter $e^{\gamma_{s,2}}$ and variance parameter $e^{\gamma_{s,3}}$. Based on the data in Figure 3.1, we have found that normal and log-normal curves tend to produce similar short-term forecasts. In this paper, we take ϕ to be the pdf of a normal curve. For a normal pdf, the season-specific parameters in our latent model have meaningful interpretations in terms of the season s outbreak: $\gamma_{s,1}$ determines the height, $\gamma_{s,2}$ determines the

peak timing, $\gamma_{s,3}$ determines the longevity, and $\gamma_{s,4}$ determines the baseline rate of ILI visits. In addition to using the latent model for outbreak forecasting, learning about the posterior distributions of the γ_s parameters can provide insight into biological features of the outbreak.

The proposed latent epidemic model is a departure from both the DLM and SIR models of Chapter 3. However, we note that it has a few salient features that make it intriguing in a hierarchical context. First, the latent curves for each season are entirely determined by the four $\gamma_{s,i}$ parameters. In turn, this greatly reduces the amount of time it takes to conduct inference on the model via MCMC, as compared to the PMCMC scheme used for the SIR models. Secondly, the four γ parameters which determine the curve for each season have direct interpretations in terms of key outbreak features. While the β and γ parameters of an SIR model also have biological interpretations, we feel that it is worth considering other outbreak parameters, and specifically the peak week and outbreak spread as determined by $\gamma_{s,2}$ and $\gamma_{s,3}$, as of interest in their own right.

A hierarchical extension to the Chain Binomial SIR model of Chapter 3 is provided in Appendix Section B.2, together with a hierarchical PMCMC algorithm which can be used to conduct inference on it.

4.3.2 Data Models

We consider data for each influenza season starting with the 32nd MMWR week in the first year of the season (approximately the middle of August), and continuing for 46 weeks after that (approximately the end of June or beginning of July in the second year). MMWR weeks are used by health care providers in the United States for the purposes of reporting disease incidences. There is typically very little seasonal influenza activity in late June, the month of July, and early August, so we exclude that data from modeling. In modeling, we denote the first week of data for each season (MMWR week 32) as $w = 1$. Thus our model uses data with $w = 1, \ldots, 47$.

Suppose we have the logit of J data sources on epidemic severity for week w in season s, labeled $D_{s,w}^{(j)}$ for $j \in \{1, \ldots, J\}$. We relate each data source to our latent epidemic model $x_{s,w}$

through equation 4.2a, where $\epsilon_{j,s,w}$ are error terms.

$$D_{s,w}^{(j)} = x_{s,w} + \beta_{j,s,w} + \epsilon_{j,s,w}, \qquad \epsilon_{j,s,w} \sim N(0, \sigma_{j,s}^2) \tag{4.2a}$$

In equation 4.2a, $\beta_{j,s,w}$ is the bias of data source j for week w in season s. Possible models for the bias $\beta_{j,s,w}$ include a deterministic function of the observed bias from the previous week, a function depending on additional weekly covariates, or an AR process. Alternatively, data sources can be modeled as unbiased by setting $\beta_{j,s,w} = 0$.

We use equation 4.2a to relate our two data sources, ILINet and GFT, to our latent model. Specifically, we relate the logit of ILINet data for season s and week w, labeled $I_{s,w}$, to latent severity by setting $\beta_{I,s,w} = 0$ for all s, w. As ILINet data are the gold standard for influenza epidemic modeling, we treat it as an unbiased estimate of the underlying outbreak.

GFT data are available approximately two weeks earlier than ILINet data, but can provide inaccurate measurements for certain seasons, as noted in Section 3.2.2. For our analysis and forecasting we model the logit of GFT data in season s and week w, labeled $G_{s,w}$, using equation 4.2a, setting $\beta_{G,s,w} = \rho(G_{s,w-1} - x_{s,w-1})$. The bias term $\beta_{G,s,w}$ sets the expected bias of the logit of GFT data in week w as equal to the latent bias from the previous week multiplied by an adjustment term ρ, allowing the expected bias in our GFT data to be carried over from week to week. The ρ term allows the bias to be shrunk towards (or expanded away from) the true latent severity.

Incorporating a lagged bias allows the model to account for situations where the bias of GFT data is temporally correlated - for example, if the epidemic is receiving an unusual amount of media coverage for part of a given season, or if another disease with similar symptoms has an outbreak, this could cause more people to conduct Google searches for influenza-related phrases, in turn causing GFT to continually overestimate the latent infected severity. Figure 3.1 displays seasons such as 2012–2013, where GFT consistently over-estimates ILINet data, and seasons such as 2005 – 2006, where GFT provides consistent under-estimates. Thus modeling GFT data as having temporally correlated bias appears justified. Additionally, as Chapter 3 demonstrated, modeling GFT as a biased source of data will tend to provide better forecasting

results.

4.3.3 Hierarchical Modeling of Outbreak and Data Parameters

We construct a hierarchical framework for both the season specific outbreak parameters γ_s and the season specific data variance parameters σ. For the outbreak parameters we assume $\gamma_{s,1:4} \sim MVN(\mu, \Sigma)$, where where $MVN(\mu, \Sigma)$ represents multivariate normal distribution with mean vector μ and covariance matrix Σ. Modeling seasonal outbreak parameters as coming from a multivariate normal distribution allows us to model correlations between outbreak parameters within a season. For example, seasons with late peak weeks (as determined by $\gamma_{s,2}$) may also tend to have larger outbreak durations (as determined by $\gamma_{s,3}$). Examining the posterior distribution of the Σ matrix will allow us to determine what correlations may be present between seasonal outbreak parameters. Using a hierarchical framework for the latent epidemic model allows us to borrow information across seasons when estimating season-specific parameters, which could lead to benefits in both forecasting ability and parameter estimation for new seasons.

We also propose a hierarchical framework for the data variance parameters $\sigma_{j,s}$ for data source j in season s in order to borrow information about the noise across seasons. Specifically, we assume $\sigma_{j,s} \overset{ind}{\sim} LN(\theta_j, \lambda_j^2)$, where $LN(\theta, \lambda^2)$ denotes a log-normal distribution, the log of which is normally distributed with mean θ and variance λ^2. Hierarchical modeling of variance parameters can likewise provide a benefit to forecasting in future seasons – if one data stream is historically more noisy than another, the hierarchical model will produce variance parameters for a new season which reflect that noise, in turn influencing forecasts made for the new season.

Information on prior distributions for hierarchical modeling parameters can be found in Section 4.3.4.

4.3.4 Prior Distributions and Inference

For $i = 1, \ldots, 4$, we use the following prior distributions for the parameters in our latent epidemic severity model $\mu_i \overset{ind}{\sim} t_\nu(m_i, s_i^2)$ and $\kappa_i \overset{ind}{\sim} Ca^+(0, S_i)$ where $Ca^+(0, S)$ is a half-Cauchy distribution with scale parameter S (Gelman et al., 2006). Prior distributions for our

ILINet model parameters are $\theta_I \sim t_\nu(m_I, s_I^2)$ and $\lambda_I \sim Ca^+(S_I)$. Similarly, for the GFT model, we use $\theta_G \sim t_\nu(m_G, s_G^2)$, $\lambda_G \sim Ca^+(S_G)$, and $\rho \sim N(m_\rho, s_\rho^2)$.

Inference on the model was conducted using an MCMC algorithm implemented through JAGS (Plummer et al.,). JAGS was run using the rjags package (Plummer, 2016) within the R statistical computing language (R Core Team, 2015b). Additional details on the settings and convergence diagnostics for our latent epidemic model and data models can be found in Appendix Section B.1.

4.4 Simulation Study

A simulation study was conducted to determine the ability of our model both to generate data that appear realistic and to accurately estimate posterior distributions of model parameters. To simulate data, we first obtain the medians of the posterior distributions of our hyperparameters for our latent epidemic model (μ_i, κ_i for $i = 1, \ldots, 4$) and for our data model ($\theta_I, \lambda_I, \theta_G, \lambda_G$) given the first 8 seasons worth of observed ILINet and GFT data (the 2003 – 2004 season through the 2013 – 2014 season). These posterior medians are then used to generate 10 new data sets. Each of these generated data sets contains 9 simulated seasons worth of data, mimicking the amount of observed data we actually have. To generate a single data set, we first use the medians of the posterior distributions of our hyperparameters to generate new values $\hat{\gamma}_{s,i}$, $\hat{\sigma}_{s,I}$, and $\hat{\sigma}_{G,s}$ for $s = 1, \ldots, 9$. The $\hat{\gamma}_{s,i}$ values are used to calculate $\hat{x}_{s,w}$ for each season $s = 1, \ldots, 9$ and week $w = 1, \ldots, 47$, and in turn, the $\hat{x}_{s,w}$ values were used, along with $\hat{\sigma}_{I,s}$, $\hat{\sigma}_{G,s}$, and the posterior median of ρ, to generate ILINet data $\hat{I}_{s,w}$ and GFT data $\hat{G}_{s,w}$ for 9 simulated seasons.

Once the 10 simulated data sets, each with 9 seasons of data, have been generated, we conduct an MCMC algorithm to conduct forecasting and inference for each simulated data set. Of interest is not only the forecasting ability of our model, but also the ability of our model to provide inference on the $\gamma_{s,i}$ outbreak parameters as the season progresses. MCMC algorithms are run for the hierarchical model specified in Section 4.3.3, using both the ILINet and GFT data models of Section 4.3.2. We refer to this model as the "ILINet and GFT Multi-Season Model". In addition to assessing the performance of the "ILINet and GFT Multi-Season

Model", we describe three competing models as follows. The "ILINet Only Multi-Season Model" is specified by using the hierarchical structure of Section 4.3.3, but only using the ILINet data model (so no GFT data are used for inference). The "ILINet and GFT Single-Season Model" is specified by using both ILINet and GFT data streams, but only modeling the current season using the single season latent model of Section 4.3.1. The "ILINet Only Single-Season Model" similarly is specified by using only ILINet data, and only modeling the current season using the single season latent model of Section 4.3.1. Thus when forecasting week w in season 9 using either of the Single-Season models, only data from earlier in season 9 are used to inform that forecast.

Comparing the inferential results provided by the Multi-Season models to those from the Single-Season models will help to determine the extent to which incorporating past data are of use in timely parameter estimation and forecasting. Likewise, although a lesser focus of this chapter, comparing the results of the ILINet and GFT models to those from the ILINet only models will help to figure out what role the inclusion of GFT data plays in accurate forecasting.

For each simulated data set, we forecast the epidemic curve for the 9th season, mirroring the forecasting done with real world data in 4.5.1. We describe our method for forecasting in Section 4.4.1. In Section 4.4.2 we evaluate the forecasting performance of our models. In addition to assessing forecasting ability, we also examine the posterior distributions given by our models as the season progresses to determine if the models are able to recover the known parameter values which generated the data. A detailed analysis of these posterior distributions is provided in Section 4.4.3.

4.4.1 Forecasting Methods

To conduct forecasting for the Multi-Season model, for each of our 10 simulated data sets we use data from the first 8 simulated seasons to forecast the course of the influenza outbreak for the 9th simulated season. We use the latent epidemic model given in Section 4.3.1, choosing a normal curve for ϕ, and we use the data model for ILINet and GFT given in Section 4.3.2. For each week $w = 1, 2, \ldots, 47$, and each season $s = 1, \ldots, 8$, we provide our model with $\hat{I}_{s,w}$ and $\hat{G}_{s,w}$. Forecasting for season 9 is accomplished by running 47 separate MCMC algorithms,

each using a different amount of simulated data from the 9th season. Specifically, each run used season 9 data up to a specified end week, labeled w_e, for $w_e = 3, \ldots, 46$. The models with data up to week w_e were used to predict the outbreak severity for the next week, $w_e + 1$.

As ILINet data is reported at a lag of up to 2 weeks, for each value of w_e we assume that we have simulated season 9 ILINet data from week 1 up to week $w_e - 2$, labeled $\hat{I}_{9,1:w_e-2}$. On the other hand, GFT data are not reported at a lag, so we assume that we have simulated season 9 GFT data from week 1 up to week w_e, labeled $\hat{G}_{9,1:w_e}$. Using different values for w_e allows the model's predictive accuracy to be assessed when given different amounts of data, and mimics the real world situation where new data may be received and incorporated into the model on a weekly basis.

We then forecast the latent severity for every week in season 9, denoted $x_{9,1:47}$, by using an MCMC algorithm to generate K samples $\boldsymbol{\gamma}_9^{(k)}$ for $k \in_{1,\ldots,K}$ from the posterior distribution $(\boldsymbol{\gamma}_9 | \hat{I}_{9,1:w_e-2}, \hat{G}_{9,1:w_e})$. Once these samples are obtained, we can generate K realizations of the latent epidemic curve given data up to week w_e as $\hat{x}_{9,1:47}^{(k)} = \left(g(1, \boldsymbol{\gamma}_9^{(k)}), \ldots, g(47, \boldsymbol{\gamma}_9^{(k)})\right)$. Forecasts for ILINet data are then obtained by taking 10,000 samples from the posterior distribution of $\sigma_{I,9}^2$, the data variance of ILINet data for the 9th simulated season. For each posterior sample $(x_{9,1:47}, \sigma_{I,9}^2)^{(k)}$, we generate $\epsilon_{I,9,w_e}^{(k)} \sim N(0, (\sigma_{I,9}^{(k)})^2)$, and then calculate $I_{9,w_e}^{(k)} = x_{9,w_e}^{(k)} + \epsilon_{I,9,w_e}^{(k)}$, giving us samples of the predicted ILINet severity for week w_e. Using these samples, we construct 95% forecast intervals by taking the 2.5th and 97.5th quantiles of the generated data.

Forecasts from the Single-Season model are constructed identically, except that no data from earlier seasons in provided to the model.

4.4.2 Forecasting with Simulated Data

Figure 4.2 displays 95% one week ahead forecast intervals for the simulated ILINet data for season 9 of each of the 10 simulated data sets. We focus here on weeks 13–33 in season 9, as in Chapter 3, as these weeks tend to cover the rise and peak of historical seasonal influenza outbreaks. The data simulated from the Multi-Season model do a good job at providing a range of curve shapes, spreads, and peak times.

