INFERENCE ON SPATIAL INFECTIOUS-DISEASE MODELS WITH PLANTATION DATA

ALEXANDER REMOROV
996712598

SUPERVISOR: PROF. JEFFREY ROSENTHAL
BASED ON JOINT WORK WITH:
PATRICK BROWN, FLORENCIA CHIMARD, JEFFREY ROSENTHAL, XIN WANG

Date: December 13, 2011.

Contents

1. Introduction

In this report we summarize our work on estimation of parameters in Spatial Infectious-Disease Models from sugar cane plant infection data, using MCMC methods. We fit a basic infectious disease model, which is a case of a more general model studied by Gibson (1997) [2] and by Keeling & Rohani (2008) [5]. For each plant the model assumes spontaneous infection which follows a Poisson process with time-dependent rate determined by a parameter μ and the state (healthy/infected) of other plants.

The likelihood function was highly dimensional, since it was determined by three parameters and 1742 infection times of the plants. For simplicity we first started with a discrete time model and tried MCMC with Gibbs Sampling. However, running a program in C implementing the algorithm took a very long time, even for 500 iterations [1] (our goal was to use at least 11000 iterations). As a result, we decided to first consider other methods of estimating the parameters.

The first thing we considered was using Particle MCMC (PMCMC) methods. A good survey paper on this rather new MCMC approach is by Andrieu, Doucet, and Holenstein (2010) [1]. They mention that PMCMC is useful for estimation of parameters in highly dimensional models with a lot of dependence in the parameters. In particular, this approach has an advantage over standard MCMC because the latter algorithm might get "stuck" in a local maximum because the likelihood function is very complex. The drawback is that the algorithm takes a significantly longer time to complete one iteration in comparison to standard MCMC, especially if a large number of particles are used.

Since we did not have previous experience with using MCMC, we first attempted applying this method to several toy examples. The first two were simplified versions of a hidden AR(1) model considered by Andrieu, Doucet, and Holenstein (2010), while the third was a simple hidden birth-death process studied by Läubli (2011) [6]. While the results for the first two examples were not satisfactory (possibly because not enough particles were used), applying PMCMC to the birth-death process produced reasonable results, and we decided to attempt to apply the method to the discrete version of our model. However, we realized that the code was getting rather complicated, and the running time would not be better than for standard MCMC with Gibbs Sampling. Thus we abandoned this approach.

We considered running parts of the program in parallel, however before doing this, we thought about whether it would be possible to simplify the likelihood function for the discrete version of the model. It turned out that a few drastic simplifications were possible, which could significantly reduce running time. After implementing these simplifications, the new version of the program ran faster, but would still take a relatively long time to complete.

[1]Xin Wang ran the algorithm for 500 iterations, and it took about 5 days to complete.

In order to further improve running time, we decided to use an approximation of the likelihood function (by truncating the kernel of the gaussian functions which were part of the likelihood function). Implementing this new idea required a programming trick, which when implemented, led to a drastic reduction in running time. The final version of the program for the discrete time model ran in around 40 minutes for 110000 iterations; the original version would take approximately 110 days to complete [2] .

Finally, we considered a continuous-time version of the program, which was more realistic than the discrete one. After implementing similar simplifications as with the discrete version, the resulting program also ran very fast and ran in around 30 minutes, while producing estimates that were similar to the discrete version. We were pleasantly surprised that using standard MCMC still allowed us to efficiently estimate the parameters in a highly dimensional and complex model. We believe that it may be possible to use a similar approach to perform inference in an even more complicated infectious disease model.

This report is organized as follows. In section 2 we present both the discrete and continuous versions of the infectious disease model fit to the data. Section 3 includes a brief description of the data. In section 4 we present the results of the original version Metropolis within Gibbs Sampler algorithm applied to the data. Section 5 contains a background on PMCMC methods, while Section 6 includes three toy examples of application of PMCMC. In section 7 we discuss our attempts to apply PMCMC to our problem. Section 8 describes the application of MCMC with Gibbs Sampling and Simplification for both the discrete and the continuous time models, and the results of this application. Section 9 provides some concluding remarks, while section 10 contains the Acknowledgements.

2. Model

Consider a field of plants $S_1, S_2, \dots, S_N$. We assume the infection is spread by aphids. When aphids land on a plant, they stay there (and reproduce), so that at any point in time a plant is either healthy or non-infected. Furthermore, once a plant gets infected, it stays infected.

Once a plant is infected, the infection spreads from this plant at constant rate θ (not dependent on how long the plant has already been infected for). As a result, the rate at which the infection spreads from an infected plant S_i to a healthy plant S_j (for $1 \leq i, j \leq N$) is $\theta f(D(i,j))$, where $D(i,j)$ is the Euclidian distance between plants S_i and S_j, and f is a kernel function.

We restricted f to be a Gaussian depending on a parameter σ:

$$f(D) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-D^2}{\sigma^2}} \tag{1}$$

For each plant S_i let τ_i be the time when plant S_i becomes infected. If plant S_i is never infected, we set $\tau_i := \infty$.

We assume that for each healthy plant S_i, the infection rate is determined by a constant μ and the cumulative rate at which the infection spreads from infected plants to the healthy plant.

[2] Based on the assumption that it takes about 5 days to execute 500 iterations.

Therefore, we define this rate at time t to be:

$$\lambda_{i,t} = \mu + \theta \sum_{j:\tau_j < t} f(D(i,j)) \tag{2}$$

The above form is similar to the model studied by Meyer et al. (2011) [7].

We set Inverse Gamma priors for the parameters μ, θ, σ so that $\mu \sim IG(a_1, b_1), \theta \sim IG(a_2, b_2), \sigma \sim IG(a_3, b_3)$. Define:

$$S(\mu, \theta, \sigma) = -(a_1 + 1)\theta - \frac{b_1}{\theta} - (a_2 + 1)\mu - \frac{b_2}{\mu} - (a_3 + 1)\sigma - \frac{b_3}{\sigma} \tag{3}$$

We assume that the field of plants is observed at times $t_0 = 0, t_1, t_2, \ldots, t_l = K$ with $0 < t_1 < t_2 < \ldots < t_k$, and at time 0 all plants are healthy. At each observation time for each plant we record whether it is healthy or infected. As a result for each plant S_i, we can record L_i - the last time the plant was observed uninfected, with $L_i = t_j$ for some j. If $L_i = K$, we know the plant is never infected during the whole observation period $[0, T]$. Otherwise the plant was first observed infected at time $U_i = t_{j+1}$, hence $\tau_i \in (L_i, U_i]$. We set uniform priors for τ_i (so in the discrete model τ_i has a uniform distribution over the integers $L_i + 1, L_i + 2, \ldots, U_i$, while in the continuous model $\tau_i \sim Uniform[L_i, U_i]$).

2.1. **Discrete Model.** In the discrete model we assume that time is discrete and $K \in \mathbb{N}$, so that $\tau_i \in \{0, 1, \ldots, K\}$. Then for $k = 1, 2, \ldots, K$, the probability that the plant S_i is healthy at time k conditional on it being healthy at time $k - 1$ is $e^{-\lambda_{i,k}}$, so that:

$$\mathbb{P}(\tau_i > k | \tau_i > k - 1) = e^{-\lambda_{i,k}}; \qquad \mathbb{P}(\tau_i = k | \tau_i > k - 1) = 1 - e^{-\lambda_{i,k}} \tag{4}$$

We assume that whether a plant becomes infected in the next period depends only on the state (healthy/infected) of the other plants in the current period. Therefore, by taking the product over the times $1, 2, \ldots, K$ and over the plant infection probabilities at each time, we get an expression for the likelihood function up to a multiplicative constant:

$$\pi(\theta, \mu, \sigma, \{\tau_i\}) \propto e^{S(\mu,\theta,\sigma)} \prod_{k=1}^{K} (\prod_{i:\tau_i = k} (1 - e^{-\lambda_{i,k}}) \prod_{i:\tau_i > k} e^{-\lambda_{i,k}}) \prod_{i \in \mathcal{X}} I(L_i < \tau_i \leq U_i) \tag{5}$$

where $\mathcal{X} = \{1, 2, \ldots, N\}$. We set $\log \pi(\theta, \mu, \sigma, \{\tau_i\}) := -\infty$ if $\tau_i \notin (L_i, U_i]$ for some $i \in \mathcal{X}$, and otherwise:

$$\begin{aligned} \log \pi(\theta, \mu, \sigma, \{\tau_i\}) &= C + S(\mu, \theta, \sigma) + \sum_{k=1}^{K} (\sum_{i:\tau_i = k} \log(1 - e^{-\lambda_{i,k}}) - \sum_{i:\tau_i > k} \lambda_{i,k}) \\ &= C + S(\mu, \theta, \sigma) + \sum_{i:\tau_i \leq K} \log(1 - e^{-\lambda_{i,k}}) - \sum_{i \in \mathcal{X}} \sum_{k=1}^{\min(K, \tau_i - 1)} \lambda_{i,k} \end{aligned} \tag{6}$$

for some constant C.

