Delft University of Technology

Bachelor Project

AM3000

Modelling Covid-19 in Delft using epidemiological models and Markov Chain Monte Carlo

Het modelleren van Covid-19 in Delft met behulp van epidemiologische modellen en Markov Chain Monte Carlo

Author:
Willemijn Bergen (5181925)

Thesis comittee:
Dr. Ir. J. Bierkens
Drs. I.A.M. Goddijn

July 5, 2022

Summary

The spread of Covid-19 is modelled in this Bachelor thesis. This is done using two compartmental epidemiological models: the SIR-model and SIRS-model. These models divide the population into three groups, namely Susceptible, Infected and Recovered. Artificial data is generated based on the stochastic version of the models. The models are applied to this artificial data and to real data of a Covid-19 wave in Delft. Bayesian statistics are used to estimate the parameters of these models. This is done by using a Markov Chain Monte Carlo (MCMC) algorithm on the different data sets. The models are compared using the Root Mean Squared Error and the Bayes factor. For modelling the Covid-19 wave as it occurred in Delft, the SIRS-model is found to be preferred over the SIR-model. This preference is based on the Root Mean Squared Error (RMSE) and the Bayes Factor. Based on a MCMC simulation, the posterior mean value of the basic reproduction number is estimated to be 1.067. This value indicates a spread of the disease instead of it dying out.

Contents

Chapter 1

Introduction

After numerous confirmed cases around the globe, on the 27th of February 2020, the first Covid-19 case in the Netherlands was confirmed. This virus has had an enormous impact on the world. There have been lockdowns and other measures to prevent the spread of this virus. These measures are taken partly based on how many people get infected. Therefore, it is necessary to find out how fast people get infected and how fast people recover.

In this research, the speed of spreading of Covid-19 is looked into. As an example, this is done for the city of Delft in the Netherlands. The used data covers a time period of 77 days, from the 25th of January 2022 until the 12th of May 2022. The goal of the research is to be able to say something about the spread of Covid-19 in the future.

To be able to say something about all these thing, a mathematical model is introduced. First, the SIR-model is used, a mathematical model to determine the spread of a disease in a population. Bayesian statistics will be used to estimate the parameters. This will be implemented using an MCMC algorithm. This MCMC algorithm makes estimations based on Bayesian statistics. First, MCMC is applied to artificial data, to check if the algorithm gives the right parameters, and then it is applied to real data. Second, a more extensive model, the SIRS-model is used. For this model the MCMC algorithm is used as well, both on the real and artificial data. This model takes into account that people are not immune forever and therefore can get reinfected, which is also the situation for Covid-19 [1]. Finally, a conclusion will be drawn about how fast Covid-19 spreads.

1.1 Notation

Throughout this thesis, the following notation will be used.

S	Number of susceptible people in deterministic model
I	Number of infected people in deterministic model
R	Number of recovered people in deterministic model
N	total population size
X_t	Number of susceptible people in stochastic model
Y_t	Number of infected people in stochastic model
β	Infection parameter
α	Recovery parameter
ϕ	Noise parameter
ξ	Parameter loss of immunity

Chapter 2

The SIR-model

2.1 Introduction

To understand the process of infectious disease transmission, a mathematical model is needed. That is why, in this chapter, the SIR-model is presented. The SIR-model is a compartmental epidemiological model. This is a simple model used to analyze the spread of infectious diseases. First, the deterministic model will be discussed. Second, a stochastic approach to the SIR-model is described.

2.2 The deterministic SIR-model

The deterministic SIR-model is a very general model of the spread of infectious diseases. It divides the population in three groups: Susceptible, Infected and Recovered. The model is described by a system of three coupled differential equations. In this model there are some assumptions made. People can only go from susceptible to infected and from infected to recovered. Once a person had the virus, they either die or are immune. Also, the population size is constant, as can be seen in 2.4. This means that people who die from the virus are counted in the recovered group. Next to this, the population is assumed homogeneous, so there are no people that are more or less likely to become infected or recover. The parameters in the model are constant. Finally, the number of people in this model is assumed continuous for easier computations. This can be seen in the differential equations describing the model:

$$\dot{S} = -\frac{\beta S(t) I(t)}{N} \tag{2.1}$$

$$\dot{I} = \frac{\beta S(t) I(t)}{N} - \alpha I(t) \tag{2.2}$$

$$\dot{R} = \alpha I(t) \tag{2.3}$$

$$S(t) + I(t) + R(t) = N \tag{2.4}$$

Equation 2.1 describes the change in susceptible people. The constant β in this equation influences the magnitude of people who were susceptible and become infected. Also, the change is influenced by the number of infected people and the number of susceptible people itself. Equation 2.2 describes the change in infected people. There are people who become infected and people who recover. The magnitude of people who become infected is influenced by the relations described above. The number of people that recover is determined by the constant α and the number of infected people. So, the values of S, I and R on time $t+1$ are dependent on