Comparing the two models on their one week ahead forecast ability, we see that both models

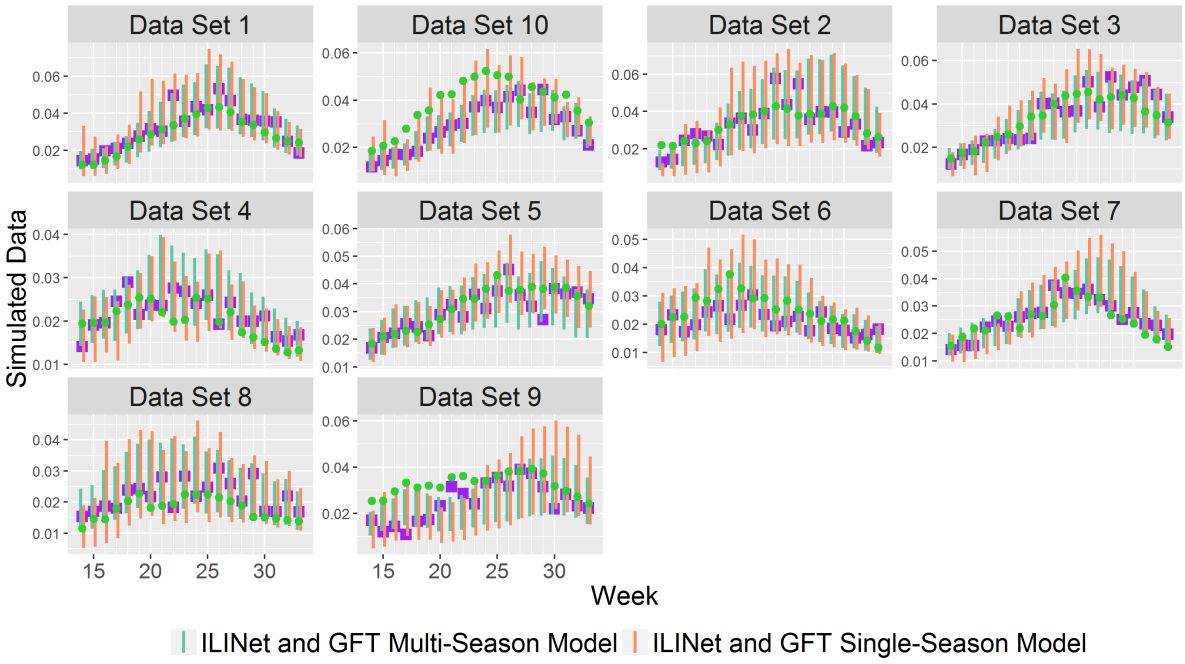

Figure 4.2: ILINet data (triangles) and GFT data (circles) for weeks 15 - 30 for each of the 10 simulated data sets. For each forecast week, we provide one week ahead 95% forecast intervals for ILINet data for both the Multi-Season and Single-Season models. The Multi-Season model uses historical data from the previous 8 simulated seasons for each data set, while the Single-Season model uses only data from the ninth simulated season. The Multi-Season model tends to produce narrower intervals than the Single-season model.

do a good job of capturing the ILINet data for the weeks considered. For many simulated data sets, such as data sets 1, 3, and 7, only a slight distinction can be seen between the intervals provided by the two models. However, one can notice that the intervals produced by the Multi-Season model tend to be narrower than those produced by the Single-Season model, while still maintaining good coverage. This is further borne out by Table 4.1, which displays coverage probabilities and average interval lengths for the 9th season of each of the 10 simulated data sets.

Both models provide near nominal coverage over the 21 weeks considered, with data set 10 possibly providing some difficulty for both models. Since the coverage proportions are only calculated over 21 weeks, we hesitate to assign too much importance to slight deviations from nominal coverage as seen in data set 10. We also note that the Multi-Season model tends to provide comparable coverage while producing narrower intervals, a desirable feature

	Data Set					
Multi-Season?	1	2	3	4	5	6
Yes	1.00 (0.007)	1.00 (0.005)	0.81 (0.009)	0.95 (0.007)	0.90 (0.004)	1.00 (0.006)
No	1.00 (0.009)	1.00 (0.007)	0.95 (0.011)	1.00 (0.008)	0.90 (0.004)	0.95 (0.006)

Multi-Season?	7	8	9	10
Yes	1.00 (0.006)	0.95 (0.005)	0.95 (0.006)	0.81 (0.006)
No	1.00 (0.008)	0.86 (0.005)	0.95 (0.007)	0.86 (0.009)

Table 4.1: 95% one week ahead forecast interval coverage proportions between the week 13 and 33 for each of the 10 simulated data sets, calculated as the proportion of 95% one week ahead forecast intervals in each season which contain the true data. Average lengths of 95% intervals are provided for each simulated data set in parentheses. Coverage proportions are displayed for both the Multi-Season model (Multi-Season = Yes) and the Single-Season model (Multi-Season = No).

in forecasting.

We remark that 50% coverage intervals were investigated between the two models as well. These intervals consistently fell below nominal coverage for both models when considered between the weeks of 13 and 33 in each simulated season. The 50% intervals do have nominal coverage when considered over the whole season (week 1 to week 47), but tend to be too narrow for use during the ramp up and peak of outbreaks. As such, we recommend the use of wider credible intervals, such as the 95% intervals discussed above.

4.4.3 Posterior Distributions from Simulated Data

Figure 4.3 displays posterior credible intervals for the γ_2 parameters for the 9th season of generate data from each of the 10 data sets. Credible intervals are shown as a function of time, showing how they evolve over the course of an outbreak season. For every data set, the credible intervals provided by the Multi-Season model are much narrower than the Single-Season model at the start of the season, reflecting the knowledge given to the Multi-Season model by data from earlier seasons. Additionally, the credible intervals from the Multi-Season model tend to cover, or come close to, the true generating parameter value, even for very early weeks in the season. The Single-Season model has great difficulty in narrowing down reasonable values for the parameter until much later in the season. Finally, we note that the intervals produced by the Single-Season model tend to converge to those of the Multi-Season model as the Single-Season

model receives more data. This will be seen again in the real data analysis of Section 4.5.2.

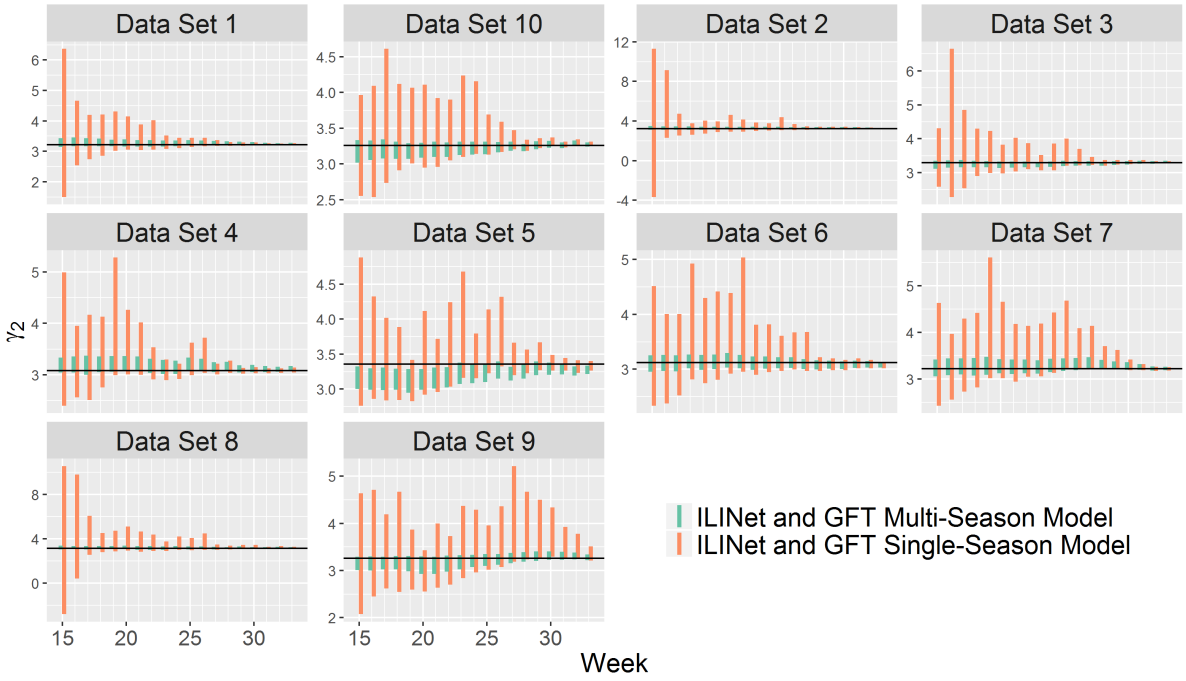

Figure 4.3: For each simulated data set, posterior 95% credible intervals for the values of γ_2 in season 9 given data up to week w for $w \in (15, \ldots, 33)$. Credible intervals are provided for both the Multi-Season and Single-Season models. The black horizontal line displays the true value of γ_2 that was used to generate the data.

Figure 4.4 displays posterior credible intervals for the γ_3 parameters for the 9th season of generate data from each of the 10 data sets, and shows a similar relationship between the two models to that seen in Figure 4.3. The Multi-Season model is again much better able to produce narrow, accurate credible intervals at early points in the season than the Single-Season model is. We note this is not due to parameter values being the same across all seasons – γ_3 values range from near 4 in data set 7 to over 5 in data set 5. In both situations, the Multi-Season model is able to produce narrow credible intervals which cover the true generating parameter value much more quickly than the Single Season model is.

4.5 Real Data Analysis

In Section 4.5.1, we use real ILINet and GFT data to forecast the trajectory of all available influenza seasons. We preform a leave-one-out cross-validation to examine the accuracy of our

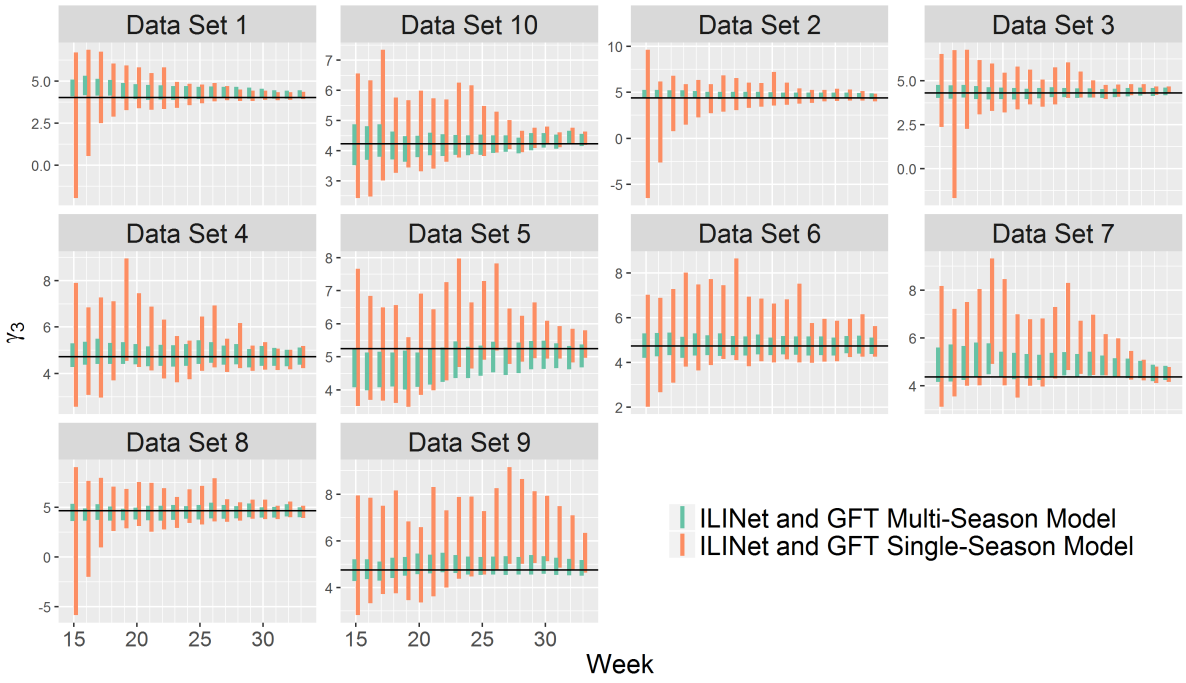

Figure 4.4: For each simulated data set, posterior 95% credible intervals for the values of γ_3 in season 9 given data up to week w for $w \in (15, \ldots, 33)$. Credible intervals are provided for both the Multi-Season and Single-Season models. The black horizontal line displays the true value of γ_3 that was used to generate the data.

forecasts across multiple seasons. The ability of our models to accurately estimate posterior distributions for outbreak parameters is detailed in Section 4.5.2.

4.5.1 Forecasting with Real Data

We analyze the forecasting ability of our models by conducting a leave-one-out analysis of our 9 seasons worth of influenza data. To conduct forecasts for season $s \in (1, \ldots, 9)$, we provide the Multi-Season model with full ILINet and GFT data from all other seasons. To forecast for week $w+1$ in season s, the Multi-Season model is also provided with ILInet data up to week $w-2$ and GFT data up to week w. The Single-Season model is given the same ILINet and GFT data from season s, but no data from any other seasons. Forecasting is conducted in the manner described in Section 4.4.1.

We begin by examining forecasts for Season 7 between our two models. Figure 4.5 displays forecast intervals for our two models for the seventh season of ILINet and GFT data. We see that the Multi-Season model does a far superior job of producing intervals which cover the observed

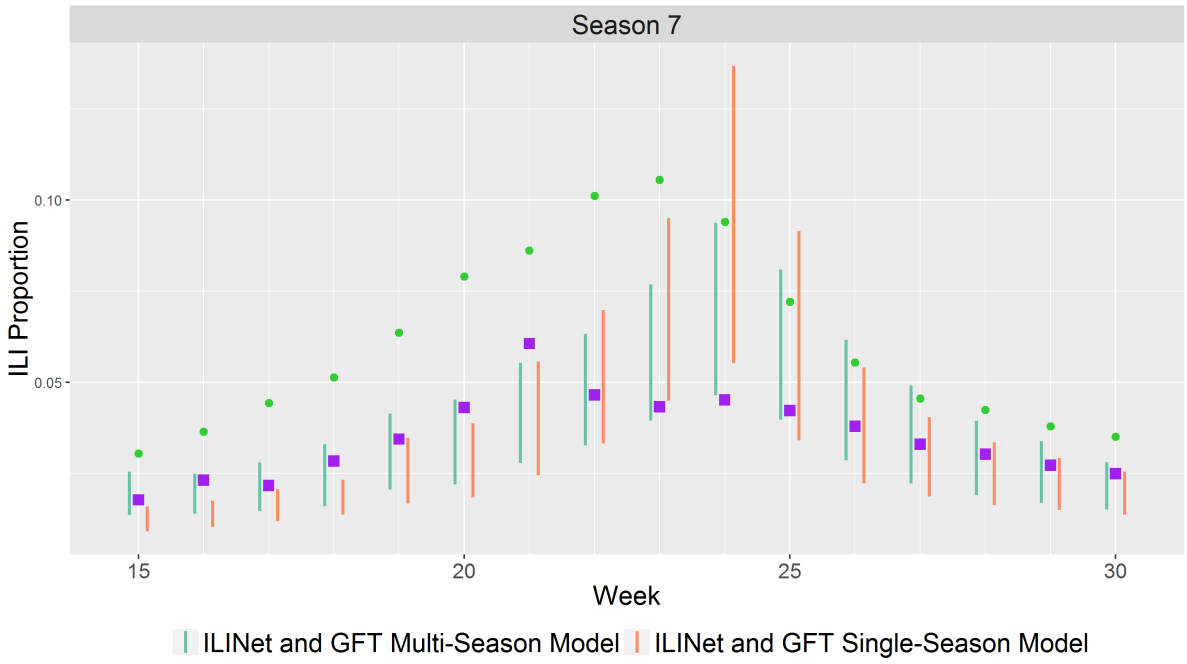

Figure 4.5: ILINet data (purple squares) and GFT data (green circles) for weeks 15 – 30 in season 7. For each week, we provide 95% one week ahead forecast intervals for ILINet data Intervals are provided for both the Multi-Season and Single-Season models. To produce intervals for the seventh season, The Multi-Season model is provided with complete data from all other seasons, while the Single-Season model is only provided with data from season 7. The Multi-Season model tends to cover the observed ILINet data more often than the Single-Season model.