2.2. **Continuous Model.** In the continuous model, we assume the infections follow a homogeneous Poisson process with rate $\lambda_{i,t}$ for plant S_i at time $t \in [0, T]$. Then:

$$\log \pi(\theta, \mu, \sigma, \{\tau_i\}) = C + S(\mu, \theta, \sigma) + \sum_{i:\tau_i > K} (-\int_0^K \lambda_{i,t} dt) + \sum_{i:\tau_i \leq K} (-\int_0^K \lambda_{i,t} dt + \log(\lambda_{i,\tau_i}))$$

which can be written as:

$$\log \pi(\mu, \theta, \sigma, \{\tau_i\}) = C + S(\mu, \theta, \sigma) - \sum_{i:\tau_i > K} [K\mu + \theta \sum_{j:\tau_j < K} (K - \tau_j) f(D(i,j))] + \tag{7}$$

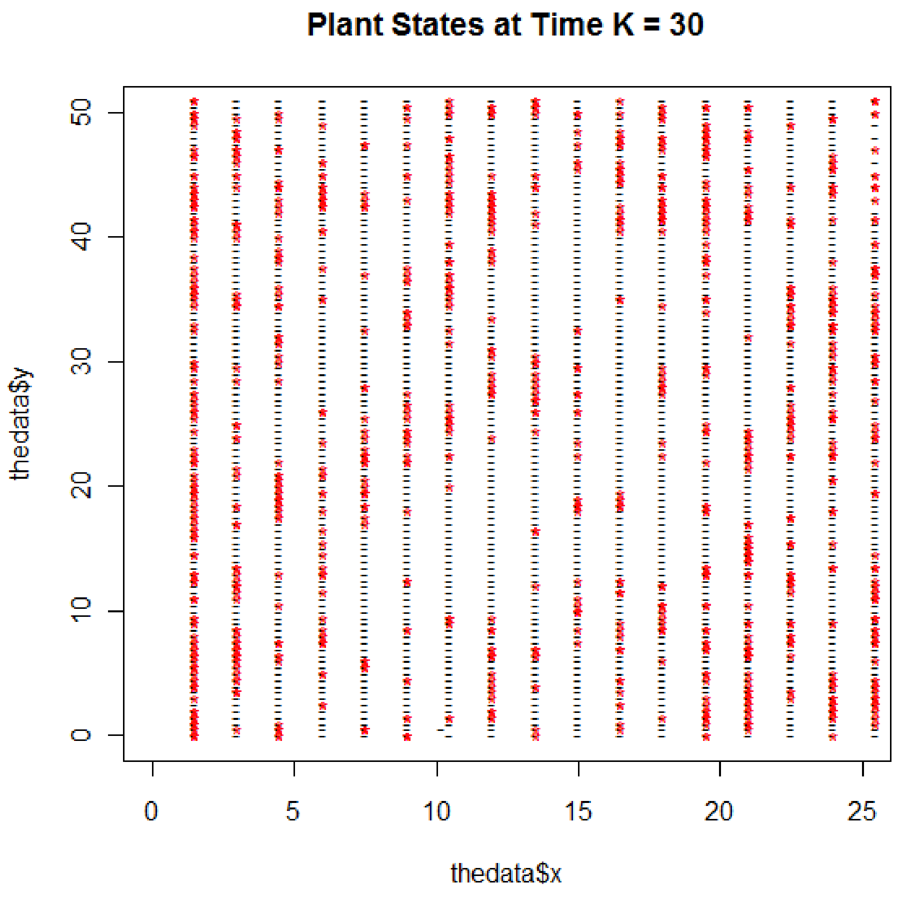
$$\sum_{i:\tau_i \leq K} [-\mu\tau_i - \theta \sum_{j:\tau_j < \tau_i} (\tau_i - \tau_j) f(D(i,j)) + \log(\mu + \theta \sum_{j:\tau_j < \tau_i} f(D(i,j)))]$$

3. Data

During the experiment a field of sugar cane plants in Guadeloupe was observed. The size of the field was around 25 by 50 m, and the plants were in a grid with dimensions around 17 by 103 plants. The total number of plants was 1742.

Plants were first observed at time 0 and were all healthy at that time. They were then observed after 6, 10, 14, 19, 23, and 30 weeks. For the discrete time model, we assume time is measured in weeks, so that $K = 30$.

The state of the plants at the time of last observation is shown below. If a plant has a red star in its location, it is infected.



4. MCMC with Gibbs Sampling and no Simplification

We first present our original approach to estimating the parameters in the discrete model (we decided to start with the discrete model since it seemed simpler). We used Metropolis within Gibbs Sampling as follows:

Step $n = 1$:
Set $\mu(1), \theta(1), \sigma(1)$ arbitrary positive real numbers, and for every $i \in \mathcal{X}$, set $\tau_i(1)$ an arbitrary integer in $(L_i, U_i]$.

Steps $n = 2, \ldots, M$:

1. Propose $\mu' \sim N(\mu(n-1), \sigma_1^2)$; set $\mu(n) := \mu'$ if $\log(U) < \log \pi' - \log \pi_{old}$; otherwise set $\mu(n) := \mu(n-1)$.
2. Propose $\theta' \sim N(\theta(n-1), \sigma_2^2)$; set $\theta(n) := \theta'$ if $\log(U) < \log \pi' - \log \pi_{old}$; otherwise set $\theta(n) := \theta(n-1)$.
3. One-by-one for each $i \in \mathcal{X}$ propose $\tau_i' \sim \tau_i(n-1) - 1 + 2 \times Bernoulli(\frac{1}{2})$. Set $\tau_i(n) := \tau_i'$ if $\log(U) < \log \pi' - \log \pi_{old}$; otherwise set $\tau_i(n) := \tau_i(n-1)$.

4. Propose $\sigma' \sim N(\sigma(n-1), \sigma_3^2)$; set $\sigma(n) := \sigma'$ if $\log(U) < \log \pi' - \log \pi_{old}$; otherwise set $\sigma(n) := \sigma(n-1)$.

In each of the above steps, U is a $Uniform(0,1)$ random variable, $\log \pi'$ is the log likelihood with the "proposed" parameter instead of the old one (while the rest of the parameter values from step $n-1$), while $\log \pi_{old}$ is the "old" log likelihood, i.e. with the parameter values from step $n-1$.

Results
In the original program (included in Appendix 2) implementing the algorithm σ was set to 1 to reduce running time and keep the model simpler. Nevertheless, when Xin ran the program for 500 iterations, it took approximately five days to complete. The goal was to execute the algorithm for 11000 iterations (and burn-in of 1000), thus if we assume that each iteration takes roughly the same amount of time (which is reasonable since the same number of steps in the calculations is performed and we assume the values of the parameters are relatively stable), then to complete this many iterations would take about $11000 \times \frac{500}{5} = 110$ days. This would be a very long amount of time, thus we considered using other approaches, and we decided to look into Particle MCMC.

5. Particle MCMC

5.1. **Background.**

5.1.1. *Hidden Markov Models.* Consider the following Hidden Markov Model (also known as a State Space Model). We have a Markov process $\{X_n\}$ with $X_n \in \mathcal{X}$ with initial and transition probability densities given by:

$$X_1 \sim \mu_\theta(\cdot);\;\; X_{n+1}|(X_n = x) \sim f_\theta(\cdot|x) \tag{8}$$

where $\theta \in \Theta$ is a parameter. The process $\{X_n\}$ is observed through a process $\{Y_n\}$ with $Y_n \in \mathcal{Y}$, and Y_ns assumed to be conditionally independent given $\{X_n\}$. Y_n depends on $\{X_n\}$ according to the following marginal probability density:

$$Y_n|(X_1, X_2, \ldots, X_n = x, \ldots, X_T) \sim g_\theta(\cdot|x) \tag{9}$$

Let T be the length of the Markov chain $\{X_n\}$. For any $k, l \in \mathbb{N}, k \le l$ denote by $x_{k:l}$ the chain $(x_k, x_{k+1}, \ldots, x_l)$. Then for fixed $\theta \in \Theta$, the joint density of $(x_{1:T}, y_{1:T})$ is:

$$p_\theta(x_{1:T}, y_{1:T}) = \mu_\theta(x_1) \prod_{n=2}^{T} f_\theta(x_n|x_{n-1}) \prod_{n=1}^{T} g_\theta(y_n|x_n) \tag{10}$$

Therefore given a prior density $p(\theta)$ for θ, the posterior density $p_\theta(x_{1:T}|y_{1:T})$ satisfies:

$$p_\theta(x_{1:T}|y_{1:T}) \propto p_\theta(x_{1:T}, y_{1:T})p(\theta) \tag{11}$$

The goal is, after observing $\{Y_n\}$, to perform Bayesian inference about θ. However, the joint density in (10) may be highly dimensional and furthermore a complicated expression, especially if there is high dependence of the X_ns. As a result, standard MCMC methods may not be able to efficiently sample from such a distribution and may "get stuck" in local maxima.

5.2. **Sequential Monte Carlo Sampling.** The following Sequential Monte Carlo (SMC) algorithm is used to produce a sample chain $\{X_n\} \in \mathcal{X}^T$ with distribution according to $p_\theta(x_{1:T}|y_{1:T})$ in (10) based on the observed data $\{Y_n\}_{1\leq n\leq T}$ for a fixed $\theta \in \Theta$. We again assume that T is the length of the Markov chain $\{X_n\}$. The idea is to sample N particles at each step; for $n \geq 2$ every next particle X_n^k is sampled based on some ancestor particle X_{n-1}^k and the corresponding obervsed value y_n. The sampling is done using impotance sampling (we assume we know the joint density $p_\theta(x_{1:T}, y_{1:T})$ up to a multiplicative constant for a given θ).

Step $n = 1$:

1. Sample the first particles $X_1^k \sim q_\theta(\cdot|y_1)$.
2. Compute and normalize weights:

$$\tilde{W}_1^k := \frac{p_\theta(X_1^k, y_1)}{q_\theta(X_1^k|y_1)} = \frac{\mu_\theta(X_1^k)g_\theta(y_1|X_1^k)}{q_\theta(X_1^k, y_1)}$$

$$W_1^k := \frac{\tilde{W}_1^k}{\sum_{l=1}^N \tilde{W}_1^l}$$

Steps $n = 2, \dots, T$:

1. Sample the ancestor indices $A_{n-1}^k \sim \mathcal{F}(\cdot|\mathbf{W}_{n-1})$.
2. Sample the nth particles $X_n^k \sim q_\theta(\cdot|y_n, X_{n-1}^{A_{n-1}^k})$. Set $X_{1:n}^k := (X_{1:(n-1)}^{A_{n-1}^k}, X_n^k)$.
3. Compute and normalize weights:

$$\tilde{W}_n^k := \frac{p_\theta(X_{1:n}^k, y_{1:n})}{q_\theta(X_n^k|y_n, X_{n-1}^{A_{n-1}^k})p_\theta(X_{1:(n-1)}^{A_{n-1}^k}, y_{1:(n-1)})} = \frac{f_\theta(X_n^k|X_{n-1}^{A_{n-1}^k})g_\theta(y_n|X_n^k)}{q_\theta(X_n^k|y_n, X_{n-1}^{A_{n-1}^k})}$$

$$W_n^k := \frac{\tilde{W}_n^k}{\sum_{l=1}^N \tilde{W}_n^l}$$

Here $\mathcal{F}(\cdot|\mathbf{W}_{n-1})$ is the discrete multinomial distribution.

The sample $\hat{X}_{1:T}$ is produced by sampling an index k in $\{1, 2, \dots, N\}$ using the discrete multinomial distribution on the last vector of weights $(W_T^1, W_T^2, \dots, W_T^N)$ and setting $\hat{X}_{1:T} := X_{1:T}^k$.

The SMC algorithm also allows to estimate the marginal likelihood $p_\theta(y_{1:T})$ as follows:

$$\hat{p}_\theta(y_{1:T}) := \hat{p}(y_1) \prod_{n=2}^T \hat{p}_\theta(y_n|y_{1:(n-1)})$$

where:

$$\hat{p}_\theta(y_n|y_{1:(n-1)}) := \frac{1}{N}\sum_{k=1}^N \tilde{W}_n^k \tag{12}$$

Problems with SMC algorithms:

1. It is not clear how to find "good" proposal densities $q_\theta(x_1|y_1)$ and $q_\theta(x_n|y_n, x_{n-1})$ for $n = 2, \ldots, T$. An "extreme" case by Gordon et. al (1993) [3] is to use the prior densities: $q_\theta(x_1|y_1) = \mu_\theta(x_1)$, $q_\theta(x_n|y_n, x_{n-1}) = f_\theta(x_n|x_{n-1})$. According to Ionides et. al (2006) [4], this is actually the only possible choice if $f_\theta(x_n|x_{n-1})$ is very difficult/expensive to sample pointwise, but not hard to sample from.
2. As T becomes large, fewer and fewer components sampled at time $n << T$ still have descendants at time T, thus when $T - n$ is large, the approximation of the marginal density $p_\theta(x_n|y_{1:T})$ may be poor. However, according to Andrieu, Doucet, and Holenstein (2010) [1], this is not a problem when using SMC in PMCMC.
3. The algorithm takes a long time to run.
4. It is possible that some of the un-normalized weights $\tilde{W}_k^n$ are so small, that they are perceived as 0 by computer software. It may happen that during a particular step all steps are so small, that they are perceived as 0; then they cannot be normalized and the program crashes.

5.3. **Conditional SMC Algorithm.** The following modification of the SMC Algorithm is used in the Particle Gibbs (PG) Sampler. The idea is to start with a fixed chain $\tilde{X}_{1:T}$ (and a fixed vector of ancestor indices $\tilde{B}_{1:T}$ with $\tilde{B}_i \in \{1, 2, \ldots, N\}$), generate the remaining particles as in a standard SMC algorithm, but ensure that the chain $\tilde{X}_{1:T}$ survives all T sampling/resampling steps.

Step $n = 1$:

1. For $k \neq \tilde{B}_1$ sample the first particles $X_1^k \sim q_\theta(\cdot|y_1)$; set $X_1^{\tilde{B}_1} := \tilde{X}_1$.
2. Compute and normalize weights:

$$\tilde{W}_1^k := \frac{\mu_\theta(X_1^k) g_\theta(y_1|X_1^k)}{q_\theta(X_1^k, y_1)}; \; W_1^k := \frac{\tilde{W}_1^k}{\sum_{l=1}^N \tilde{W}_1^l}$$

Steps $n = 2, \ldots, T$:

1. For $k \neq \tilde{B}_n$ sample the ancestor indices $A_{n-1}^k \sim \mathcal{F}(\cdot|\mathbf{W}_{n-1})$; set $A_{n-1}^{\tilde{B}_n} := \tilde{B}_{n-1}$.
2. For $k \neq \tilde{B}_n$ sample the nth particles $X_n^k \sim q_\theta(\cdot|y_n, X_{n-1}^{A_{n-1}^k})$; set $X_n^{\tilde{B}_n} := \tilde{X}_n$. For all k set $X_{1:n}^k := (X_{1:(n-1)}^{A_{n-1}^k}, X_n^k)$.
3. Compute and normalize weights:

$$\tilde{W}_n^k := \frac{f_\theta(X_n^k|X_{n-1}^{A_{n-1}^k}) g_\theta(y_n|X_n^k)}{q_\theta(X_n^k|y_n, X_{n-1}^{A_{n-1}^k})}; \; W_n^k := \frac{\tilde{W}_n^k}{\sum_{l=1}^N \tilde{W}_n^l}$$

The sample $\hat{X}_{1:T}$ is then produced just like in the standard SMC algorithm.

5.4. **Particle Markov Chain Monte Carlo - PMMH.** We first present the Particle Marginal Metropolis Hastings (PMMH) Algorithm for performing inference for $\{X_n\}$ and θ based on observed data $\{y_n\}_{1 \leq n \leq T}$.

Step 1:

1. Set $\theta(0)$ arbitrary.
2. Run an SMC algorithm targeting $p_{\theta(0)}(x_{1:T}|y_{1:T})$; sample $X_{1:T}(0) \sim \hat{p}_{\theta(0)}(\cdot|y_{1:T})$; let $\hat{p}_{\theta(0)}(y_{1:T})$ be the estimate of the marginal likelihood.

Steps $k = 2, \ldots, M$:

1. Sample $\theta^* \sim q(\cdot|\theta(i-1))$.
2. Run an SMC algorithm targeting $p_{\theta^*}(x_{1:T}|y_{1:T})$; sample $X^*_{1:T} \sim \hat{p}_{\theta^*}(\cdot|y_{1:T})$; let $\hat{p}_{\theta^*}(\cdot|y_{1:T})$ be the marginal likelihood estimate.
3. Compute the following acceptance probability π:
$$\pi := 1 \wedge \frac{\hat{p}_{\theta^*}(y_{1:T})p(\theta^*)}{\hat{p}_{\theta(i-1)}(y_{1:T})p(\theta(i-1))}\frac{q(\theta(i-1)|\theta^*)}{q(\theta^*|\theta(i-1))}$$
With probability π set $\theta(i) := \theta^*$, $X_{1:T}(i) := X^*_{1:T}$, $\hat{p}_{\theta(i)}(y_{1:T}) := \hat{p}_{\theta^*}(y_{1:T})$; otherwise set $\theta(i) := \theta(i-1)$, $X_{1:T}(i) := X_{1:T}(i-1)$, $\hat{p}_{\theta(i)}(y_{1:T}) := \hat{p}_{\theta(i-1)}(y_{1:T})$.

5.5. **Particle Gibbs - PG.** We now present the Particle Gibbs (PG) sampler. The algorithm works in the same way as the standard Gibbs sampler, as it alternates between sampling from $p(\theta|x_{t:T}, y_{1:T})$ and $p(x_{1:T}|\theta, y_{1:T}) = p_\theta(x_{1:T}|y_{1:T})$. However, the second type of sampling - from $p_\theta(x_{1:T}|y_{1:T})$ - is often impossible, thus the Conditional SMC sampler is used for this part instead.

Step 1:

1. Set $\theta(0)$ arbitrary, $X_{1:T}(0)$, $B_{1:T}(0)$ arbitrary.

Steps $k = 2, \ldots, M$:

1. Sample $\theta(i) \sim p(\cdot|X_{1:T}(i-1), y_{1:T})$.
2. Run a Conditional SMC algorithm targeting $p_{\theta(i)}(x_{1:T}|y_{1:T})$ conditional on $\tilde{X}_{1:T} := X_{1:T}(i-1)$ with ancestral lineage $\tilde{B}_{1:T} := B_{1:T}(i-1)$; let $\hat{p}_{\theta(i)}(\cdot|y_{1:T})$ be the marginal likelihood estimate.
3. Sample $X_{1:T}(i)$ from $\hat{p}_{\theta(i)}(\cdot|y_{1:T})$ and record $B_{1:T}(i)$.

6. Toy Examples

6.1. **Example 1: Hidden AR(1) Processes.** Before attempting to apply PMCMC to our model, we decided to try it on several toy examples. We first considered two simplified versions of the non-linear hidden Markov model explored by Andrieu, Doucet, Holenstein (2010) [1]. The first version was:

$$X_1 \sim N(0,1); X_n \sim N(X_{n-1}, \sigma_1^2), n = 2, \ldots, T \tag{13}$$

$$Y_n \sim N((\frac{X_n}{2000})^2, \sigma_2^2), n = 1, \ldots, T \tag{14}$$

with parameters σ_1, σ_2. The second was:

$$X_1 \sim N(0,1); X_n \sim N(X_{n-1} + \mu_1, 0.01^2), n = 2, \ldots, T \tag{15}$$

$$Y_n \sim N(\frac{X_n}{10} + \mu_2, 0.01^2), n = 1, \ldots, T \tag{16}$$

with parameters μ_1, μ_2.
For the first model we used the proposal densities $\sigma_1^* \sim |N(\sigma_1(i-1), \tau^2)|, \sigma_2^* \sim |N(\sigma_2(i-1), \tau^2)|$ and for the second model $\mu_1^* \sim N(\mu_1(i-1), \tau^2), \mu_2^* \sim N(\mu_2(i-1), \tau^2)$. Here, τ is a parameter chosen before the algorithm is run.