ILINet data in each week than the Single-Season model does, especially for earlier weeks in the season. The Single-Season model consistently under-predicts the values of the ILINet data for weeks 15 – 21. This under-prediction occurs because, as will be seen in Section 4.5.2, the Single-Season model consistently overestimates the value of γ_3 for these weeks. High values of γ_3 in turn cause more spread out epidemic curves, which rise more slowly and lead to under-predictions. The Multi-Season model is able to more quickly pick up on a reasonable value of γ_3, and in turn does a better job at tracking the ILINet data than the Single-Season model does.

Figure 4.6 displays forecast intervals between the two models for all 9 seasons of data. Immediately evident is that both models have difficulty in forecasting for seasons when epidemic severity ramps up quickly. For example, in season 4 both models produce highly inaccurate forecasts for weeks 25 – 28, a period during which severity increases rapidly. A similar phe-

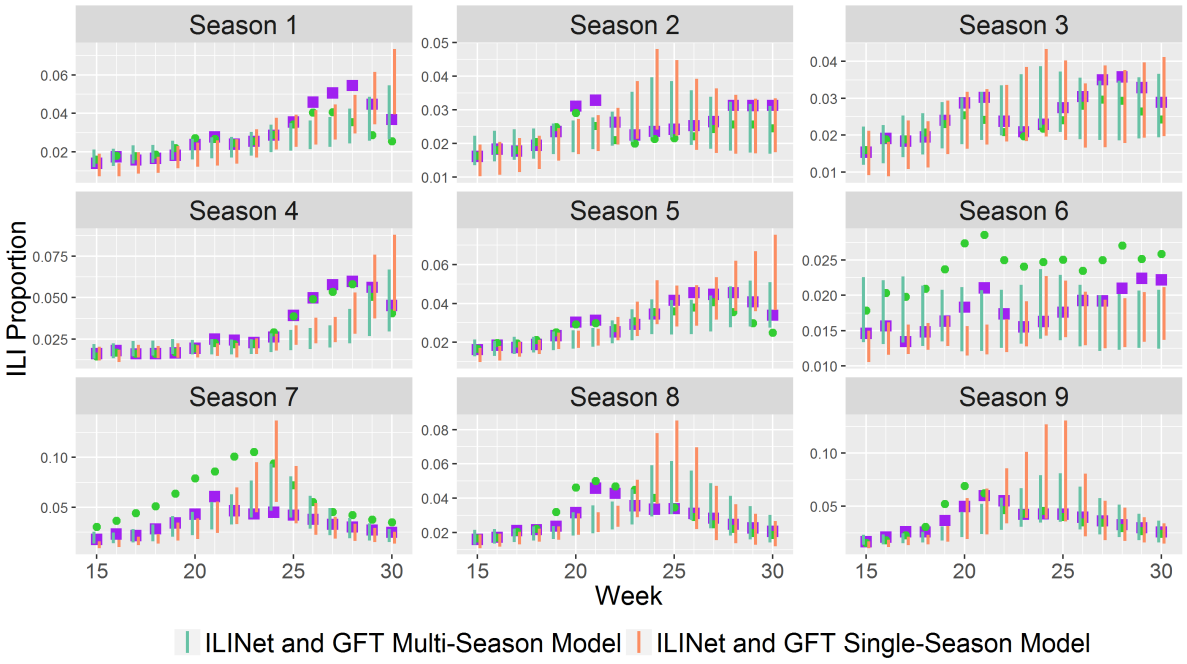

Figure 4.6: ILINet data (purple squares) and GFT data (green circles) for weeks 15 – 30 for seasons 1 – 9. For each week, we provide 95% one week ahead forecast intervals for ILINet data Intervals are provided for both the Multi-Season and Single-Season models. To produce intervals for a given season, The Multi-Season model is provided with complete data from all other seasons, while the Single-Season model is only provided with data from that season.

nomenon happens for portions of seasons 1 and 8. In all of these under-estimation scenarios, the models have difficulty correctly estimating the γ_3 parameter which controls the spread of the curve. The overly large estimates for γ_3 produced by the two models for these scenarios results in curves with more gradual changes in slope, in turn leading to under-estimation issues. This is, unfortunately, a feature of the model parametrization we have chosen. However, we believe that given more historical seasons of data, the Multi-Season model would be better able to pick up on these rapid ascents and descents in severity.

Putting aside the inaccuracy of both models for the aforementioned periods of quickly increasing severity, we see that the Multi-Season model tends to provide somewhat more accurate forecasts. This is noticeable in season 6, where the hierarchical information about the variance of the two data streams causes the Multi-Season model to produce appropriately wide forecast intervals which do a reasonable job of covering the true data. In this season, the Single-Season model produces intervals that are too narrow before week 24. We also remark that as each sea-

son progresses, the Single-Season intervals tend to converge to the Multi-Season intervals. This directly shows the benefit of using multiple seasons of data – the Single-Season model needs to observe 20 to 30 weeks of data in order to learn enough about the parameters to produce intervals that mirror the Multi-Season intervals. The historical data used by the Multi-Season model causes it to produce more reasonable posterior distributions for outbreak parameters during the earlier weeks in the season than the Single-Season model is able to. The ability of the Multi-Season model to produce accurate posterior distributions for parameters is examined more in Section 4.5.2.

We also provide numerical summaries of forecasting performance in Table 4.2, which displays average 95% coverage proportions and average interval lengths for our two models for each season. Notably, the Multi-Season model provides coverage at least as good as the Single-Season model for all but one season. Additionally, the intervals produced by the Multi-Season model tend to be narrower than the Single-Season model. We note that, as in Section 4.4.1, 50% forecast intervals were also examined. These intervals tended to be too narrow over weeks 13 – 33 in each season, and consistently under-covered the ILINet data for both models, so they are not included here. We again recommend the use of wider than 50% intervals, such as the 95% intervals of this section.

	Season					
Multi-Season?	1	2	3	4	5	6
Yes	0.76 (0.005)	0.86 (0.004)	0.86 (0.005)	0.81 (0.006)	0.76 (0.005)	0.71 (0.003)
No	0.59 (0.006)	0.86 (0.005)	0.95 (0.006)	0.57 (0.007)	0.67 (0.006)	0.62 (0.002)

Multi-Season?	7	8	9
Yes	0.76 (0.007)	0.90 (0.005)	0.81 (0.007)
No	0.38 (0.007)	0.67 (0.006)	0.81 (0.010)

Table 4.2: Average 95% one week ahead forecast interval coverage proportions for seasons 1 through 9, calculated as the proportion of 95% one week ahead forecast intervals in each season which contain the true data. Average lengths of 95% intervals are provided for each season in parentheses. Coverage proportions are displayed for both the Multi-Season model (Multi-Season = Yes) and the Single-Season model (Multi-Season = No).

4.5.2 Posterior Distributions of Outbreak Parameters

In this Section, we analyze each model's ability to accurately capture the posterior distribution of outbreak parameters in a timely manner. We focus specifically on the parameters γ_2 and γ_3, which determine the peak week of the outbreak and the spread of the outbreak curve, respectively.

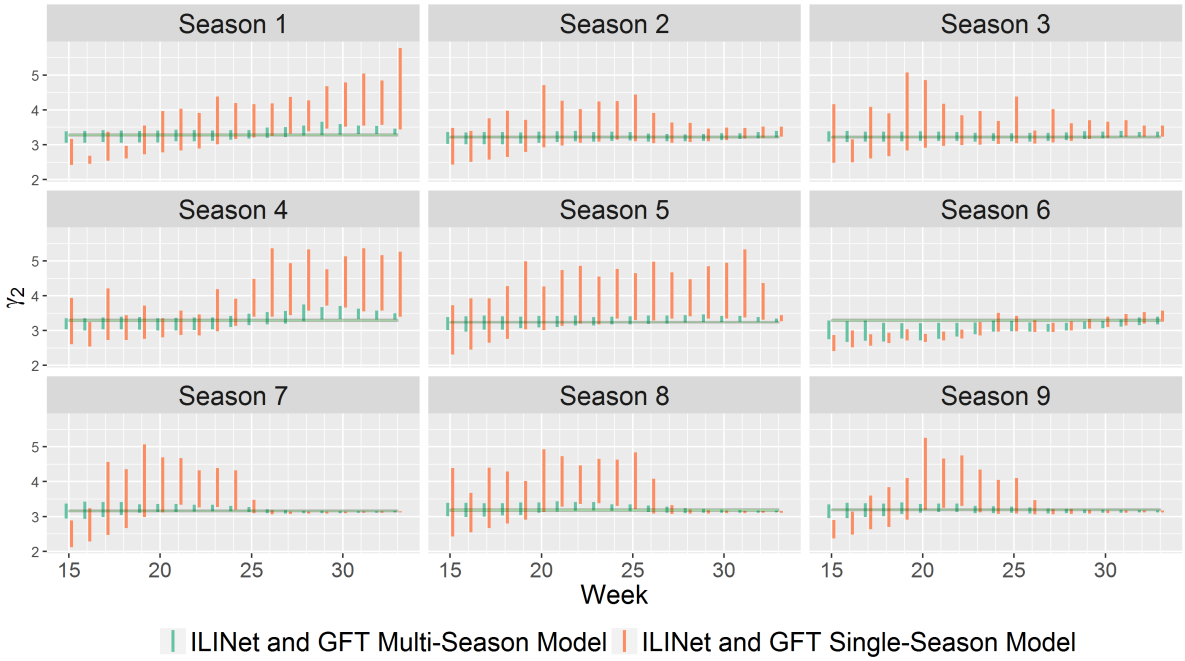

Figure 4.7: Posterior 95% credible intervals for the values of γ_2 given data up to week w for $w \in (15, \ldots, 33)$. Credible intervals are provided for both the Multi-Season and Single-Season models. The light-green band displays the posterior 95% credible interval of the parameter produced by the Multi-Season model given the entire season's data (up to week 47).

In Figure 4.7, we present posterior credible intervals for γ_2 for the Multi-Season and Single-Season models as a function of time. The green band in each facet displays the posterior 95% credible interval produced by the Multi-Season model when provided data up to week 47 (the last week) of that season. This posterior distribution of γ_2 given all the data in a season can be thought of as a target towards which the posterior distributions of each model will move as more data becomes available. We note that the posterior 95% credible intervals from the Single-Season model given data up to week 47 matched up closely with those from the Multi-Season model, and as such are not provided here. Comparing the trajectories of the

posterior distributions across time for the two models, we see that the Multi-Season model trajectories start of narrower and closer to the target posterior that the Single-Season model for all 9 seasons. The Single-Season model produces many incredibly wide intervals, especially for earlier weeks in the season, reflecting its lack of knowledge as to what a reasonable peak week could be.

Figure 4.8 displays posterior credible intervals for the γ_3 parameter which determines the spread of the outbreak curves in each season. The green-band target intervals are somewhat wider for γ_3 than they were for γ_2, reflecting greater uncertainty in the precise spread of the outbreak curve even after the season has ended. Nonetheless, the Multi-Season model still does a noticeably superior job at producing credible intervals that are close to the target interval as compared to the Single-Season model. Additionally, the intervals produced by the Multi-Season model tend to be narrower than those from the Single-Season model, providing more precise inference at earlier points in the season.

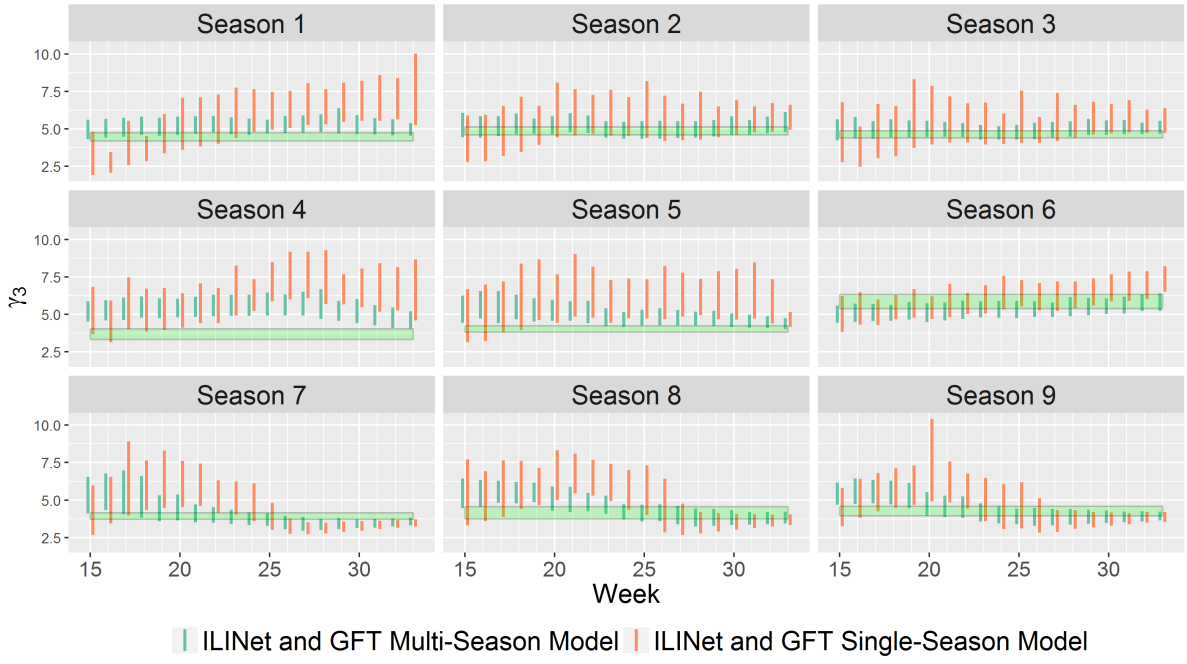

Figure 4.8: Posterior 95% credible intervals for the values of γ_3 given data up to week w for $w \in (15, \ldots, 33)$. Credible intervals are provided for both the Multi-Season and Single-Season models. The light-green band displays the posterior 95% credible interval of the parameter from the Multi-Season model given the entire season's data (up to week 47).