For simplicity we assume a fat prior for the vector of parameters θ, so that $p(\theta) = 1$ for all θ. Then for each model the acceptance probability reduces to:

$$\pi := 1 \wedge \frac{\hat{p}_{\theta^*}(y_{1:T})}{\hat{p}_{\theta(i-1)}(y_{1:T})} \frac{q(\theta(i-1)|\theta^*)}{q(\theta^*|\theta(i-1))}$$

Finally for SMC sampling, we also assume the simplest case, i.e. $q_\theta(x_1|y_1) = \mu_\theta(x_1)$, $q_\theta(x_n|y_n, x_{n-1}) = f_\theta(x_n|x_{n-1})$.

Note that in order to make sure the non-scaled weights $\tilde{W}_k^n$ do not get too small, we made the mean of Y_n small relative to the variance in the first model - in view of (13), (14) - and made the variance of X_n and Y_n small relative to the mean in the second model - in view of (15), (16). We wanted to not allow the non-scaled weights to not get too small since in (12) the estimate of marginal likelihood is based on those weights, and therefore if $\tilde{W}_k^n$ get sufficiently small, they are perceived as 0 by the program and as a result the estimate of marginal likelihood becomes 0, which does not allow for proper execution of the algorithm.

Results

In the first model the parameter is (σ_1, σ_2) and in the second it is (μ_1, μ_2). For each model we generated a sample Markov Chain $\{x_n\}$ and then a sample of "observed" data $\{y_n\}$, which is then the only data used to make inference about the parameter. We set $N = 200$ particles, $T = 30$ for the first model, and $N = 100$ particles, $T = 20$ for the second model. For the first model we set the "true" value of parameter to be $(2, 0.25)$ and for the second - $(2, 0.5)$. We used 11000 iterations with a burn-in of 1000. We tried the following values for τ: 0.1, 0.01, 0.005, 0.001. For each value of τ, the algorithm was run twice (if the program "crashed" as will be explained later, the algorithm was run again.) The C programs are included in Appendix 2.

The results of running the PMMH algorithm on each of the two models are presented in tables 1 and 2. They are very bad for all possible values of τ we tried. The acceptance probability seems stuck around 0.75 for the first model, and also tends to be high for the second model, while the estimates of the parameter are usually way off in comparison to the true value. Also, quite often the non-scaled weights in some step of the SMC algorithm were all perceived as 0 by the program because they were too small, and as a result the program was not even able to produce an estimate (i.e. it "crashed"). In the tables, r_{acc} is the acceptance rate; the two lines for each value of τ correspond to the two runs.

The unsatisfactory results may be attributed to the fact that there were not enough particles used in the algorithm, or there were not enough iterations. Andrieu, Doucet, Holenstein used 50,000 iterations with a burn-in of 10,000, and found that when the length of the chain was $T = 100$, at least $N = 2000$ particles should be used. Furthermore, as we already mentioned, quite often some of the non-scaled weights are perceived as 0, resulting in a bad approximation of the marginal likelihood. Finally, it is possible that there were some mistakes in the code. However, we focused more on carrying out successful estimation of the parameters for the infectious disease model on the plant data, therefore we did not spend much time working with these two examples and moved on to another, simpler example, just to make sure that the PMMH algorithm worked at least on that one.

TABLE 1. Results of PMMH Algorithm applied to First Model

τ	$\hat{\sigma}_1^2$	$\hat{\sigma}_2^2$	r_{acc}
0.1	2.502950	4.138815	0.747432
	4.138815	2.039372	0.745704
0.01	0.982180	0.471581	0.759069
	1.301243	0.244384	0.756978
0.005	1.811632	0.270937	0.752523
	1.020260	0.375343	0.752978
0.001	1.247902	0.721446	0.757160
	1.323852	0.439553	0.747886

TABLE 2. Results of PMMH Algorithm applied to Second Model

τ	$\hat{\mu}_1^2$	$\hat{\mu}_2^2$	r_{acc}
0.1	7.635100	2.043363	0.701336
	-3.148410	3.814380	0.944995
0.01	1.486573	1.840478	0.999818
	1.319873	-0.591851	0.606055
0.005	1.842549	0.388695	0.999818
	1.679002	0.801131	0.999909
0.001	1.933463	0.321237	0.999909
	2.236695	0.378275	1.000000

6.2. **Birth-Death Process.** We next try to reproduce the results published by Läubli (2011) [6] for estimating the parameters of a simple birth-death process. Läubli considers the following model, in which the sample spaces $\mathcal{X}$ for X_n and $\mathcal{Y}$ for Y_n are both $\{0, 1\}$, and:

$$P(X_n = 1|X_{n-1} = 0) = \theta_1, P(X_n = 0|X_{n-1} = 0) = 1 - \theta_1,\ n = 2, \dots, T$$

$$P(X_n = 0|X_{n-1} = 1) = \theta_2, P(X_n = 1|X_{n-1} = 1) = 1 - \theta_2,\ n = 2, \dots, T$$

for unknown parameters θ_1, θ_2, and $X_1 \sim Bernoulli(p)$, for known parameter p. The process $\{X_n\}$ is observed through a process $\{Y_n\}$ defined as:

$$P(Y_n = x|X_n = x) = q,\ n = 1, \dots, T$$

for a known parameter q.

The proposals for $\theta_1(i + 1)$ and $\theta_2(i + 1)$ are made independently as follows. For $k = 1, 2$, $\theta_k(i)$ is transformed on a logit scale to $h(\theta_k(i))$. Then a new value h' is sampled conditional on $h(\theta_k(i))$ so that $h' \sim N(h(\theta_k(i)), \sigma^2)$, where σ is a parameter chosen before the algorithm is run.

Then h' is transformed using the inverse-logit transformation to get a value $\theta_k(i+1) := h^{-1}(h')$.

Just like with the first two toy examples, we assumed the simplest case, i.e. $q_\theta(x_1|y_1) = \mu_\theta(x_1)$, $q_\theta(x_n|y_n, x_{n-1}) = f_\theta(x_n|x_{n-1})$.

Results

We ran the PMMH algorithm using the code provided by Läubli for $p = 0.99, q = 0.88$, $\sigma = 1$. We set the "true" value of (θ_1, θ_2) to be $(0.4, 0.7)$ for the first two runs and $(0.3, 0.3)$ for the next two runs. For chosen θ_1, θ_2 we generated a sample Markov Chain $\{x_n\}$ and then a sample of "observed" data $\{y_n\}$, which is then the only data used to make inference about the parameter. We ran the algorithm for $M = 1000$ iterations and burn-in of $B = 100$ and obtained satisfactory results, i.e. the mean of $\theta_1(i)$ and $\theta_2(i)$ for $i = B+1, B+2, \ldots, M$ was relatively close to the true values of θ_1, θ_2. The results are presented in Table 3. The R program is included in Appendix 2.

Table 3. Results of PMMH Algorithm applied to Birth-Death Process

True (θ_1, θ_2)	Estimate $(\hat{\theta}_1, \hat{\theta}_2)$	r_{acc}
(0.4, 0.7)	(0.3708613, 0.7761187)	0.249
	(0.5195078, 0.7059881)	0.260
(0.3, 0.3)	(0.3022617, 0.2888999)	0.206
	(0.2649763, 0.3029364)	0.226

We next decided to attempt to apply the Particle Gibbs algorithm to our model.

7. Application of Particle Gibbs to Sugar Canes Model

In order to apply the Particle Gibbs algorithm to the sugar canes data with discrete time, we need to modify the model. In particular, we need to consider the state of the plants at each time $k = 0, 1, \ldots, K$. Thus for $i \in \mathcal{X}$ and $k \in \{0, 1, \ldots, K\}$ define:

$$X_i(k) = \begin{cases} 1, & \text{if plant } S_i \text{ is infected at time } k \\ 0, & \text{otherwise} \end{cases}$$

Then, as in the original model, we have for $i \in \mathcal{X}$ and time $k \geq 1$:

$$\mathbf{P}(X_i(k) = 1 | X_i(k-1) = 0) = 1 - e^{-\lambda_{i,k}} \tag{17}$$

For $k \in \{0, 1, \ldots, K\}$ denote by $\mathbf{X}(k)$ the plant sickness data at time k, i.e. $\mathbf{X}(k) = (X_1(k), X_2(k), \ldots, X_n(k))$. We observe data $\mathbf{Y} = (\mathbf{X}(t_0), \mathbf{X}(t_2), \ldots, \mathbf{X}(t_l))$, where $0 = t_0 < t_1 < \ldots < t_l = K$. Let Z be the complete data, so that $\mathbf{Z} = (\mathbf{X}(0), \mathbf{X}(1), \ldots, \mathbf{X}(K))$.

7.1. Application of Gibbs Sampler.

We will estimate the parameter from the observed data $\mathbf{Y}$ as follows. We run a PG sampler, that iteratively samples $\theta(i)$ and $\mathbf{Z}(i)$:

Step 1:

1. Set $\theta(0)$, $\mathbf{Z}(0)$ arbitrary.

Steps $m = 2, \ldots, M$:

1. Sample $\theta(m)|\mathbf{Z}(m-1)$. (Note that with a usual Gibbs Sampler we sample $\theta(m)|\mathbf{Z}(m-1), \mathbf{Y}$, but in this problem $\mathbf{Y}$ is just a subsequence of $\mathbf{Z}(n)$ for all n, and so the information set $\mathbf{Z}(m)$ is the same as the information set $\mathbf{Z}(m), \mathbf{Y}$). This will be done using the rejection sampler. In particular:
   - We first propose $\theta' \sim N(\theta(m-1), \sigma_1^2)$, and accept (i.e. set $\theta(m) := \theta'$) iff $\log(U) < \log \pi' - \log \pi_{old}$.
   - We then propose $\mu' = N(\mu(m-1), \sigma_2^2)$, and accept (i.e. set $\mu(m) := \mu'$) iff $\log(U) < \log \pi' - \log \pi_{old}$.
   - We then propose $\sigma' = N(\sigma(m-1), \sigma_3^2)$, and accept (i.e. set $\sigma(m) := \sigma'$) iff $\log(U) < \log \pi' - \log \pi_{old}$.
2. Sample $\mathbf{Z}(m)|\theta(m), \mathbf{Y}, \mathbf{Z}(m-1)$. This part is done using the conditional SMC Algorithm. There is however an issue with implementation of this algorithm. For convenience, we will use T instead of K in the remainder of this section.