4.6 Discussion

Our model provides a statistical framework through which historical outbreak data can be used to help monitor and predict seasonal influenza epidemic severity. The comparisons in Sections 4.4 and Section 4.5 demonstrate that the inclusion of historical data in a hierarchical modeling framework can have noticeable impact on forecasting and inferential ability. With respect to forecast intervals, we have seen that the use of data from previous seasons tends to provide narrower intervals for the current season, with higher coverage than would be obtained from using the current season's data alone. An even more noticeable benefit to the Multi-Season hierarchical model is its ability to accurately estimate parameters at early points in outbreak seasons. In Sections 4.4.3 and 4.5.2, we have shown that the Multi-Season model is able to provide reasonable distributions for latent outbreak parameters much sooner than the Single-Season model. Knowledge about these parameters at early points in an influenza season could provide a great benefit to health care centers interested in knowing the trajectory of the current season's outbreak curve.

While we use ILINet and GFT data to inform our epidemic predictions, it would be relatively straightforward to include additional sources of epidemic data into the model by specifying the relationship between the new data source and latent influenza severity in the data model. This provides a flexible method for allowing any number of new pieces of information to be included, which could further improve our forecasting ability. Another aspect of the model which could be easily modified is the form of the latent epidemic model. We chose to use a normal kernel to produce an outbreak curve, but other distributional forms (for example, a non-standardized t distribution or a log-normal distribution) could just as easily be implemented, and possibly provide more accurate estimates for certain epidemic seasons. Bimodal epidemic seasons, which could occur due to infections from different influenza strains (Song et al., 2013), could realize a better fit with a mixture distribution of two different outbreak curves, which would present a more accurate representation of the underlying disease biology.

The model could also be expanded to look at multiple locations within a country by providing each location with its own epidemic curve for a given season, which could themselves

come from a season-specific hierarchical distribution. In addition, the model could be used to evaluate sequential correlation across seasonal epidemic curves, by including a mechanism for the epidemic in season s to influence some aspect (spread, peak location, or height) of the epidemic curve in season $s + 1$. This could be especially useful for modeling situations like the 2008 - 2009 and 2009 - 2010 seasons, where A/H1N1 caused a late, secondary peak in the first season, and an extremely early peak in the second season.

CHAPTER 5. Data Visualization and Missing Data Models

In this chapter, we present two shorter research topics. The first topic is a computer application for visualizing data produced by the CDC. The second topic describes a model for increasing the accuracy of disease outbreak data obtained from sentinel surveillance networks by estimating missing influenza observations.

5.1 An Application for Automated CDC Data Visualization

We first present an application named **CDCPlot** that displays publicly available data obtained from the CDC's National Notifiable Diseases Surveillance System (NNDSS). The CDC hosts tables with NNDSS data for a multitude of diseases at https://data.cdc.gov. Data is available at national, regional, and state levels, and is uploaded to the CDC's website on a weekly basis. However, until recently, no free, open-source method of visualizing such data existed. As data are updated weekly, health care researchers interested in tracking the spread of diseases frequently had to go through the cumbersome process of downloading multiple spreadsheets from different urls, combining the data, and producing plots. **CDCPlot** was designed to address that shortcoming by providing an online application that automatically scrapes data from the https://data.cdc.gov website when the data are updated every week. These data are used to produce plots of disease outbreaks over time at multiple different spatial scales. The **CDCPlot** application uses the **shiny** R package web application framework (Chang et al., 2016).

An integral feature of CDCPlot is automated weekly updating. Every Thursday, a script is run which scrapes data for individual diseases from multiple tables on the https://data.cdc.gov/ website. These data for individual diseases are reformatted and combined into a single data file

which contains counts for all diseases, across all locations and time points. For each disease and location, we compute simple alert thresholds, and set alerts when the weekly disease counts surpass these thresholds. Alert thresholds are calculated using a moving average technique, where an alert is sent if the disease count in the current week is more than two running standard deviations above the running mean for that disease. Running means and standard deviations are calculated over the last 10 weeks of data.

The different geographical scales that the application provides can be used to easily identify specific locations that are giving rise to high disease counts. For example, looking at Legionellosis data for the whole country shows a spike in cases in mid-August of 2015. Refining our plots to look at individual regions shows that most of these cases came from the Mid-Atlantic region. Refining further still to look at all locations within the Mid-Atlantic region reveals a large spike in cases in New York City, which was experiencing an outbreak of Legionellosis at that time.

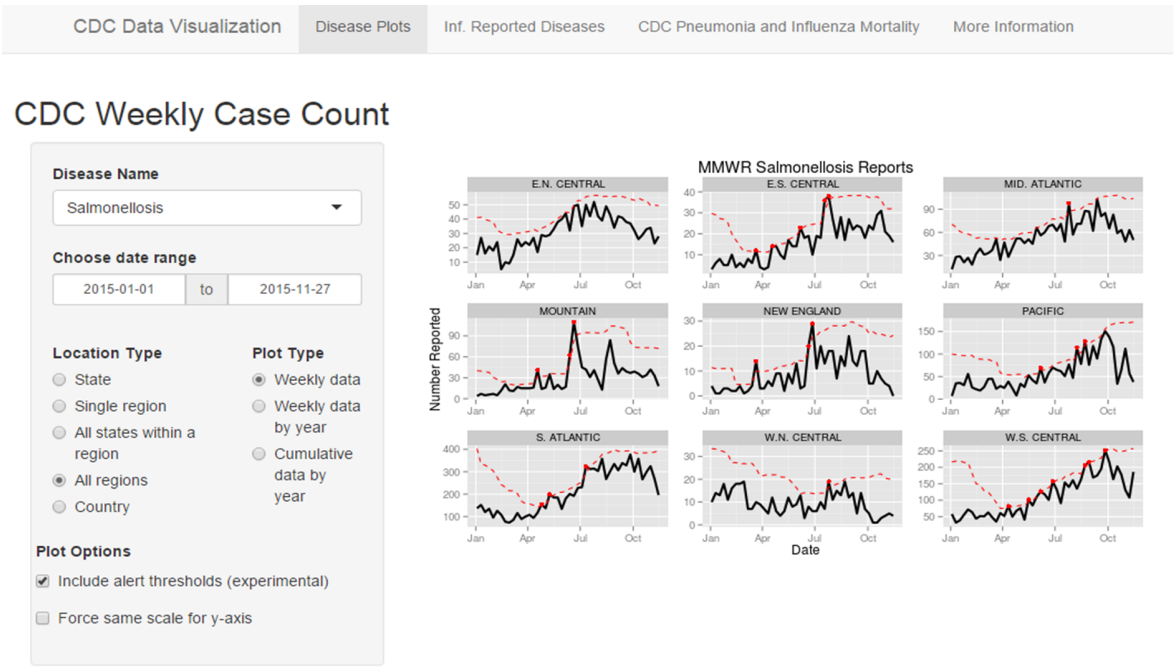

Figure 5.1: The user interface for the **CDCPlot** application. Users can choose the disease, date range, and location to plot. Alert thresholds (red dashed lines) are calculated using a 10-week moving average method, and give an indication of when a disease has been increasing more quickly than might be expected.

CDCPlot can be found at https://gallery.shinyapps.io/CDCPlot/.

5.2 A Bayesian Spatio-temporal Model for Estimating Missing Influenza Counts

ILINet data are derived from a sentinel surveillance network composed of more than 3,300 health care providers. Each health care provider in a given state reports the proportion of patients who visited them and who tested positive for an influenza-like illness to that state's department of public health. These departments, in turn, report their proportions to the CDC, which takes a weighted average of all state-reported proportions to produce the ILINet indicator. Not only is this process time consuming, as ILINet data are usually reported at a lag of one to two weeks, but it is also entirely reliant on submissions from volunteering health care providers. Over the course of an influenza season, many providers will either submit data late, or not at all, for certain weeks. When health care providers submit data past the due date, the ILINet indicator is retroactively recalculated to include this tardy data. This leads to ILINet values for a given week that can change noticeably from when they are first published.

Benefits in ILINet data accuracy can be realized by considering the locations that are missing data in any week. For example, if a health care center had reported very high ILI proportions in each of the last 5 weeks, and then failed to submit a proportion on time this week, it is reasonable to think that missing proportion would likewise be above average.

We present a model to account for missing data in ILINet calculations, resulting in more accurate and timely estimates of influenza severity. Suppose we have S locations reporting weekly ILI data, labeled $s = 1, \ldots, S$. Each location reports both the total number of patients seen in the last week, labeled $N_{s,w}$ for location s and week w, and the number of patients who tested positive for an ILI, labeled $y_{s,w}$. We further define $m_{s,w}$ as an indicator variable denoting whether data are missing ($m_{s,w} = 1$) or present ($m_{s,w} = 0$) at location s for week w. We model:

$$m_{s,w} \sim \text{Bernoulli}(p_{s,w})$$

$$\text{logit}(p_{s,w}) = \alpha_0 + \alpha_1 y_{s,w} + \beta_s$$

The above model relates the probability of providing missing data for location s and week w to the number of patients who tested positive for an ILI that health care provider s had during week w via the α_1 term. Additionally, the β_s term provides a non-week specific offset for certain locations which consistently submit (or don't submit) their data on time. The α_1 term will provide us with information about the number of ILI patients a health care center received even if that center submitted no data for a given week.

Define $x_{s,w}$ as the logit of the latent proportion of the population with an ILI at location s for week w. We model

$$y_{s,w} \sim \text{Binomial}(N_{s,w}, ilogit(x_{s,w}))$$
$$x_{s,w} = g(w, \boldsymbol{\gamma}_s)$$

where $g(w, \boldsymbol{\gamma}_s)$ is defined in Chapter 4 Section 4.3.1. This provides a parametric form for the progress of influenza severity over time at a given location.

Finally, we model

$$N_{s,w} \sim \text{Poisson}(e^{\lambda_{s,w}})$$
$$\lambda_{s,w} \sim N(\lambda_{s,w-1}, \sigma^2_{\lambda_s})$$

Thus the total number of patients a health care center has in a week is modeled as an observation from a Poisson distribution with parameter $e^{\lambda_{s,w}}$, where $\lambda_{s,w}$ follows a random walk over time.

The γ_s parameters and σ_{λ_s} parameters are further modeled hierarchically as in Section 4.3.3. This allows locations to borrow information about parameter values from each other, which has the potential to increase accuracy in a manner similar to that seen with the hierarchical model of Chapter 4.

Note that the above model can be used to estimate ILI proportions $x_{s,w}$ from a location s and week w even if that location did not submit data for that week. To accomplish this, an MCMC algorithm can be run to obtain the posterior distribution of γ_s given all the available

data from this week and previous weeks. Since $x_{s,w}$ is a deterministic function of γ_s, obtaining posterior samples of γ_s will in turn provide samples of $x_{s,w}$.

To assess the ability of the above model to produce more accurate ILINet forecasts, we used sentinel surveillance data from the Kansas Department of Public Health (DPH). The Kansas DPH collects ILI data from 35 health care providers throughout the state, and calculates a state-wide ILI proportion by averaging the proportions received from each of the 35 providers in week w as $ILI_w = \frac{1}{S}\sum_{s=1}^{S} \frac{y_{s,w}}{N_{s,w}}$ for $S = 35$. However, an average of 55% of providers fail to submit data on time in any given week. As such, the state-wide proportion calculated at the official deadline is often based on a much smaller number of providers. As time goes on, more and more providers submit data, causing this proportion to change (sometimes drastically.) This leads to the concept of two state-wide ILI proportions: the "temporary" proportion is the proportion calculated using only those health care centers who have reported their data by the deadline. The "final" proportion is the proportion calculated once all health care providers have submitted their data (which can occur weeks after the deadline has passed.)

Figure 5.2 uses the above missing data model to estimate the "final" state-wide proportion based only on the data received by the deadline. In each facet, the actual "temporary" and "final" state-wide proportions are displayed, along with the posterior distribution for the "final" state-wide proportion produced by the model. These posterior distributions tend to be much closer to the true final proportion than the temporary proportion is. Especially for weeks 22, 23, and 24, which display a large discrepancy between the temporary and final proportions, we find that the model does a very good job of estimating where the final proportion will end up.

Further refinements that would possibly benefit estimation include the use of historical data to help inform seasonal parameters, the use of a latent SIR model instead of the parametric model specified above, and the use of a more sophisticated spatial model, wherein correlations in parameters for locations spatially close to each other could explicitly be modeled. However, even in its current iteration, we feel that it can provide useful estimates of true state-wide ILI prevalence that will perform better than the currently used "temporary proportion".

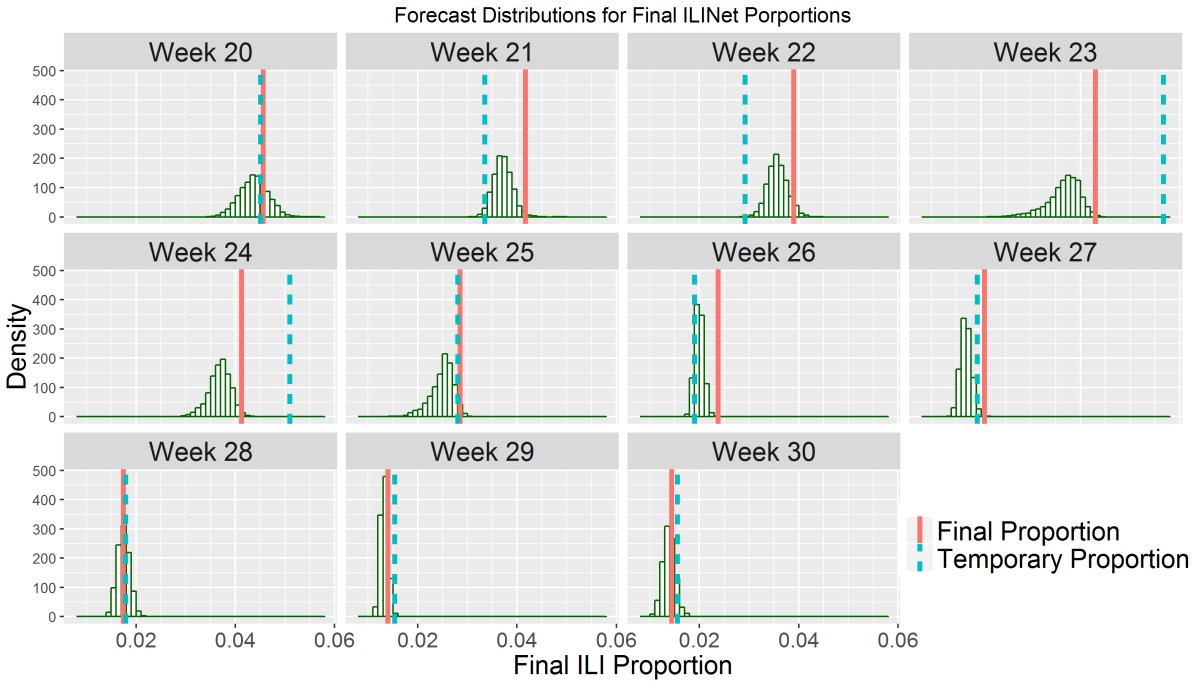

Figure 5.2: Histograms of the posterior distribution of the "final" state-wide ILI proportion for weeks 20 – 30 of the 2014 – 2015 influenza season, where week 0 corresponds to MMWR week 32. Weekly posterior distributions for the "final" state-wide ILI proportion are obtained by averaging over the posterior distributions of the ILI proportions for the 35 individual sites within Kansas, given data up to the submission deadline for that week. The posterior distributions of the "final" ILI proportion tend to be closer to the observed "final" proportions (red lines, calculated as the average ILI proportion once all data have been received) than the observed "temporary" proportions (blue lines, calculated as the average ILI proportion using only data received by the deadline) are.

CHAPTER 6. CONCLUSION

This dissertation has examined a number of concepts related to forecasting the course of disease, and specifically influenza, outbreaks. Chapter 2 provided an overview of Bayesian inferential techniques in the applicable to time-series models in general, with an example of implementing and conducting inference on an SIR model. The **NIMBLE** package provides a flexible framework for both writing models for disease outbreaks and writing algorithms to analyze these models. We anticipate our future outbreak modeling efforts (for example, the hierarchical SIR model of Chapter 4 Section B.2) will be conducted within the **NIMBLE** framework.

Chapters 3 and 4 studied two important modeling techniques that can, and should, be implemented in future outbreak models. Chapter 3 analyzed the benefit of using multiple, possibly-biased data streams in forecasting, and found that the use of more timely, biased data can greatly improve forecast accuracy. We feel as though there is no longer a reason to exclude near-real time data, such as that from Google or Twitter, in any model designed for outbreak forecasting, as careful modeling of the bias in these data sources can ensure more accurate forecasts.

Chapter 4 demonstrated that incorporating information on influenza outbreaks from previous seasons can have a significant benefit on forecasts produced for the current season. While this may not be a realistic modeling method for more uncommon diseases, where appropriate historical data may not be available, it is perfectly suited for seasonal and recurring diseases. Although the hierarchical framework was onyl applied to a parametric latent outbreak curve outbreak model, we believe that the forecasting and inference benefits observed would extend to other models for latent outbreaks, such as a compartmental SIR mode. We furthermore find it surprising that more currently used disease models do not incorporate a temporal hierarchical

framework, as they could stand to benefit from doing so.

Chapter 5 described two additional projects undertaken. The **CDCPlot** application has proved useful to epidemiologists and other health care professionals, enabling quick and up-to-date visualization of commonly used public health data. The missing data model of Chapter 5 Section 5.2 has the potential to increase the accuracy of recently reported ILINet data by taking into account the likely values of missing data.

APPENDIX A. Appendix to Chapter 3

A.1 A Generic Model For Multiple Data Sources

Suppose we have a single latent state of interest, s_t at time t. The evolution equation for s_t is modeled as following a local linear trend model, that is

$$s_t = s_{t-1} + \gamma_{t-1} + w_{s,t} \qquad w_{s,t} \sim N(0, \sigma_s^2)$$
$$\gamma_t = \gamma_{t-1} + w_{\gamma,t} \qquad w_{\gamma,t} \sim N(0, \sigma_\gamma^2)$$

Suppose we also have a total of L unbiased data sources, labeled $U_{l,t}$ at time t for $l = 1, \ldots, L$. Additionally, we have M biased data sources, labeled $B_{m,t}$ at time t where $m = 1, \ldots, M$. For each of our M biased data sources, we introduce a latent additive bias term $\beta_{m,t}$ for $m = 1, \ldots, M$. Each $\beta_{m,t}$ is independently modeled as following a local linear trend model by

$$\beta_{m,t} = \beta_{m,t-1} + \lambda_{m,t-1} + w_{\beta_m,t} \qquad w_{\beta_m,t} \sim N(0, \sigma_{\beta_m}^2)$$
$$\lambda_{m,t} = \lambda_{m,t-1} + w_{\lambda_m,t} \qquad w_{\lambda_m,t} \sim N(0, \sigma_{\lambda_m}^2)$$

Define $x_t = (z_t, \gamma_t, \beta_{1,t}, \lambda_{1,t} \ldots, \beta_{M,t}, \lambda_{M,t})$.

The observation equation for each unbiased data source is $U_{l,t} = z_t + \epsilon_{l,t}$, where $\epsilon_{l,t} \sim N(0, \sigma_{U_l}^2)$. The observation equation for each biased data source is $B_{m,t} = z_t + \beta_{m,t} + \zeta_{m,t}$, where $\zeta_{m,t} \sim N(0, \sigma_{B_m}^2)$. Thus the unbiased data are treated as a noisy estimate of the latent state of interest, while the biased data are a noisy estimate of the latent state of interest plus an evolving bias term.

A primary attraction of including biased data in our model is the assumption that this biased data are received more promptly than the unbiased data. To reflect this, we assume that at time T, unbiased data $U_{l,t}$ are available at times $t = 1, \ldots, T - \delta$ for some $\delta \geq 0$,

and that biased data $B_{m,t}$ are available at times $t = 1, \ldots, T$. Thus, for $t \leq T - \delta$, let $y_t = (U_{1,t}, \ldots, U_{L,t}, B_{1,t}, \ldots, B_{M,t})$. For $T - \delta < t \leq T$, let $y_t = (B_{1,t}, \ldots, B_{M,t})$.

Define

$$K = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad\qquad J = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Our model is then specified as a dynamic linear model by:

$$\begin{aligned} x_t &= G_t x_{t-1} + w_t & w_t &\sim N(0, W_t) \\ y_t &= F_t x_t + v_t & v_t &\sim V(0, V_t) \end{aligned}$$

where

$$G_t = \left(\begin{array}{c|c} [1]_{1\times 1} & [0]_{1\times 2M} \\ \hline [0]_{2M\times 1} & [I]_{M\times M} \otimes K \end{array}\right) \quad W_t = \text{Diag}\left(\sigma_z^2, \sigma_\gamma^2, \sigma_{\beta_1}^2, \sigma_{\lambda_1}^2, \ldots, \sigma_{\beta_M}^2, \sigma_{\lambda_M}^2\right) \qquad t \leq T$$

$$F_t = \left(\begin{array}{c|c} [1]_{L\times 1} & [0]_{L\times 2M} \\ \hline [1]_{2M\times 1} & [I]_{M\times M} \otimes J \end{array}\right) \quad V_t = \text{Diag}\left(\sigma_{U_1}^2, , \ldots, \sigma_{U_L}^2, \sigma_{B_1}^2, \ldots, \sigma_{B_M}^2\right) \qquad t \leq T - \delta$$

$$F_t = \left(\begin{array}{c|c} [1]_{2M\times 1} & [I]_{M\times M} \otimes J \end{array}\right) \quad V_t = \text{Diag}\left(\sigma_{B_1}^2, \ldots, \sigma_{B_M}^2\right) \qquad t \leq T + \delta$$

A.2 A Bootstrap Filter for Chain Binomial SIR Models

We first show that the Chain Binomial SIR model specified in Section 3.3.2 can be written as a state space model. Define $x_t = (S_t, I_t, R_t, \Delta_{1,t}, \Delta_{2,t})$ for any $t \geq 0$. Note that for any $t \geq 1$, we can write

$$f_\lambda(x_t|x_{t-1}) = f_\lambda(S_t, I_t, R_t, \Delta_{1,t}, \Delta_{2,t}|x_{t-1}) \tag{A.1}$$

$$= f_\lambda(S_t, I_t, R_t|\Delta_{1,t}, \Delta_{2,t}, x_{t-1}) f_\lambda(\Delta_{1,t}, \Delta_{2,t}|x_{t-1}) \tag{A.2}$$

In Equation A.2, note that

$$f_\lambda(S_t, I_t, R_t|\Delta_{1,t}, \Delta_{2,t}, x_{t-1}) = \delta(S_t - (S_{t-1} - \Delta_{1,t}))\times$$
$$\delta(I_t - (I_{t-1} + \Delta_{1,t} - \Delta_{2,t}))\delta(R_t - (R_{t-1} + \Delta_{2,t}))$$

where $\delta(x - y)$ denotes the Dirac delta function. Thus the transition equation for the latent S_t, I_t, and R_t states can be considered deterministic conditioned upon $\delta_{1,t}$, $\delta_{2,t}$, and x_{t-1}.

Also, in Equation A.2,

$$f_\lambda(\Delta_{1,t}, \Delta_{2,t}|x_{t-1}) = f_\lambda(\Delta_{1,t}|x_{t-1}) f_\lambda(\Delta_{2,t}|x_{t-1})$$

which will be the product of two binomial densities. We further set $\delta_{1,0} = 0$ and $\delta_{2,0} = 0$ and define $S_0 = N - I_0$, $I_0 \sim Binom(N, p_0)$, and $R_0 = 0$. Thus the Chain Binomial model can be written as a state space model with the above transition equations.

Below, we provide a Bootstrap filtering algorithm (Gordon et al., 1993) specified for the Chain Binomial SIR Model model. The Bootstrap filter requires a choice of proposal distribution q for the latent states x, denoted $q(x_t|x_{t-1}, y_t)$. For ease of computation, we choose $q(x_t|x_{t-1}, y_t) = p(x_t|x_{t-1})$. Included in the bootstrap algorithm is an option to resample particles at a given time point only if the Effective Sample Size, or ESS, is too low (Smith et al., 2001). The ESS is calculated as $ESS = \frac{1}{\sum_{p=1}^{P} \pi_t^{(p)}}$. Specifically, if $\frac{ESS}{P} < \tau$, where P is the total number of particles used, a resampling step is performed. We use a τ value of 0.9.

Algorithm 6 Bootstrap Filter for a Chain Binomial SIR Model

for p in $1:P$ **do**
  Generate $(S_0, I_0, R_0)^{(p)} \sim p_\lambda(S_0, I_0, R_0)$
  Set $\pi_0^{(p)} = \frac{1}{P}$
end for
for t in $1:T$ **do**
  for p in $1:P$ **do**
    Generate $(\Delta_{1,t}, \Delta_{2,t})^{(p)} \sim p_\lambda(\Delta_{1,t}, \Delta_{2,t}|x_{t-1}^{(p)})$
    Generate $(S_t, I_t, R_t)^{(p)} = p_\lambda(S_t, I_t, R_t|(\Delta_{1,t}, \Delta_{2,t}, x_{t-1})^{(p)})$
    Set $x_t^{(p)} = (S_t, I_t, R_t, \Delta_{1,t}, \Delta_{2,t})^{(p)}$
    Calculate $w_t^{(p)} = p(y_t|x_t^{(p)})\pi_{t-1}^{(p)}$
    Normalize $w_t^{(p)}$ as $\pi_t^{(p)} = \frac{w_t^{(p)}}{\sum_{i=1}^P w_t^{(i)}}$
    if $\frac{ESS}{P} < \tau$ **then**
      Sample $x_t^{(p)} \sim \sum_{i=1}^P \pi_t^{(i)} \delta(x - \tilde{x}_t^{(i)})$
      Set $\pi_t^{(i)} = \frac{1}{P}$ for $i = 1, \ldots, P$.
    else
      Set $x_t^{(p)} = \tilde{x}_t^{(p)}$
    end if
  end for
  Calculate $\tilde{p}(y_{t|1:t-1}) = \frac{1}{P} \sum_{p=1}^P w_t^{(p)}$
end for

A.3 Posterior Distributions of DLM Model Parameters

Posterior distributions displayed in this section are obtained from modeling with real ILINet and GFT data. Models were given ILINet data up to week 30 and GFT data up to week 32 to obtain posterior distributions of model parameters after the outbreak peak had been reached in each season. As remarked in Section 3.6.1, parameter posterior estimates for σ_I and σ_G are variable from week to week in the "ILINet and GFT Unbiased Model", so the posterior distributions seen in this section do not necessarily reflect those obtained in other weeks.

Figure A.1 shows posterior distributions of σ_z and σ_I for all 9 seasons and 3 models. The posterior distributions of σ_z are consistent across models, with season 6 providing a posterior distribution shifted slightly to the left of the other seasons. The posterior distributions of σ_I again consistent between the "ILINet and GFT Bias Model" and the "ILINet Only Model". However, the "ILINet and GFT Unbiased Model" produces noticeably higher distributions for σ_I in seasons 6, 7, and 8. Since the unbiased model does not account for the bias in GFT data,

it tends to treat ILINet data as very noisy in seasons where bias is present. Additionally, the unbiased model produces a bimodal posterior for σ_I in season 8. The bi-modality arises from the unbiased model's difficulty in partitioning variance between ILINet and GFT data.

Figure A.2 shows posterior distributions of σ_G and σ_β for all 9 seasons. The posterior distributions of σ_G are similar between the two models which include GFT data until season 8. In season 8, the unbiased model produces a bimodal curve, reminiscent of that seen in Figure A.2. In season 9, the GFT model produces a much higher estimate for σ_G than the biased model. Note that the biased model, in turn, produces a somewhat higher estimate for σ_β in season 9, demonstrating that the biased model was able to partition the variance of the GFT data into σ_G and σ_β, while the unbiased model lumped all GFT variance into σ_G.

A.4 Posterior Distributions of SIR Model Parameters

Posterior distributions displayed in this section are obtained from modeling with real ILINet and GFT data. Models were given ILINet data up to week 30 and GFT data up to week 32 to obtain posterior distributions of model parameters after the outbreak peak had been reached in each season.

Figure A.3 shows the posterior distributions of the β and γ parameters for each season and each SIR model. Parameter posterior distributions are consistent between the models, with the "ILINet Only Model" tending to produce slightly less peaked posterior distributions.

Figure A.4 shows the posterior distributions of the σ_I parameter for each season and each SIR model. We see a discrepancy between the "ILINet and GFT Unbiased Model" and the other two models in seasons 6 and 7, the two seasons which exhibited the most bias in GFT data. Again, the unbiased model is unable to correctly assign variance between σ_I and σ_G. For seasons 6 and 7, the unbiased model treats ILINet data as highly variable, and GFT data as relatively more accurate.

Figure A.5 shows the posterior distributions of the σ_G and ρ parameters for each season and each SIR model which used GFT data. Posterior distributions of σ_G match up well between the two models. We note that two of the seasons for which ρ has a very narrow distribution centered near one are the season in which GFT data displayed a large amount of bias, confirming that

the ρ parameter is able to detect GFT bias within a season.

A.5 Data Source Information

To fit our models we use ILINet data through the 2014 – 2015 season, which can be obtained from U.S. Centers for Disease Control and Prevention (2016a). We use the **cdcfluview** R package (Rudis, 2015) to automate the retrieval of ILINet data.