Suppose we want to sample $\mathbf{X}_{1:T}$ from $f_\theta(\cdot|\mathbf{Y}, \tilde{\mathbf{X}}_{1:T})$, by sampling the elements of the chain $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T$ one-by-one (as in the standard conditional SMC algorithm). Recall that for $2 \le k \le T$, the weight corresponding to particle $\mathbf{X}_n^k$ is:

$$\tilde{W}_n^k := \frac{f_\theta(\mathbf{X}_n^k|\mathbf{X}_{n-1}^{A_{n-1}^k}) g_\theta(\tilde{\mathbf{Y}}_n|\mathbf{X}_n^k)}{q_\theta(\mathbf{X}_n^k|\tilde{\mathbf{Y}}_n, \mathbf{X}_{n-1}^{A_{n-1}^k})} \tag{18}$$

where $\tilde{\mathbf{Y}}_n$ is the observation at time n. However, we only observe $\mathbf{Y} = (\mathbf{X}(t_1), \mathbf{X}(t_2), \ldots, \mathbf{X}(t_l))$. Define $h : I = \{1, \ldots, T\} \to J = \{t_1, \ldots, t_l\}$ so that for each $n \in \{1, 2, \ldots, T\}$, $h(n) = \min\{k | k \in J \text{ and } k \ge n\}$. Then all the information observed at time n is exactly the information observed at time $h(n)$ (because any observations made after time $h(n)$ contain the same amount, or less information, about the states of the plants up to time $h(n)$, than the observation made at time $h(n)$). Therefore $\tilde{\mathbf{Y}}_n = \mathbf{Y}_{h(n)}$ for $n = 1, \ldots, T$, and the formula for the weight in (18) becomes:

$$\tilde{W}_n^k := \frac{f_\theta(\mathbf{X}_n^k|\mathbf{X}_{n-1}^{A_{n-1}^k}) g_\theta(\mathbf{Y}_{h(n)}|\mathbf{X}_n^k)}{q_\theta(\mathbf{X}_n^k|\mathbf{Y}_{h(n)}, \mathbf{X}_{n-1}^{A_{n-1}^k})}$$

For the naive sampling, we assume $q_\theta(\mathbf{X}_n^k|\mathbf{Y}_{h(n)}), \mathbf{X}_{n-1}^{A_{n-1}^k}) = f_\theta(\mathbf{X}_n^k|\mathbf{X}_{n-1}^{A_{n-1}^k}$ and thus:

$$\tilde{W}_n^k := g_\theta(\mathbf{Y}_{h(n)}|\mathbf{X}_n^k) \tag{19}$$

However, in our model, we have some values of n for which $h(n) - n \ge 4$. Take one such value for n. Then $g_\theta(\mathbf{Y}_{h(n)}|\mathbf{X}_n^k)$ is evaluated by considering all possible chains $\hat{\mathbf{X}}_1 := \mathbf{X}_n^k, \hat{\mathbf{X}}_2, \ldots, \hat{\mathbf{X}}_{h(n)-n}, \hat{\mathbf{X}}_{h(n)-n+1} := \mathbf{Y}_{h(n)}$, calculating the likelihood for each chain, and adding the resulting values. While calculating the likelihood for a single chain is not very difficult, the number of possible chains when $h(n) - n \ge 4$ may be huge. In the problem, there are $1742 \approx 1800$ plants, and $m = 6$ observations made, with the time lapse between consecutive observations usually at least 4, therefore there is a good chance that for some n (corresponding to the day right after some day when an observation was made), $h(n) - n \geq 4$ and the number of plants that get infected between time n and $h(n)$ (inclusive) is at least $\frac{1800}{6} = 300$. Thus there are at least 4^{300} different sets of infection times for those plants, and thus at least $4^{300} \approx 4.1 \times 10^{180}$ paths. It will take a very long time (even with parallelization) to calculate so many paths (and this is only for one particle), thus another approach must be used. One idea is to use an approximation for $g_\theta(\mathbf{Y}_{h(n)}|\mathbf{X}_n^k)$; this may be possible if $\mathbf{X}_t$ follows a continuous process; however there does not seem to be an efficient way to produce a good approximation when $\mathbf{X}_n$ is a discrete process.

We first recall the approach used by Läubli (2011) to run the PG sampler for a Stochastic Oregonator. In that problem, the hidden Markov chain is $\{\mathbf{X}_n\}$ with $\mathbf{X}_n = (X_n^1, X_n^2, X_n^3)$ following a jump process with a specified jump matrix $A \in \mathbb{Z}^{3\times 5}$ (so that 5 different reactions are possible). The rate (intensity) of reaction i ($1 \leq i \leq 5$) is:

$$\mu_i(\mathbf{x}) = \sum_{j=1}^{5} \theta_j h_j(\mathbf{x}),$$

$$\text{where } h(\mathbf{x}) = (x^2, x^1x^2, x^1, \frac{1}{2}x^1(x^1-1), x^3)^T$$

and θ_j are the unknown hazard rates. The process is observed at discrete times $t_1, t_2, \ldots, t_m$ with some measurement error $\epsilon_{t_i} \sim N(\mathbf{x}_{t_i}, \Sigma_\nu)$, where Σ_ν is a diagonal matrix with diagonal entries $\frac{1}{\nu}$, where ν is unknown; let $\mathbf{y}_i$ be the observation at time i. Thus the probability density of the observations is:

$$f_\nu(\mathbf{y}_{1:m}|\mathbf{x}_{t_1}, \ldots, \mathbf{x}_{t_m}) = \prod_{i=1}^{m}\prod_{k=1}^{3} \sqrt{\frac{\nu}{2\pi}} e^{-\frac{\nu(y_i^k - x_{t_i}^k)^2}{2}} \tag{20}$$

The problem is, given observations $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m$, to make inference about $\omega := (\theta_1, \ldots, \theta_5, \nu)^T$.

Läubli applies the SMC sampler to this problem as follows. The algorithm must produce a sample chain $\hat{\mathbf{x}}_{1:T} := (\mathbf{x}_{t_1}, \mathbf{x}_{t_2}, \ldots, \mathbf{x}_{t_m})$ based on the observed chain $\mathbf{y}_{1:T}$ and parameter $\hat{\omega} = (\hat{\theta}_1, \ldots, \hat{\theta}_5, \hat{\nu})^T$. This is done by first producing N samples of $\hat{\mathbf{x}}_1$ conditional on $\mathbf{y}_1$ and $\hat{\nu}$ - which is easy in view of (20), so that $\hat{\mathbf{x}}_1 \sim N(\mathbf{y}_1, \Sigma_{\hat{\nu}})$. The weights for these first particles $\hat{\mathbf{x}}_1^1, \ldots, \hat{\mathbf{x}}_1^N$ are calculated using (20), and then standardized just like in Step 1 of the SMC algorithm.

Then for every fixed $n = 2, 3 \ldots, T$:

1. For $k = 1, \ldots, N$ ancestor indices A_{n-1}^k are sampled as in the SMC algorithm.
2. (*This is the key step*) For $k = 1, \ldots, N$ the chain $\mathbf{x}_t^{(k)}$ is simulated on the time interval $[t_{n-1}, t_n]$ with starting value $\mathbf{x}_{t_{n-1}} := \hat{\mathbf{x}}_{n-1}^{A_{n-1}^k}$ and hazard rates $\hat{\theta}_1, \ldots, \hat{\theta}_5$. The value of particle $\mathbf{x}_n^k$ is set to be the value of the chain at time t_n, i.e. $\hat{\mathbf{x}}_n^k := \mathbf{x}_{t_n}^{(k)}$.
3. The weights are calculated using (20) and standardized, as in the SMC algorithm.

We now show how this approach can be applied to our problem. First, we note that forward-simulating the chain is easy, given the current state $\mathbf{x}_t = (x_1(t), \ldots, x_L(t))$ (where L is the total number of plants) and parameters μ, θ.

1. For every healthy plant j, i.e. for which $x_j(t) = 0$, we calculate the rate λ_j, defined as:

$$\lambda_j := \mu + \theta \sum_{j:\tau_j<t} f(D(i,j))$$

2. The next state $\mathbf{x}_{t+1}$ is sampled as:

$$x_j(t+1) = \begin{cases} 1, & x_j(t) = 1 \\ Bernoulli(1 - e^{-\lambda_j}), & x_j(t) = 0 \end{cases}$$

Note that sometimes we will impose the restriction that it is possible to obtain the next observed state from our sampled next state. Thus, if $t < h(t)$, we need to ensure that if a plant is healthy at time t, and healthy at time $h(t)$, then it is also healthy at time $t+1$, so that:

$$x_j(t+1) = \begin{cases} 1, & x_j(t) = 1 \\ 0, & x_j(t) = 0 \text{ and } x_j(h(t)) = 0 \\ Bernoulli(1 - e^{-\lambda_j}), & \text{otherwise} \end{cases}$$

If we try to apply Läubli's approach to this problem, during ith step we would generate N particles $\mathbf{x}^k(t_i)$ based on $\mathbf{x}^{A^k_{i-1}}(t_{i-1})$ by forward-simulating the chain. The problem with this approach is we would need to enforce $\mathbf{x}^k(t_i)$ to be equal to $\mathbf{y}(t_i)$ since the sample values $\mathbf{x}(t_1), \ldots, \mathbf{x}(t_l)$ are generated only for the times when observations have been made. But if $\mathbf{y}(t_i) = \mathbf{x}^k(t_i)$, each weight would be equal to 1, and thus every possible chain generated would have the same likelihood of being sampled.