Our models also use GFT data through the 2014 – 2015 influenza season, which were obtained from Google (2016). Google stopped making its GFT data publicly available after the 2014–2015 season.

MMWR weeks are used by the Morbidity and Mortality Weekly Report as a standard index for weeks corresponding to epidemiological data. MMWR weeks run from Sunday through Saturday, with the first MMWR week for a given year being the first week with at least four days within that year (U.S. Centers for Disease Control and Prevention, 2016b). We used the **RMMWRweek** R package (Niemi, 2015) to convert between calendar dates and MMWR weeks.

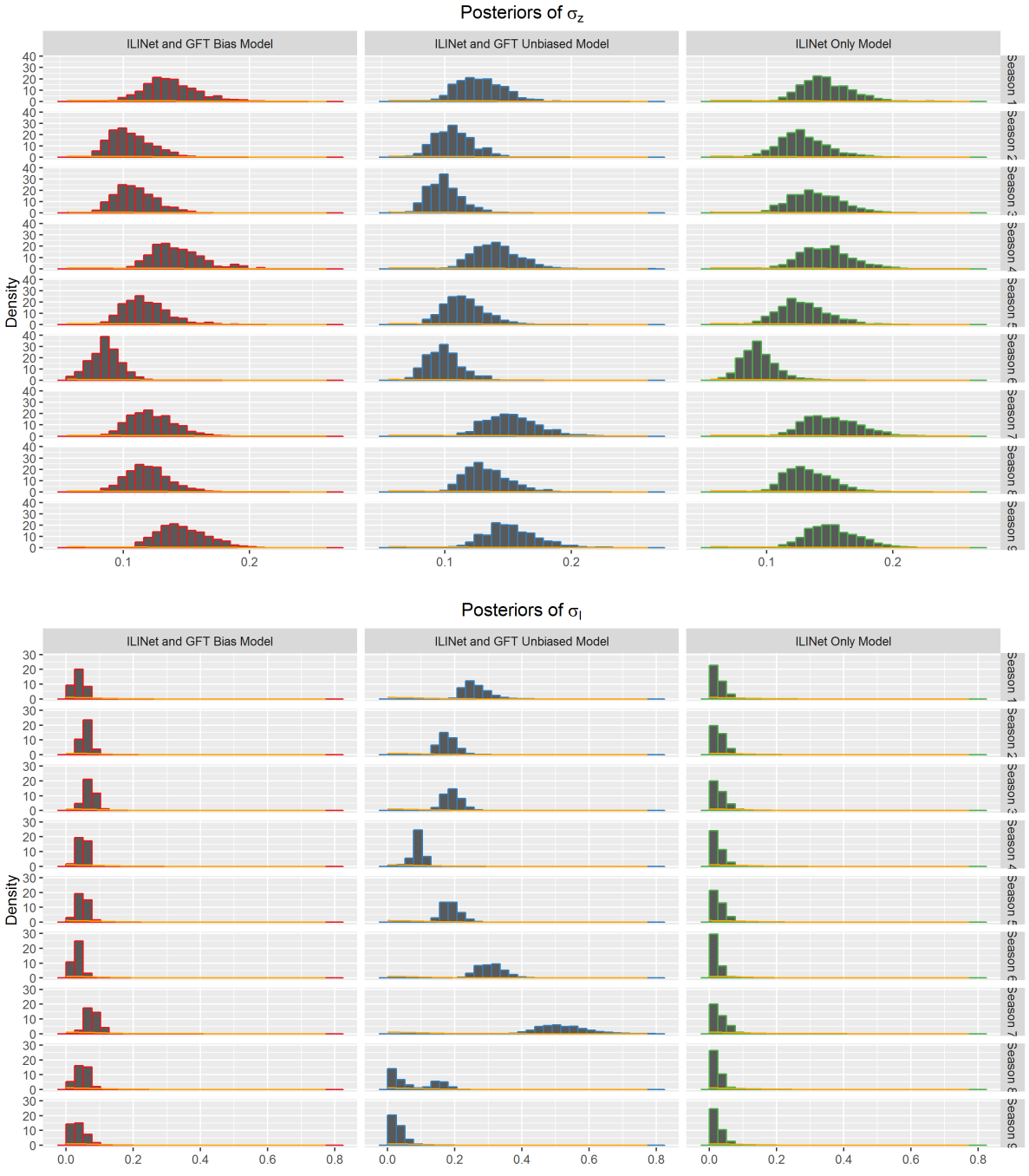

Figure A.1: Posterior distributions of σ_z and σ_I parameters for the three DLM models, faceted by season. Prior distributions (orange curves) are included.

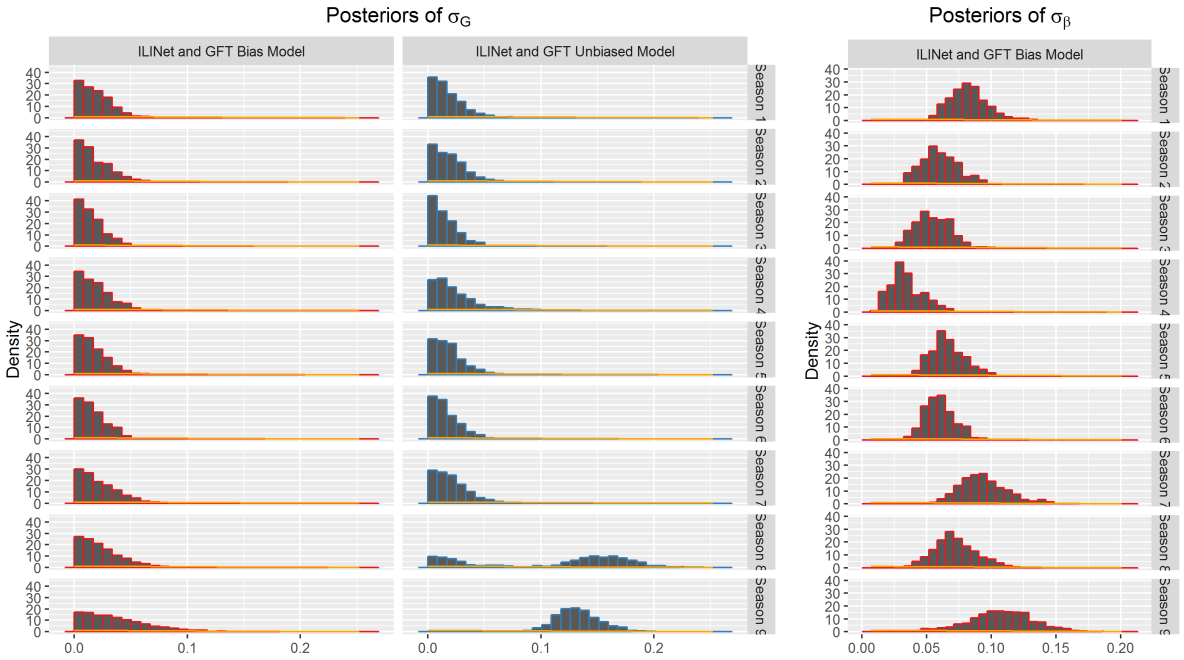

Figure A.2: Posterior distributions of σ_G and σ_β parameters for the DLM models, faceted by season. Prior distributions (orange curves) are included.

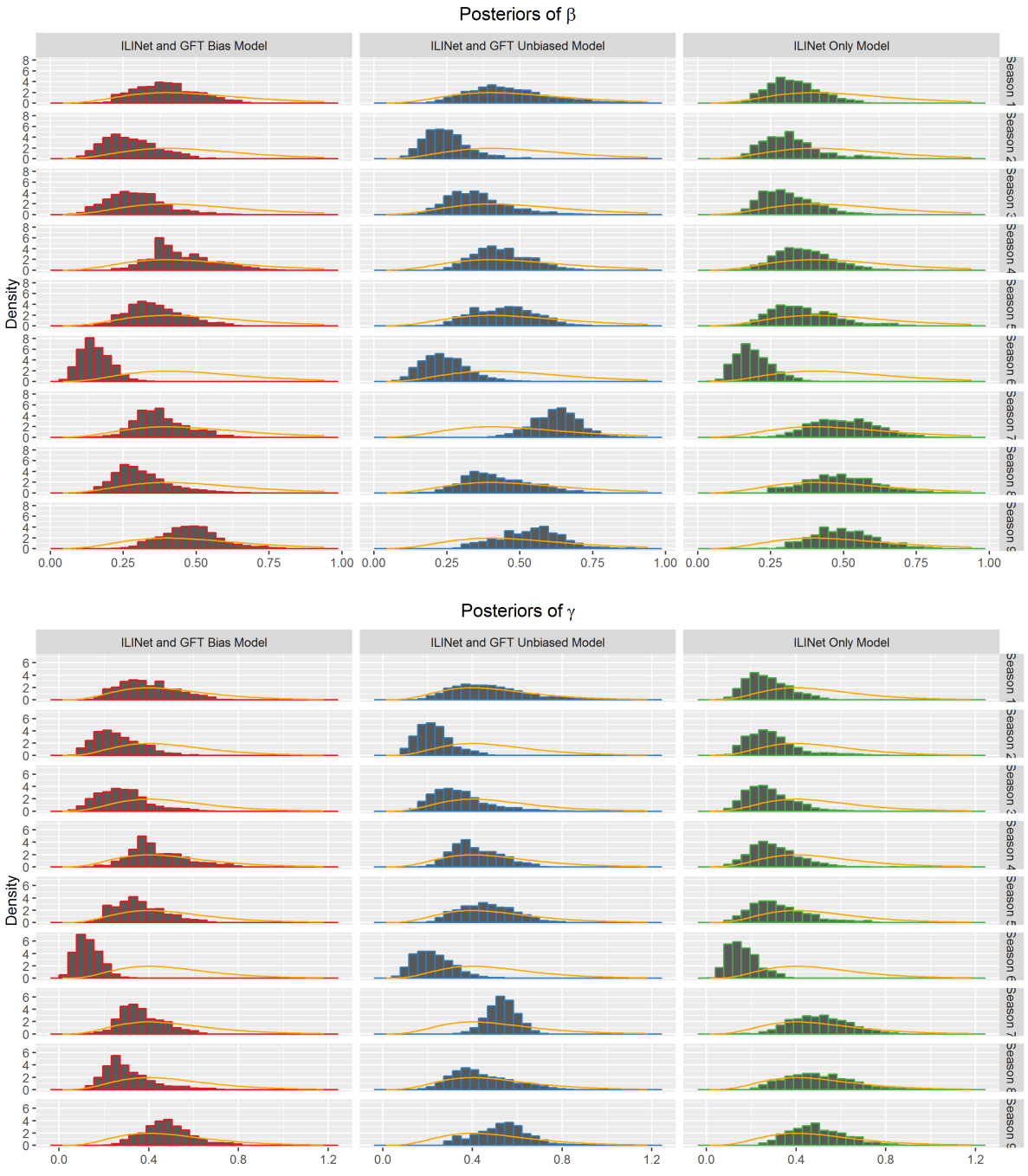

Figure A.3: Posterior distributions of β and γ parameters for SIR models, faceted by season. Prior distributions (orange curves) are included.

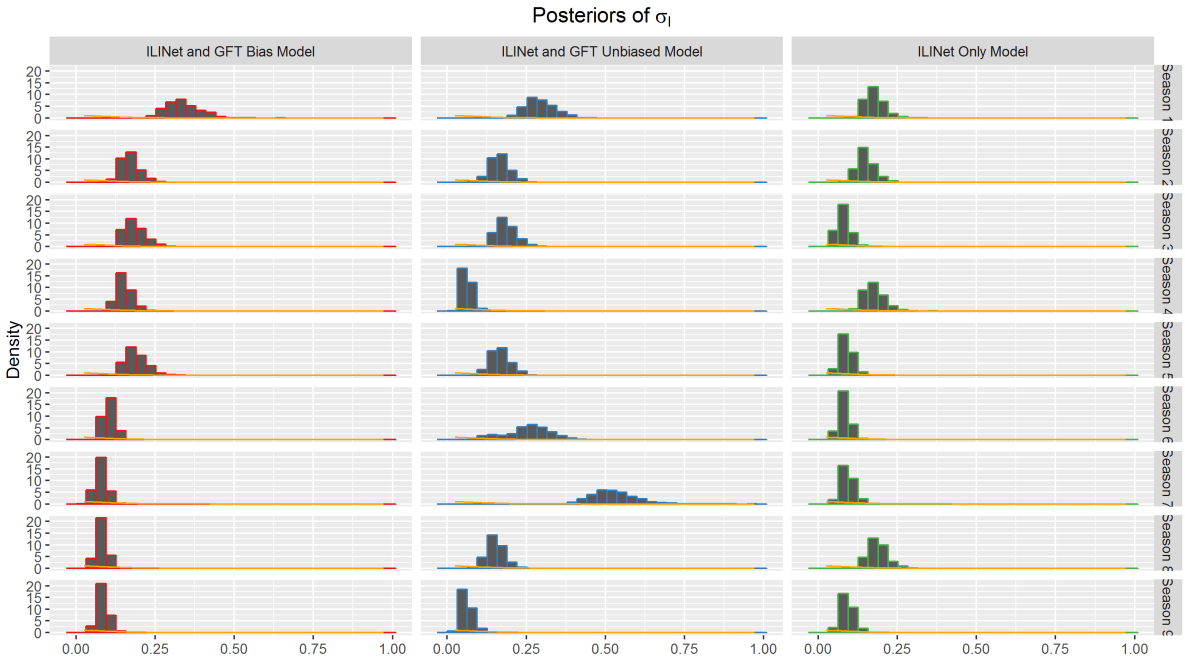

Figure A.4: Posterior distributions of the σ_I parameter for the SIR models, faceted by season. Prior distributions (orange curves) are included.

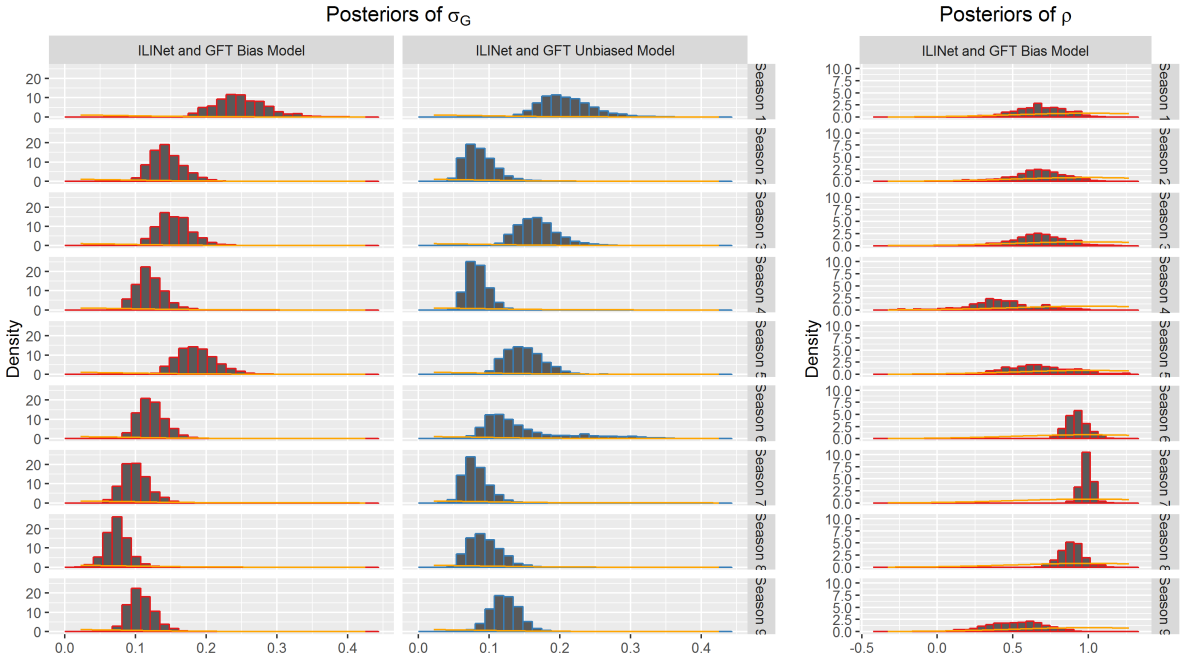

Figure A.5: Posterior distributions of σ_G and ρ parameters for SIR models, faceted by season. Prior distributions (orange curves) are included.

APPENDIX B. Appendix to Chapter 4

B.1 MCMC Settings and Convergence

The following settings were used for the hyperparameters for our data model:

$$\theta_I \sim t_5(-1, 10), \qquad \lambda_I \sim Ca^+(0, .4)$$

$$\theta_G \sim t_5(-1, 10), \qquad \lambda_G \sim Ca^+(0, .4)$$

The following settings were used for the hyperparameters for our latent model:

$$\mu_1 \sim t_5(0, 10), \qquad \mu_2 \sim t_5(3, 10)$$

$$\mu_3 \sim t_5(1.5, 10), \qquad \mu_4 \sim t_5(-4, 10)$$

$$\kappa_i \sim Ca^+(0, 1)$$

To forecast epidemic severity, we first ran an MCMC algorithm using only the first eight seasons of real ILINet and GFT data (from 2003 – 2004 to 2013 – 2014). For this MCMC algorithm, we ran a single chain, and used the median of the prior distributions as starting values for our parameters. We set a long burn-in of 100,000 iterations, and then sampled every 10th iteration after that, until 10,000 iterations were sampled. Geweke diagnostics were run for the posterior samples using a Bonferroni adjustment for multiple hypotheses, and no significant departure from convergence was found.

For subsequent MCMC algorithms, such as those used for forecasting severity for individual weeks in the 2014 – 2015 season, we again ran a single chain. For these chains we used the medians of the posterior distributions obtained from our initial, longer MCMC run as initial values for our parameters. Using such informative starting values means these algorithms were initialized at a state that was likely close to convergence, allowing us to use a short burn-in period for a much faster run time. For these MCMC algorithms, we used a burn-in of

10,000 iterations, and then sampled every 10th iteration after that, until 10,000 iterations were sampled. To diagnose convergence, we again ran Geweke diagnostics for the posterior samples using a Bonferroni adjustment.

B.2 Hierarchical SIR Model and Hierarchical PMCMC Inference

In Section B.2.1, we introduce a hierarchical SIR model which can be used to conduct inference on disease outbreaks which span multiple seasons. Each season is represented by an individual SIR model, the parameters of which are related by higher-level distributions. Then, in Section B.2.2, we provide a hierarchical PMCMC algorithm for conducting inference on the hierarchical SIR model.

B.2.1 Latent Epidemic Model

Suppose we want to model S seasons, each with W weeks. Let $S_{s,w}, I_{s,w}$, and $R_{s,w}$ be the susceptible, infectious, and recovered compartments respectively in season s and week w. We use the Chain Binomial model of Chapter 3 Section 3.3.2 to model movements between compartments in season s for weeks $w = 1, \ldots, W$ as

$$\begin{aligned}
S_{s,w} &= S_{s,w-1} - \Delta_{1,s,w} \\
I_{s,w} &= I_{s,w-1} + \Delta_{1,s,w} - \Delta_{2,s,w} \\
R_{s,w} &= R_{s,w-1} + \Delta_{2,s,w} \\
\Delta_{1,s,w} &\sim Binom(S_{s,w-1}, 1 - e^{-\beta_s \frac{I_{s,w-1}}{N}}) \\
\Delta_{2,s,w} &\sim Binom(I_{s,w-1}, 1 - e^{-\gamma_s})
\end{aligned}$$

We additionally specify initial states for season s by

$$\begin{aligned}
S_{s,0} &= N - I_{s,0} \\
I_{s,0} &\sim Binom(N, p_{0,s}) \\
R_{s,0} &= 0
\end{aligned}$$

Hierarchical distributions are employed fr the γ, β, and p_0 parameters, allowing us to share information across seasons. Let $\lambda_s = (log(\beta_s), log(\gamma_s), logit(p_{0,s}))$. We model

$$\lambda_s \sim MVN(\mu, \Sigma)$$

where $MVN(\mu, \Sigma)$ denotes the multivariate normal distribution with mean vector μ and covariance matrix Σ. Drawing season specific outbreak parameters from a higher-level distribution will allow forecasts and inference for the current season to benefit from the data of previous seasons. As seen in Section 4.5, this can lead to much improved results, especially if parameter inference is of interest.

We use ILINet and GFT data to inform our model, again as in Chapter 3 Section 3.3.2. Let $ILI_{s,w}$ and $GFT_{s,w}$ denote the logit of ILINet and GFT data for season s and week w. We model

$$\begin{aligned} ILI_{s,w} &= logit\left(\frac{I_{s,w}}{N}\right) + \epsilon_{ILI,s,w} \\ GFT_{s,w} &= logit\left(\frac{I_{s,w}}{N}\right) + \rho(GFT_{s,w-1} - logit\left(\frac{I_{w-1}}{N}\right)) + \epsilon_{GFT,s,w} \end{aligned}$$

where $\epsilon_{ILI,s,w} \overset{ind}{\sim} N(0, \sigma^2_{ILI,s})$ and $\epsilon_{GFT,s,w} \overset{ind}{\sim} N(0, \sigma^2_{GFT,s})$, and where I_t is the latent number of people in the I compartment at time t, and N is our total population size. Thus data error terms are distributed with season and data stream specific variances. We further model these season specific variances as coming from a higher level distribution via

$$\begin{aligned} log(\sigma_{ILI,s}) &\sim N(\theta_{ILI}, \kappa^2_{ILI}) \\ log(\sigma_{GFT,s}) &\sim N(\theta_{GFT}, \kappa^2_{GFT}) \end{aligned}$$

Thus the season specific data variances for the year we are interested in forecasting for can be informed by data variances from previous seasons. We remark that the data model above

is highly flexible, as additional data streams can be incorporated or removed as they become available for certain seasons.

B.2.2 A Hierarchical PMCMC Algorithm

To conduct inference on our hierarchical SIR model, we first reparameterize our seasons specific λ_s parameters as:

$$\lambda_s = \mu + A\mathbf{z}_s$$

where A is the Cholesky decomposition of Σ, so that $AA^T = \Sigma$, and where $\mathbf{z}$ is a vector of independent $N(0,1)$ random variables.

We likewise reparameterize

$$log(\sigma_{ILI,s}) = \theta_{ILI} + \kappa_{ILI} w_s$$
$$log(\sigma_{ILI,s}) = \theta_{GFT} + \kappa_{GFT} w_s$$

where w_s is an independent $N(0,1)$ random variable.

Reparameterizing our model in this way allows us to define an MCMC proposal distribution for $\mu, \Sigma, \mathbf{z}, \theta, \kappa$, and $\mathbf{w}$, where $\mathbf{z}$ is a vector of all $\mathbf{z}_s$, $\mathbf{w}$ is a vector of all w_s, and θ and κ are vectors of the means and standard deviations of our data distributions. Note that all of the parameters in our proposal distribution are independent from each other, which will facilitate MCMC mixing.

We use an MVN proposal distribution for the above parameters. Algorithm 7 defines a hierarchical PMCMC algorithm which can be used to conduct inference.

Algorithm 7 PMMH Algorithm

for iteration $i \geq 0$ **do**

  Sample $(\mu, \Sigma, \mathbf{z}, \theta, \kappa, \mathbf{w})^* \sim q((\mu, \Sigma, \mathbf{z}, \theta, \kappa, \mathbf{w})^* | (\mu, \Sigma, \mathbf{z}, \theta, \kappa, \mathbf{w})^{(i-1)})$

  For each season $s = 1, \ldots, S$, calculate $\lambda_s^* = f(\mu^*, \Sigma^*, \mathbf{z}_s^*)$ and $\sigma_s^* = g(\theta^*, \kappa^*, w_s^*)$.

  For each season $s = 1, \ldots, S$, run an SMC algorithm to get a sample $x_{s,1:T}^* \sim p(x_{s,1:T} | ILI_{s,1:T}, \lambda_s^*, \sigma_s^*)$ and a marginal likelihood estimate $\hat{p}(ILI_{s,1:T} | \lambda_s^*, \sigma_s^*)$

  Compute the Metropolis Hastings ratio p^*

  Generate $r \sim unif(0, 1)$

  if $p^* > r$ **then**

    Set $(\mu, \Sigma, \mathbf{z}, \theta, \kappa, \mathbf{w})^{(i)} = (\mu, \Sigma, \mathbf{z}, \theta, \kappa, \mathbf{w})^*$ and $x_{1:S,1:T}^{(i)} = x_{1:S,1:T}^*$

  else

    Set $(\mu, \Sigma, \mathbf{z}, \theta, \kappa, \mathbf{w})^{(i)} = (\mu, \Sigma, \mathbf{z}, \theta, \kappa, \mathbf{w})^{(i-1)}$ and $x_{1:S,1:T}^{(i)} = x_{1:S,1:T}^{(i-1)}$

  end if

end for

The above hierarchical PMCMC algorithm calls for an SMC algorithm to be used to obtain samples of the latent compartment states and estimates of the likelihood at each iteration. The Bootstrap Filter specified in Chapter 3 Appendix Section A.2 could be used for this purpose, or a more sophisticated algorithm such as the auxiliary particle filter could be employed.

B.2.3 Demonstration of Hierarchical PMCMC Inference

To demonstrate the application of the hierarchical PMCMC algorithm to the hierarchical SIR model, inference was conducted using the complete set of ILINet and GFT data from seasons 1 through 9. A single MCMC chain was run for 60,000 iterations. The first 20,000 iterations were discarded, and the remaining iterations were thinned by 2, resulting in 20,000 samples from the posterior distribution of both our latent SIR parameters and our data parameters. The hierarchical MCMC algorithm was written within the **NIMBLE** R package. In Figure B.1, we provide trace plots of the γ_s and β_s parameters for each season s.

Although we have not conducted forecasting using the hierarchical SIR model, we anticipate

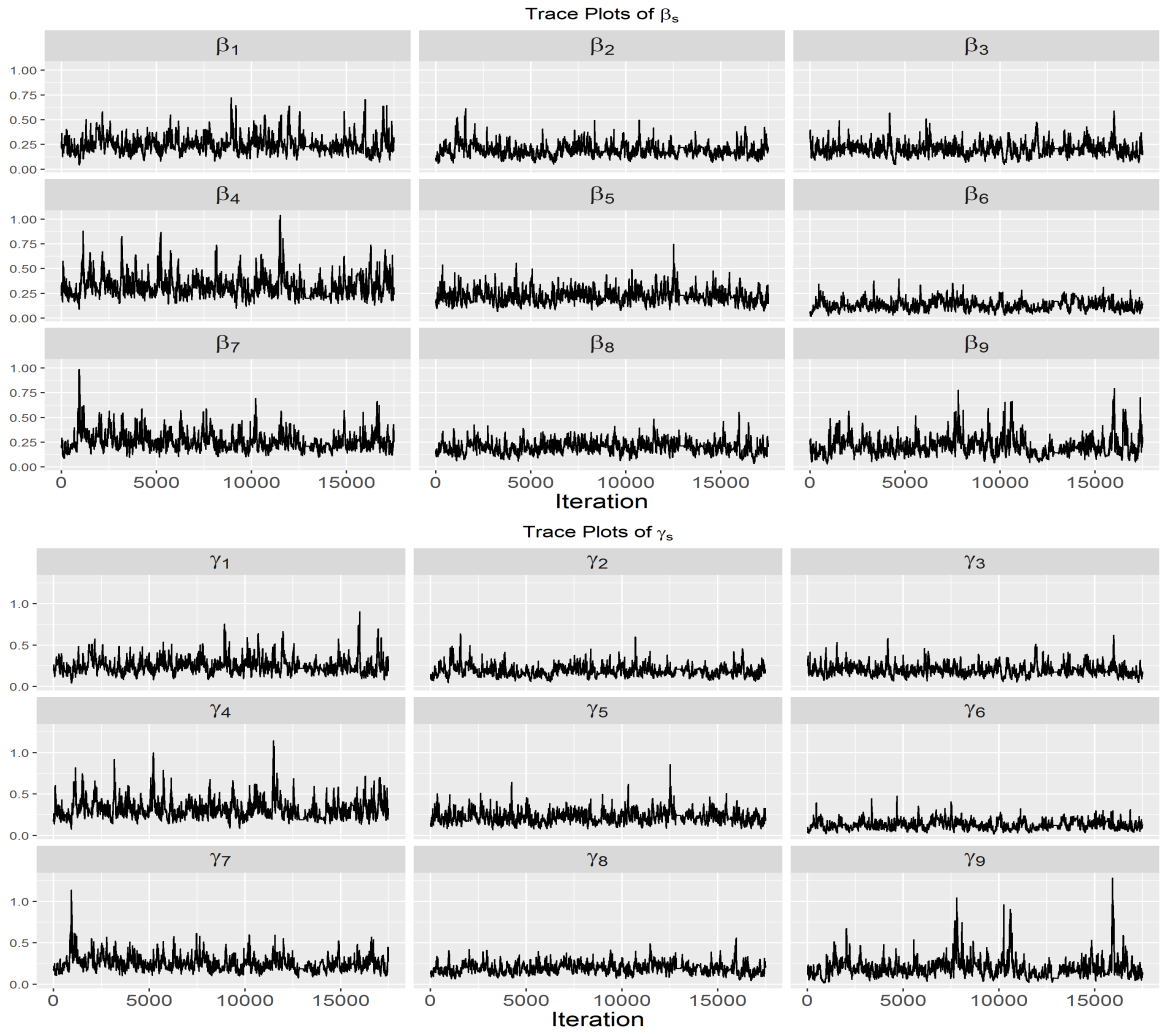

Figure B.1: Trace plots for seasonal γ_s and β_s parameters obtained using a hierarchical PMCMC algorithm for seasons 1 through 9.

that it would perform quite well, based on both the success of the single-season SIR model in Chapter 3 and the demonstrated benefits of hierarchical modeling seen in Chapter 4. We intend to examine the model's forecasting ability in the near future.