We propose a slightly different approach. For each $j = 1, 2 \ldots, m$ we simulate the chain on the time interval $[t_{j-1}, t_j - 1]$, with starting value $\mathbf{y}_{t_j}$ and sample parameters μ, θ, σ. We assign to the particle at time j the value of the chain at time $t_j - 1$. Then for each particle $\mathbf{x}^k_j$ the weights are calculated as $W^k_j := \tilde{\pi}_{\mu,\theta,\sigma}(\mathbf{y}_{t_j} | \mathbf{x}^k_j)$, where:

$$\tilde{\pi}_{\mu,\theta,\sigma}(x(t+1)|x(t)) := \prod_{j:x_j(t+1)=0} e^{-\lambda_{j,t}} \prod_{j:x_j(t+1)=1, x_j(t)=0} (1 - e^{-\lambda_{j,t}})$$

However, the issue with this "naive" approach is that we are not using the information at time t_j which we do actually have. Thus, because $\mathbb{P}(x_j(t+1) = 1 | x_j(t) = 0)$ is usually very small, most of the plants at time $t_j - 1$ in the simulated chain would be healthy because they were observed as healthy at time t_{j-1}, and in almost all of the generated chains the plants that get infected between time t_{j-1} and t_j have to do so at time t_j, which is unrealistic. But we know from the observed data that quite a few plants get infected between time t_{l-1} and t_l, so there is a rather significant probability that a significant portion of these plants were already infected before time $t_{l-1} - 1$ and $t_l - 1$, respectively.

As a result, it is necessary to simulate the chain one step at a time, taking into account not just the "forward-propagation" probability, but also the conditional probability $g_\theta(\mathbf{Y}_{h(n)}|\mathbf{X}_n^k)$, which is possible to calculate using a similar trick (and is better than considering all possible chains), but is complicated from a programming point of view, as it requires recursion on the number of periods between the next observed state $\mathbf{Y}_{h(n)}$ and the "current" simulated state $\mathbf{X}_n^k$. Furthermore, it is not clear how many particles are needed to obtain an appropriate approximation of $g_\theta(\mathbf{Y}_{h(n)}|\mathbf{X}_n^k)$ using this method. Finally, we realized that executing this algorithm would also take a long time, and it might actually take longer than the standard MCMC with Gibbs Sampling. We thus decided to go back to our original approach and consider a simplification.

8. MCMC with Gibbs Sampling and Simplification

8.1. **Discrete Model.** We first analyze the complexity of the Metropolis within Gibbs algorithm for the execution of a single iteration, if no simplifications are made. Consider an arbitrary iteration of the algorithm. We propose a new value $\tau_i' = \tau_i \pm 1$ (with probability $\frac{1}{2}$ each) one-by-one for each plant S_i of the N plants. We then accept iff $\log U < \log \pi(\theta, \mu, \sigma, \{\tau_h'\}) - \log \pi(\theta, \mu, \sigma, \{\tau_h\})$ and reject otherwise.
Recall that if $U_i < \tau_i \leq L_i$ for every plant S_i, the formula for $\log \pi(\theta, \mu, \{\tau_h\})$ is:

$$\log \pi(\theta, \mu, \sigma, \{\tau_h\}) = C + S(\mu, \theta, \sigma) + \sum_{k=1}^{K} \Big(\sum_{i:\tau_i = k} \log(1 - e^{-\lambda_{i,k}}) - \sum_{i:\tau_i > k} \lambda_{i,k} \Big) \tag{21}$$

$$= C + S(\mu, \theta, \sigma) + \sum_{k=1}^{K} \Big(\sum_{i \in \mathcal{X}} [I(\tau_i = k) \log(1 - e^{-\lambda_{i,k}}) - I(\tau_i > k)\lambda_{i,k}] \Big) \tag{22}$$

where:

$$\lambda_{i,k} = \mu + \theta \sum_{j:\tau_j < k} f(D(i,j)) = \mu + \theta \sum_{j \in \mathcal{X}} f(D(i,j)) I(\tau_j < k) \tag{23}$$

The calculation of $\log \pi(\theta, \mu, \sigma, \{\tau_h\})$ is done by iterating through each time k in the chain (for $k = 1, \dots, K$), and during each iteration, calculating $\lambda_{i,k}$ for each plant S_i using (23). The calculation of each $\lambda_{i,k}$ is done by iterating through all plants j and adding $\theta f(D(i,j))$ to the running total if $\tau_j < k$. Thus the calculation of each $\lambda_{i,k}$ takes $O(N)$ time, the calculation of all $\lambda_{i,k}$ for fixed k takes $O(N^2)$ time. Therefore the time complexity of a single computation of $\log \pi(\theta, \mu, \{\tau_h\})$ takes $O(N^2 K)$ time.

There are $O(N)$ calculations of $\log \pi(\theta, \mu, \sigma, \{\tau_h\})$ made during a single iteration of the Gibbs sampler (as we propose μ', θ', σ' and τ_i' for each plant S_i of the N plants). Therefore a single iteration of the algorithm takes $O(N^3 K)$ time.

There are $O(N)$ calculations of $\log \pi(\theta, \mu, \{\tau_h\})$ made during a single iteration of the Gibbs sampler (as we propose μ', θ' and τ'_x for each plant x of the N plants). Therefore a single iteration of the Gibbs sampler takes $O(N^3 K)$ time.

We now show how this running time can be significantly improved due to the large number of cancelations in the expression $\log \pi(\theta, \mu, \sigma, \{\tau'_h\}) - \log \pi(\theta, \mu, \sigma, \{\tau_h\})$ when $\tau'_i = \tau_i$ for all but one plant S_i, and for that plant S_i, $\tau'_i = \tau_i \pm 1$. We will show what happens in the case when $\tau'_i = \tau_i + 1$ (the case $\tau'_i = \tau_i - 1$ is similar). Let:

$$\log \pi_k(\theta, \mu, \sigma, \{\tau_h\}) = \sum_{i \in \mathcal{X}} [I(\tau_i = k) \log(1 - e^{-\lambda_{i,k}}) - I(\tau_i > k)\lambda_{i,k}]$$

We consider simplifying each of the expressions

$$D_k = \log \pi_k(\theta, \mu, \sigma, \{\tau'_h\}) - \log \pi_k(\theta, \mu, \sigma, \{\tau_h\})$$

Case 1: $k < \tau_i$ or $k > \tau_i + 1$. Then $I(\tau_j < k) = I(\tau'_j < k)$ for each $j \in \mathcal{X}$, therefore by (23), $\lambda_{j,k} = \lambda'_{j,k}$ for all j. Again, since $I(\tau_j < k) = I(\tau'_j < k)$ and $I(\tau_j = k) = I(\tau'_j = k)$ for each $j \in \mathcal{X}$, it follows by (22) that $D_k = 0$.

Case 2: $k = \tau_i$. Then $(\tau_j < k) = I(\tau'_j < k)$ for each $j \in \mathcal{X}$, therefore $\lambda_{j,k} = \lambda'_{j,k}$ for all j. Furthermore $I(\tau_j < k) = I(\tau'_j < k)$ for each $j \in \mathcal{X} - \{i\}$. Therefore:

$$D_k = D_{\tau_i} = -\log(1 - e^{-\lambda_{i,k}}) - \lambda_{i,k}$$

The complexity of computation of D_{τ_i} is the same as the complexity of computation of $\lambda_{i,k}$, which is $O(N)$.

Case 3: $k = \tau_i + 1$. Then $I(\tau_j < k) = I(\tau'_j < k)$ for each $j \in \mathcal{X} - \{i\}$. Therefore $\lambda'_{j,k} - \lambda_{j,k} = -\theta f(D(i,j))$ for each $j \in \mathcal{X} - \{i\}$. We have:

$$D_k = D_{\tau_i+1} = \sum_{j \in \mathcal{X}} [I(\tau'_j = \tau_i + 1) \log(1 - e^{-\lambda'_{j,k}}) - I(\tau'_j > \tau_i + 1)\lambda'_{j,k}] -$$

$$\sum_{j \in \mathcal{X}} [I(\tau_j = \tau_i + 1) \log(1 - e^{-\lambda_{j,k}}) - I(\tau_j > \tau_i + 1)\lambda_{j,k}]$$

$$= \sum_{j \in \mathcal{X} - \{i\} : \tau_j = \tau_i + 1} [\log(1 - e^{-\lambda'_{j,k}}) - \log(1 - e^{-\lambda_{j,k}})] - \sum_{j \in \mathcal{X} - \{x\} : \tau_j > \tau_i + 1} (\lambda'_{j,k} - \lambda_{j,k}) + [\log(1 - e^{-\lambda_{i,k}}) + \lambda_{i,k}]$$

$$= \sum_{j \in \mathcal{X} - \{i\} : \tau_j = \tau_i + 1} [\log(1 - e^{-\lambda'_{j,k}}) - \log(1 - e^{-\lambda_{j,k}})] + \sum_{j \in \mathcal{X} - \{i\} : \tau_j > \tau_i + 1} \theta f(D(i,j)) + [\log(1 - e^{-\lambda_{i,k}}) + \lambda_{i,k}]$$

The first sum cannot be simplified further. The computation of that sum takes $O(Ns_k)$ time, where s_k is the number of plants that get infected at time $k = \tau_i + 1$ (excluding plant x). The computation of the second sum takes $O(N)$ time, while the computation of the third sum also takes $O(N)$ time. Thus the computation of D_{τ_i+1} takes $O(Ns_{\tau_i+1})$ time.

Using the above simplification it follows that the computation of $D = \log \pi(\theta, \mu, \sigma, \{\tau'_h\}) - \log \pi(\theta, \mu, \sigma, \{\tau'_h\})$ can be done in $O(Ns_{\tau_i+1})$ time. On average, about $\frac{N}{K}$ plants get sick at a particular time (and this number is usually twice less than that since at time K around a third of all the plants get sick), so we can assume that $s_{\tau_i+1} \in O(\frac{N}{K})$ and thus the computation of D can be done in $O(\frac{N^3}{K})$ time when τ_i' is proposed for a single plant S_i. Hence the total computation of D for each of the N plants takes $O(\frac{N^3}{K})$ time. The computation of D when we propose μ' or θ' takes $O(N^2K)$ time. Thus the complexity of a single iteration of the Gibbs Sampler can be reduced to $O(N^2 \max(\frac{N}{K}, K))$. In our problem $N > K^2$ and thus the complexity becomes $O(\frac{N^3}{K})$, an improvement of a factor of K^2 over the previous bound of $O(N^3K)$. In our problem $K = 30$, thus this factor is around 900.

We also note that $D = \log \pi(\theta, \mu', \sigma, \{\tau_h\}) - \log \pi(\theta, \mu, \sigma, \{\tau_h\})$ can be significantly simplified, leading to faster computation of the portion of the algorithm in which a new value of μ is proposed. While this does not change the running time complexity (as such simplification is not possible when a new value of θ is proposed), it still leads to a significant reduction in computation time. Using (21), we have:

$$D = \sum_{k=1}^{K}[\sum_{i:\tau_i=k} (\log(1 - e^{-\lambda'_{i,k}}) - \log(1 - e^{-\lambda_{i,k}}))] - \sum_{k=1}^{K}[\sum_{i:\tau_i>k} (\lambda'_{i,k} - \lambda_{i,k})] \tag{24}$$

$$= \sum_{k=1}^{K}[\sum_{i:\tau_i=k} (\log(1 - e^{-\lambda'_{i,k}}) - \log(1 - e^{-\lambda_{i,k}}))] - \sum_{k=1}^{K}[\sum_{i:\tau_i>k} (\mu' - \mu)] \tag{25}$$

The second sum in (24) contains a lot more terms than the first one, and each term in the sum requires the calculation of $\lambda'_{i,k}, \lambda_{i,k}$. However, in (25) we see the second sum is a very simple expression that takes much less time to compute.

Results

The above simplification was implemented in the program by Alexander together with Xin. We first did not implement the simplification in (25) and we kept σ fixed at 1, since we just wanted a rough estimate of how much less time the new program would take to complete 11000 iterations (burn-in of 1000). The program was run on a subset of the data - a 10 by 10 subgrid of the whole grid containing 120 plants. It took around 15 minutes to complete (compared to the time of about 90 minutes for the original program). For the smaller grid $N = 120, K = 30$, thus $N < K^2$ and the asymptotic improvement in running time should be $O(N)$ - which was not the case. Nevertheless, the improvement in running time was significant. However, we would still expect the program to take over a day to complete on the full grid, and we therefore decided to further simplify the likelihood function.

8.2. **Kernel Truncation.** We decided to truncate the kernel of the Gaussian function f. In particular, we used a new model with a simpler expression for $\lambda_{i,k}$:

$$\lambda_{i,k} = \mu + \theta \sum_{j:\tau_j<k \ \& \ D(i,j)\leq 4\sigma} f(D(i,j)) \tag{26}$$

This significantly simplified the calculation of $\lambda_{i,k}$, since now in the summation of $f(D(i,j))$ we would need to only consider the plants S_j "close enough" to plant S_i instead of all 1742 plants.

In particular, if $\sigma \approx 1$, all plants S_j at distance at most 4σ from S_i would be contained in a 8×8 square with center at the location of plant S_i. The horizontal distances between the plants are about 1.5, while the vertical ones are 0.5, therefore the plants contained in that square form approximately a 5×17 grid, containing 85 plants - which is around 20 times less than the total number of plants.

The implementation of the truncated kernel in the program was non-trivial. Originally, the plant locations were essentially stored in a two-dimensional array (the x-coordinates were stored in one array, and the y-coordinates in the other). Thus, for a fixed plant S_i and fixed σ, to obtain the list of all plants S_j with $D(i,j) \leq 4\sigma$, we would still need to traverse the whole array, thus taking $O(N)$ time, providing no improvement over the program version with no truncation in terms of running time complexity.

Xin came up with the following solution. Before starting the MCMC algorithm, for each plant S_i, we would create an array $SortedPlant_i$, which contained all plants $j \in \mathcal{X}$, sorted by their distance from plant S_i (with $SortedPlant[0]$ being plant S_i, $SortedPlant[1]$ being the second closest plant to S_i, etc.) Then, when the calculation of $\lambda_{i,k}$ was necessary for some fixed i, k, we would traverse the plants in $SortedPlant_i$ until we came across a plant S_j such that $D(i,j) > 4\sigma$, at which point the loop would stop. This idea was proposed and implemented by Xin.

(Note that I attempted a different solution at first, but it was not as efficient, and the code implementation contained a few mistakes. This was then modified to arrive at Xin's solution to the problem.)

Results

Xin ran the new version of the program on the full grid for 11000 iterations (this time including the estimation of σ), and it took around 90 minutes to complete. She then also implemented the simplification for μ proposal using (25), and the resulting program took just 55 minutes to complete - an improvement by a factor of around 2880 over the estimated time of 110 days it would take for the original program to complete the same number of iterations (note the original program kept σ fixed at 1). The program is included in Appendix 2.

The Inverse Gamma priors for the parameters were set to be:

$$\mu \sim IG(1, 0.01), \theta \sim IG(1, 0.05), \sigma \sim IG(1, 1)$$

The values of σ_i used for parameter proposals were set to be:

$$\sigma_1 := 0.01, \sigma_2 := 0.002, \sigma_3 := 0.1$$

I ran the program again. My computer is slower than Xin's, and the program took 2.5 hours to complete. The results of the run are presented below. For each parameter estimated, we list the estimated value, acceptance rate, varfact, and standard error. The burn-in used was $B = 1000$.

TABLE 4. Results of Estimation for Discrete Model

Parameter	Estimate	r_{acc}	Varfact	Std. Error
μ	0.003091912	0.207818	5.352486	0.0000201076
θ	0.041815	0.265182	32.33533	0.0002403047
σ	1.088274	0.448455	34.94238	0.0056628540

The average acceptance rate of τ_i was 0.734830 (averaging over all τ_i). We do not include the estimated values for τ_i, since there are 1742 of them. We also present the time it took to complete each portion of the algorithm:

- Sorting plants to produce the arrays $SortedPlant_i$: 32.521 seconds.
- Updating mu: 100.077 seconds.
- Updating theta: 4079.273 seconds.
- Updating sigma: 727.8630 seconds.
- Updating τ_i: 4182.659 seconds.

Finally, we have included the trace plots for μ, σ and σ in the parameters in Appendix 1. They look quite good.

8.3. **Continuous Model.** Now let us go back to the continuous model. To estimate the parameters in the model, we use the same Metropolis within Gibbs Algorithm as in the discrete model, however the proposal for τ_i' is different. One-by-one, for each $i \in \mathcal{X}$, we propose $\tau_i' \sim N(\tau_i, \sigma_4^2)$. We then accept (i.e. set $\tau_i(n) := \tau_i'$) iff $\log(U) < \log \pi' - \log \pi_{old}$, where $\log \pi$ is defined as in (7). We demonstrate how $\log \pi' - \log \pi_{old}$ can be simplified.

Suppose $\tau_x' = \tau_x$ for all plants S_x except for some plant S_z, for which $\tau_z \leq K$. Suppose $\tau_z' > \tau_z$ (the case $\tau_z' < \tau_z$ is similar). Then:

$$\log \pi(\mu, \theta, \sigma, \{\tau_h'\}) - \log \pi(\mu, \theta, \sigma, \{\tau_h\}) = - \sum_{x:\tau_x' > K} [K\mu + \theta \sum_{y:\tau_y' < K} (K - \tau_y') f(D(x, y))]$$

$$+ \sum_{x:\tau_x' \leq K} [-K\tau_x' - \theta \sum_{y:\tau_y' < \tau_x'} (\tau_x' - \tau_y') f(D(x, y)) + \log(\mu + \theta \sum_{y:\tau_y' < \tau_x'} f(D(x, y)))]$$

$$+ \sum_{x:\tau_x > K} [K\mu + \theta \sum_{y:\tau_y < K} (K - \tau_y) f(D(x, y))]$$

$$- \sum_{x:\tau_x \leq K} [-K\tau_x - \theta \sum_{y:\tau_y < \tau_x} (\tau_x - \tau_y) f(D(x, y)) - \log(\mu + \theta \sum_{y:\tau_y < \tau_x} f(D(x, y)))]$$

$$= \sum_{x:\tau_x > \tau_z'} \theta(\tau_z' - \tau_z) f(D(x, z)) + \sum_{x:\tau_z < \tau_x \leq \tau_z'} \theta(\tau_x - \tau_z) f(D(x, z)) \tag{27}$$

$$- \mu(\tau_z' - \tau_z) + \sum_{x:\tau_z \leq \tau_x < \tau_z'} [-\theta(\tau_z' - \tau_x) f(D(x, z))] + \sum_{x:\tau_x < \tau_z} [-\theta(\tau_z' - \tau_z) f(D(x, z))] \tag{28}$$

$$+\log(\lambda'_{z,\tau'_z}) - \log(\lambda_{z,\tau_z}) + \sum_{x:\tau_z<\tau_x\le\tau'_z} [\log(\lambda'_{x,\tau'_x}) - \log(\lambda_{x,\tau_x})] \tag{29}$$

In (27) the first term corresponds to change in log likelihood for plants x, for which $\tau_x > \tau'_z$, to get infected; the second - the change in log likelihood for plants satisfying $\tau_z < \tau_x \le \tau'_z$. The terms in (28) correspond to the change in log likelihood for plant z. (29) contains the terms for which λs change since the values λ_{x,τ_x} change only for plant z and for plants x for which $\tau_z < \tau_x \le \tau'_z$. Note that:

$$\lambda'_{z,\tau'_z} = \mu + \theta \sum_{x:\tau_x<\tau_z} f(D(x,z)) + \theta \sum_{x:\tau_z\le\tau_x<\tau'_z} f(D(x,z));\ \lambda_{z,\tau_z} = \mu + \theta \sum_{x:\tau_x<\tau_z} f(D(x,z))$$

$$\Rightarrow \lambda'_{z,\tau'_z} = \lambda_{z,\tau_z} + \theta \sum_{x:\tau_z\le\tau_x<\tau'_z} f(D(x,z))$$

and

$$\lambda'_{x,\tau'_x} = \mu + \theta \sum_{y:\tau'_y<\tau_x} f(D(x,y));\ \lambda_{x,\tau_x} = \mu + \theta \sum_{y:\tau_y<\tau_x} f(D(x,y))$$

so that $\lambda_{x,\tau_x} = \lambda'_{x,\tau'_x} + \theta f(D(x,z))$.