Bibliography

Aberson, S. D. (2003). Targeted observations to improve operational tropical cyclone track forecast guidance. *Monthly weather review*, 131(8).

Aerts, S., Lambrechts, D., Maity, S., Van Loo, P., Coessens, B., De Smet, F., Tranchevent, L.-C., De Moor, B., Marynen, P., Hassan, B., et al. (2006). Gene prioritization through genomic data fusion. *Nature biotechnology*, 24(5):537–544.

Andersson, H. and Britton, T. (2012). *Stochastic epidemic models and their statistical analysis*, volume 151. Springer Science & Business Media.

Andrieu, C., Doucet, A., and Holenstein, R. (2010). Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342.

Barboza, L., Li, B., Tingley, M. P., and Viens, F. G. (2014). Reconstructing past temperatures from natural proxies and estimated climate forcings using short- and long-memory models. *Ann. Appl. Stat.*, 8(4):1966–2001.

Berrocal, V. J., Gelfand, A. E., and Holland, D. M. (2010). A bivariate spaceâĂŞtime downscaler under space and time misalignment. *Ann. Appl. Stat.*, 4(4):1942–1975.

Black, A. J. and McKane, A. J. (2010). Stochastic amplification in an epidemic model with seasonal forcing. *Journal of Theoretical Biology*, 267(1):85–94.

Brammer, L., Blanton, L., Epperson, S., Mustaquim, D., Bishop, A., Kniss, K., Dhara, R., Nowell, M., Kamimoto, L., and Finelli, L. (2011). Surveillance for influenza during the 2009 influenza a (h1n1) pandemic -âĂŞ united states, april 2009âĂŞmarch 2010. *Clinical Infectious Diseases*, 52(suppl 1):S27–S35.

Brauer, F. (2008). *Mathematical Epidemiology*, chapter Compartmental Models in Epidemiology, pages 19–79. Springer Berlin Heidelberg, Berlin, Heidelberg.

Carlin, B. P., Polson, N. G., and Stoffer, D. S. (1992). A monte carlo approach to nonnormal and nonlinear state-space modeling. *Journal of the American Statistical Association*, 87(418):493–500.

Carpenter, J., Clifford, P., and Fearnhead, P. (1999). Improved particle filter for nonlinear problems. In *Radar, Sonar and Navigation, IEE Proceedings-*, volume 146, pages 2–7. IET.

Carter, C. K. and Kohn, R. (1994). On gibbs sampling for state space models. *Biometrika*, 81(3):541–553.

Cauchemez, S., Bhattarai, A., Marchbanks, T. L., Fagan, R. P., Ostroff, S., Ferguson, N. M., Swerdlow, D., Sodha, S. V., Moll, M. E., Angulo, F. J., et al. (2011). Role of social networks in shaping disease transmission during a community outbreak of 2009 h1n1 pandemic influenza. *Proceedings of the National Academy of Sciences*, 108(7):2825–2830.

CDC (2016). Overview of Influenza Surveillance in the United States. http://www.cdc.gov/flu/pdf/weekly/overview.pdf. [Online; accessed 19-February-2016].

Chang, W., Cheng, J., Allaire, J., Xie, Y., and McPherson, J. (2016). *shiny: Web Application Framework for R*. R package version 0.13.0.

Chowell, G., Bettencourt, L. M., Castillo-Chavez, C., and Hyman, J. M. (2009). *Mathematical and statistical estimation approaches in epidemiology*. Springer.

Chretien, J.-P., George, D., Shaman, J., Chitale, R. A., and McKenzie, F. E. (2014). Influenza forecasting in human populations: A scoping review. *PLoS ONE*, 9(4):e94130.

Conesa, D., Martínez-Beneito, M., Amorós, R., and López-Quílez, A. (2015). Bayesian hierarchical poisson models with a hidden markov structure for the detection of influenza epidemic outbreaks. *Statistical methods in medical research*, 24(2):206–223.

Creal, D. (2012). A survey of sequential monte carlo methods for economics and finance. *Econometric Reviews*, 31(3):245–296.

de Valpine, P., Turek, D., Paciorek, C. J., Anderson-Bergman, C., Temple Lang, D., and Bodik, R. (2015). Programming with models: writing statistical algorithms for general model structures with NIMBLE. *ArXiv e-prints*.

Doucet, A., De Freitas, N., and Gordon, N. (2001). An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer.

Doucet, A., Godsill, S., and Andrieu, C. (2000). On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, 10(3):197–208.

Doucet, A. and Johansen, A. M. A tutorial on particle filtering and smoothing: Fifteen years later.

Douglas, A. P., Breipohl, A. M., Lee, F. N., and Adapa, R. (1998). Risk due to load forecast uncertainty in short term power system planning. *Power Systems, IEEE Transactions on*, 13(4):1493–1499.

Dugas, A. F., Jalalpour, M., Gel, Y., Levin, S., Torcaso, F., Igusa, T., and Rothman, R. E. (2013). Influenza forecasting with google flu trends. *PLoS ONE*, 8(2):e56176.

Efron, B. and Tibshirani, R. (1986). Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75.

Epperson, S., Blanton, L., Kniss, K., Mustaquim, D., Steffens, C., Wallis, T., Dhara, R., Leon, M., Perez, A., . Chaves, S., Abd, A., Gubareva, L., Xu, X., Villanueva, J., Bresee, J., Cox, N., Finelli, L., and Brammer, L. (2014). Influenza activity - united states, 2013-14 season and composition of the 2014-15 influenza vaccines. *Morbidity and Mortality Weekly Report*, 63(22):483–490.

Evensen, G. (2003). The ensemble kalman filter: theoretical formulation and practical implementation. *Ocean Dynamics*, 53:343–367.

Frühwirth-Schnatter, S. (1994). Data augmentation and dynamic linear models. *Journal of time series analysis*, 15(2):183–202.

Gelman, A. et al. (2006). Prior distributions for variance parameters in hierarchical models (comment on article by browne and draper). *Bayesian analysis*, 1(3):515–534.

Gillijns, S., Mendoza, O. B., Chandrasekar, J., De Moor, B., Bernstein, D., and Ridley, A. (2006). What is the ensemble kalman filter and how well does it work? In *American Control Conference, 2006*, pages 6–pp. IEEE.

Ginsberg, J., Mohebbi, M. H., Patel, R. S., Brammer, L., Smolinski, M. S., and Brilliant, L. (2009). Detecting influenza epidemics using search engine query data. *Nature*, 457(7232):1012–1014.

Goldstein, E., Cobey, S., Takahashi, S., Miller, J. C., and Lipsitch, M. (2011). Predicting the epidemic sizes of influenza a/h1n1, a/h3n2, and b: A statistical method. *PLoS Med*, 8(7):e1001051.

Google (2016). Google Flu Trends. https://www.google.org/flutrends/about/data/flu/us/data.txt. [Online; accessed 19-February-2016].

Gordon, N. J., Salmond, D. J., and Smith, A. F. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET.

Hethcote, H. W. (2000). The mathematics of infectious diseases. *SIAM review*, 42(4):599–653.

Hickmann, K. S., Fairchild, G., Priedhorsky, R., Generous, N., Hyman, J. M., Deshpande, A., and Del Valle, S. Y. (2015). Forecasting the 2013âĂŞ2014 influenza season using wikipedia. *PLoS Comput Biol*, 11(5):e1004239.

Jewell, C. P., Kypraios, T., Neal, P., Roberts, G. O., et al. (2009). Bayesian analysis for emerging infectious diseases. *Bayesian Analysis*, 4(3):465–496.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

Kastner, G. (2016). Dealing with stochastic volatility in time series using the r package stochvol. *Journal of Statistical Software*, 69(1):1–30.

King, A., Nguyen, D., and Ionides, E. (2016). Statistical inference for partially observed markov processes via the r package pomp. *Journal of Statistical Software*, 69(1):1–43.

Knape, J. and de Valpine, P. (2012). Fitting complex population models by combining particle filters with markov chain monte carlo. *Ecology*, 93(2):256–263.

Knorr-Held, L. and Richardson, S. (2003). A hierarchical model for space–time surveillance data on meningococcal disease incidence. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 52(2):169–183.

Lampos, V., Miller, A. C., Crossan, S., and Stefansen, C. (2015). Advances in nowcasting influenza-like illness rates using search query logs. *Scientific reports*, 5.

Lawson, A. B. (2013). *Bayesian disease mapping: hierarchical modeling in spatial epidemiology*. CRC press.

Lazer, D., Kennedy, R., King, G., and Vespignani, A. (2014). The parable of google flu: Traps in big data analysis. *Science*, 343(6176):1203–1205.

Lekone, P. E. and Finkenstädt, B. F. (2006). Statistical inference in a stochastic epidemic seir model with control intervention: Ebola as a case study. *Biometrics*, 62(4):1170–1177.

Liu, J. and West, M. (2001). Combined parameter and state estimation in simulation-based filtering. In *Sequential Monte Carlo methods in practice*, pages 197–223. Springer.

Lunn, D., Jackson, C., Best, N., Thomas, A., and Spiegelhalter, D. (2012). *The BUGS Book: A practical introduction to Bayesian analysis*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis.

McCallum, H., Barlow, N., and Hone, J. (2001). How should pathogen transmission be modelled? *Trends in ecology & evolution*, 16(6):295–300.

Murray, L. (2015). Bayesian state-space modeling on high-performance hardware using libbi. *Journal of Statistical Software*, 67(1):1–36.

Niemi, J. (2015). *MMWRweek: Convert Dates to MMWR Day, Week, and Year*. R package version 0.1.1.

NIMBLE Development Team (2015). *NIMBLE Users Manual, Version 0.4*.

Nishiura, H. et al. (2011). Real-time forecasting of an epidemic using a discrete time stochastic model: a case study of pandemic influenza (h1n1-2009). *Biomed Eng Online*, 10:15.

Nsoesie, E., Mararthe, M., and Brownstein, J. (2013). Forecasting peaks of seasonal influenza epidemics. *PLoS currents*, 5.

Ortiz, J. R., Zhou, H., Shay, D. K., Neuzil, K. M., Fowlkes, A. L., and Goss, C. H. (2011). Monitoring influenza activity in the united states: A comparison of traditional surveillance systems with google flu trends. *PLoS ONE*, 6(4):e18687.

Paul, M. J., Dredze, M., and Broniatowski, D. (2014). Twitter improves influenza forecasting. *PLoS currents*, 6.

Petris, G. (2010). An r package for dynamic linearmmodels. *Journal of Statistical Software*, 36(1):1–16.

Pitt, M. K. (2002). Smooth particle filters for likelihood evaluation and maximisation.

Pitt, M. K. and Shephard, N. (1999a). Filtering via simulation: Auxiliary particle filters. *Journal of the American statistical association*, 94(446):590–599.

Pitt, M. K. and Shephard, N. (1999b). Time varying covariances: A factor stochastic volatility approach. *Bayesian statistics*, 6:547–570.

Pitt, M. K. and Shephard, N. (2001). Auxiliary variable based particle filters. In *Sequential Monte Carlo methods in practice*, pages 273–293. Springer.

Plummer, M. (2016). *rjags: Bayesian graphical models using MCMC*. R package version 4-5.

Plummer, M., Best, N., Cowles, K., and Vines, K. (2006). Coda: Convergence diagnosis and output analysis for mcmc. *R News*, 6(1):7–11.

Plummer, M. et al. Jags: A program for analysis of bayesian graphical models using gibbs sampling.

Polson, N. G., Stroud, J. R., and Müller, P. (2008). Practical filtering with sequential parameter learning. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(2):413–428.

R Core Team (2015a). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria.

R Core Team (2015b). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rudis, B. (2015). *cdcfluview: Retrieve U.S. flu season data from the CDC FluView portal*. R package version 0.4.0.

Shaman, J. and Karspeck, A. (2012). Forecasting seasonal outbreaks of influenza. *Proceedings of the National Academy of Sciences*, 109(50):20425–20430.

Shaman, J., Karspeck, A., Yang, W., Tamerius, J., and Lipsitch, M. (2013). Real-time influenza forecasts during the 2012 - 2013 season. *Nature Communications*, 4.

Smith, A., Doucet, A., de Freitas, N., and Gordon, N. (2001). *Sequential Monte Carlo methods in practice*. Springer Science & Business Media.

Song, J. Y., Cheong, H. J., Choi, S. H., Baek, J. H., Han, S. B., Wie, S.-H., So, B. H., Kim, H. Y., Kim, Y. K., Choi, W. S., Moon, S. W., Lee, J., Kang, G. H., Jeong, H. W., Park, J. S., and Kim, W. J. (2013). Hospital-based influenza surveillance in korea: Hospital-based influenza morbidity and mortality study group. *Journal of Medical Virology*, 85(5):910–917.

Stan Development Team (2015). Stan: A c++ library for probability and sampling, version 2.10.0.

Thompson, W. W., Shay, D. K., Weintraub, E., Brammer, L., Bridges, C. B., Cox, N., and Fukuda, K. (2004). Influenza-associated hospitalizations in the united states. *The Journal of the American Medical Association*, 292(11):1333–1340.

Todeschini, A., Caron, F., Fuentes, M., Legrand, P., and Del Moral, P. (2014). Biips: software for bayesian inference with interacting particle systems. *arXiv preprint arXiv:1412.3779*.

U.S. Centers for Disease Control and Prevention (2016a). ILINet Data. http://gis.cdc.gov/grasp/fluview/fluportaldashboard.html. [Online; accessed 19-February-2016].

U.S. Centers for Disease Control and Prevention (2016b). MMWR Weeks. https://wwwn.cdc.gov/nndss/document/MMWR_week_overview.pdf. [Online; accessed 19-February-2016].

Wikle, C. K., Milliff, R. F., Herbei, R., and Leeds, W. B. (2013). Modern statistical methods in oceanography: A hierarchical perspective. *Statist. Sci.*, 28(4):466–486.

Yang, S., Santillana, M., and Kou, S. C. (2015). Accurate estimation of influenza epidemics using google search data via argo. *Proceedings of the National Academy of Sciences*, 112(47):14473–14478.

Yang, W., Karspeck, A., and Shaman, J. (2014). Comparison of filtering methods for the modeling and retrospective forecasting of influenza epidemics. *PLoS Comput Biol*, 10(4):e1003583.

Zhang, L. and Wu, X. (2006). An edge-guided image interpolation algorithm via directional filtering and data fusion. *Image Processing, IEEE Transactions on*, 15(8):2226–2238.

Zhou, Y. (2015). vsmc: Parallel sequential monte carlo in c++. *Journal of Statistical Software*, 62(1):1–49.