Finally, we note that the portion of the algorithm in which a new value of μ is proposed can be simplified in essentially the same way as with the discrete model.

Results

I modified the final version of the program for the discrete model to implement the Metropolis within Gibbs Sampler for the continuous model, including the simplification described above. The program ran in only 40 minutes on my computer (and would thus take even less time on Xin's computer.) The reason why the program ran faster for the continuous version than for the discrete one can be attributed to the fact that the expression for $\log\pi$ in (7) is simpler than in (6). However, we realized that the expression in (6) can be rewritten in the same form as (7), which would give faster computation time of $\log\pi$ and hence even faster execution of the discrete version for the program. The program is included in Appendix 2.

We note that the continuous model is more realistic than the discrete one. Because the parameter estimates are similar for both models, and the computation time is rather small for both models (after all simplifications have been implemented), it is more desirable to use the continuous model.

The values of σ_i used for parameter proposals were set to be:

$$\sigma_1 := 0.005, \sigma_2 = 0.0005, \sigma_3 = 0.07, \sigma_4 = 1$$

The estimation results are presented below. For each parameter estimated, we list the estimated value, acceptance rate, varfact, and standard error. We see that the estimates and standard errors for the continuous version are close to those for the discrete one.

TABLE 5. Results of Estimation for Continuous Model

Parameter	Estimate	r_{acc}	Varfact	Std. Error
μ	0.003108482	0.567091	16.37271	0.0000172784
θ	0.03929908	0.403364	30.68109	0.0002306296
σ	1.103936	0.530273	34.75518	0.00582527

The average acceptance rate of τ_i was 0.824523 (averaging over all τ_i). We do not include the estimated values for τ_i, since there are 1742 of them. We also present the time it took to complete each portion of the algorithm:

- Sorting plants to produce the arrays $SortedPlant_i$: 35.882 seconds.
- Updating mu: 551.147 seconds.
- Updating theta: 546.323 seconds.
- Updating sigma: 553.214 seconds.
- Updating τ_i: 712.049 seconds.

We have included the trace plots for μ, σ and σ in the parameters in Appendix 1. They look quite good. Finally, we have also included a histogram of the estimated values of τ_i in Appendix 1 (we have only included values for which $\hat{\tau}_i < K$). We see that for each $i \in \mathcal{X}$ and $\hat{\tau}_i < K$, the estimated value $\hat{\tau}_i$ tends to be close to the midpoint of the interval in which the plant got infected, i.e. the value $\frac{L_i+U_i}{2}$. This is not surprising, since we are looking at the mean of $\tau_i(n)$ for $n = B+1, B+2, \ldots, M$, and from the observed infection states it seems that the plants become infected in a relatively even manner. We note however that while the distribution of the simulated values $\tau_i(n)$ may be rather interesting, the more important part in this problem are the values of the model parameters μ, θ, σ.

We also ran the program (included in Appendix 2) estimating the parameters of the continuous time model using Metropolis within Gibbs (and using simplifications) but no kernel truncation for 1000 iterations, with a burn-in of 100. The program ran in around 122.7 minutes, which would correspond to around 1349.7 minutes (or 22.5 hours) for 11000 iterations, about 33.7 times slower than for the continuous time version with kernel truncation. The estimates were close to those for the version with truncated kernel. (For greater confidence in our results, we would need to run the program for more iterations, e.g. 11000 instead of 1000). The similarity in estimated values suggests that using the truncated kernel is appropriate for this problem.

The estimation results are presented below. For each parameter estimated, we list the estimated value, acceptance rate, varfact, and standard error.

The average acceptance rate of τ_i was 0.825757 (averaging over all τ_i). We do not include the estimated values for τ_i, since there are 1742 of them. We also present the time it took to complete each portion of the algorithm. Not surprisingly, most of the time was spent on updating τ_i:

- Updating mu: 916.023 seconds.
- Updating theta: 914.963 seconds.

Table 6. Results of Estimation for Continuous Model, no Truncation

Parameter	Estimate	r_{acc}	Varfact	Std. Error
μ	0.003062761	0.576000	7.921256	0.00003762047
θ	0.03817787	0.379000	30.61165	0.0007866118
σ	1.131010	0.533000	31.73272	0.01966360

- Updating sigma: 914.369 seconds.
- Updating τ_i: 4615.493 seconds.

9. Conclusion

We have discussed our work on the problem of estimating the parameters in Spatial Infectious-Disease Models from Guadeloupean sugar cane plant infection data, using MCMC methods. We started with estimating a relatively basic discrete time model, but found our method of estimation, Metropolis within Gibbs Algorithm, too inefficient. Particle MCMC was then considered as an alternative, but was found to be too complex for this problem, especially from the computational point of view - as we would probably need a lot of particles to obtain a decent approximation of marginal likelihood.

We then went back to the MCMC with Gibbs Sampling, and realized that it was possible to significantly simplify the computation of $\log \pi' - \log \pi_{old}$ when new values of τ_i were proposed (one-by-one), as well as when a new value of μ was proposed. Computation time was further reduced by truncating the kernel of the Gaussian function included in the likelihood function expression, leading to a significant reduction of the number of terms that had to be computed. The implementation of the simplification was relatively straightforward, while the implementation of the truncated kernel idea required a programming trick, which was eventually successfully completed by Xin. The final version of the program for the discrete model ran in 55 minutes on Xin's computer, which is 2280 times faster than the original estimated time of 110 days that it would take to complete 11000 iterations for the original program.

Finally, we modified our program to estimate the continuous infectious-disease model. We found that the calculation of $\log \pi$ could be significantly simplified, and the final version of the program took 40 minutes to run on my computer (which is slower than Xin's computer). The parameter estimates were similar to the estimates produced for the discrete time model. They were also similar to the estimates produced for the continuous time model with no kernel truncation, suggesting that using the truncated kernel model is appropriate for this problem.

We were happy that we managed to reduce computation time a lot, and achieved the desired estimation of the continuous time model in a short period of time. We believe that it may be possible to use a similar approach to perform inference in an even more complicated infectious disease model. One possible extension of our current continuous model is to consider $\lambda_{i,t}$ dependent on time. Furthermore, we could consider time-varying rate at which the infection spreads from an infected plant (instead of just a constant θ) - for example, if the plant is infected for a longer period of time, the infection on it is more severe, and the rate θ at which the infection spreads from the plant is greater.

10. Acknowledgements

This project was jointly completed by Alexander Remorov, Patrick Brown, Florencia Chimard, Jeffrey Rosenthal, and Xin Wang. When I started the project this term, the data was already collected, the discrete version of the model was written down, and the C program for estimating the model using the Metropolis within Gibbs Sampler with no simplification was written down, as well as the R program for producing the display of the canes and the R program for summarizing the estimation results. The description of the model, the data, and the original estimation approach are described in Jeffrey Rosenthal's Course Notes for the STA3431 Monte Carlo Methods Course (2010) [8]. The three programs were also presented during the course by Jeffrey Rosenthal (2010) [9].

Throughout the course, the program for estimating the discrete time model was written together by Alexander and Xin, with Alexander and Xin checking each other's work on the program to ensure no mistakes were made. Patrick Brown wrote some initial code to estimate the continuous time model, and this was then combined with the final version of the program for estimating the discrete time model to obtain the final version of the code to estimate the continuous time model - which was written by Alexander.

Some parts of the introduction in the report are based on the starting manuscript of the paper about this project (the part of the manuscript completed so far has been written by Patrick Brown).

Throughout the term, prof. Brown, prof. Rosenthal, Xin, and Alexander met on a weekly basis and exchanged regular email communication with the ideas on how to proceed with the project. Thus the ideas and results presented in this individual report are the product of a joint effort.

References

[1] Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

[2] G. Gibson. Markov chain monte carlo methods for fitting spatiotemporal stochastic models in plant epidemiology. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 46(2):215–233, 1997.

[3] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear and non-gaussian bayesian state estimation. *Radar and Signal Processing, IEE Proc. F.*, 140:107–113, 1993.

[4] E. L. Ionides, C. Bret, and A. A. King. Inference for nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 103(49):18438–18443, 2006.

[5] M. Keeling and P Rohani. *Modeling infectious diseases in humans and animals.* Princeton University Press, 2008.

[6] Marco Läubli. Particle markov chain monte carlo for partially observed markov jump processes. 2011.

[7] S. Meyer, J. Elias, and M. Höhle. A space-time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, 2011.

[8] Jeffrey Rosenthal. Sta3431 (monte carlo methods) lecture notes, winter 2010. 2010.

[9] Jeffrey Rosenthal. Sta3431 (monte carlo methods) programs, winter 2010. 2010.