parameters α and β and on the values of S, I and R on time t. This can be seen as the following relation:

$$(S_{t+1}, I_{t+1}, R_{t+1}) \sim f(\alpha, \beta, S_t, I_t, R_t)$$

2.3 The stochastic SIR-model

The stochastic SIR-model is different than the deterministic one. It describes the probabilities of a person to become infected and of a person to recover. Similar notation as in the deterministic model is used. The population is still of size N. The assumption is made that the epidemic process has a lack-of-memory property and therefore is Markov. This means that the time that someone becomes infected or recovers is exponentially distributed, since the exponential distribution is the only continuous distribution with this property. The model is the process (X_t, Y_t), where X_t are the susceptible people at time t and Y_t are the infected people at time t. This stochastic model for infectious diseases is also known as the General Stochastic Epidemic (GSE). Based on the deterministic model, this gives the following rates for the exponential distributions:

$$(i, j) \to (i - 1, j + 1) : \frac{\beta}{N} X_t Y_t$$
$$(i, j) \to (i, j - 1) : \alpha Y_t$$

With these rates and given the current history at time t, the transition probabilities are:

$$\mathbb{P}[X_{t+\delta t} - X_t = -1, Y_{t+\delta t} - Y_t = 1 | \mathcal{H}_t] = \frac{\beta}{N} X_t Y_t \delta t + o(\delta t) \tag{2.5}$$

$$\mathbb{P}[X_{t+\delta t} - X_t = 0, Y_{t+\delta t} - Y_t = -1 | \mathcal{H}_t] = \alpha Y_t \delta t + o(\delta t) \tag{2.6}$$

$$\mathbb{P}[X_{t+\delta t} - X_t = 0, Y_{t+\delta t} - Y_t = 0 | \mathcal{H}_t] = 1 - \frac{\beta}{N} X_t Y_t \delta t - \alpha Y_t \delta t + o(\delta t) \tag{2.7}$$

In these probabilities $\mathcal{H}_t$ is the history of the process at time t and $o(\delta t)$ is an error term that will go to zero as δt goes to zero. Using the law of large numbers, it can be found that the stochastic model converges to the deterministic model as $N \to \infty$.

2.4 Summary

The SIR-model is a model for disease transmission. It divides a population into three groups: Susceptible, Infected and Recovered. In the deterministic model the changes of these groups are described as differential equations. In the stochastic model they are described as probabilities. In both models there are two parameters, β and α, influencing the changes. β influences number of people that get infected and α the number of people that recover.

Chapter 3

Statistical methodology

3.1 Introduction

In order to get information from the data, a statistical methodology is needed. This statistical methodology will give more information about how the data relates to the actual values of the parameters in the SIR-model.

3.2 Statistical method

The statistical method that will be used is Bayesian statistics. Namely, a distribution based on prior knowledge and the likelihood is used. This follows from Bayes' theorem for continuous variables:

$$p(\theta|X) = \frac{p(X|\theta)}{\int p(X|\theta)p(\theta)\,d\theta} \tag{3.1}$$

In this equation θ represents vector of the parameters and X the data. The notation $p()$ represents the probability density function and $X|\theta$ stands for X given our knowledge of θ. In this equation we call $p(\theta|X)$ the posterior distribution, $p(\theta)$ the prior distribution and $p(X|\theta)$ the likelihood, which is the statistical model for the data. So, actually the belief of the distribution of a parameter is updated with the knowledge of the data. To actually calculate the integral in Equation 3.1, the probability that the data is true, becomes harder in higher dimensions. Since it is constant, what is used for this model is that the posterior is proportional to the prior times the likelihood up to this normalisation factor:

$$p(\theta|X) \propto p(X|\theta)p(\theta) \tag{3.2}$$

To relate this to the data, the likelihood and prior distributions should be determined.

3.3 Likelihood

First, the likelihood should be considered. It is assumed that the observations are the number of people that are ill at a each time, where continuous time is assumed. The Negative Binomial distribution with an expected value of $I_{observed,t}|I_{actual,t}$ and a spread around this expected value of ϕ is chosen for the sampling distribution. This is a discrete distribution, which also holds for the infected students. Next to this, the codomain of this distribution is $\mathbb{R}^+$. There could be a difference between the observations of people that are ill and the actual people that are ill. This distribution can deal with the difference between these two numbers in a correct way. Namely, through an extra parameter ϕ, which determines the spread around the expected value. This

gives the likelihood with parameters $\theta = (\beta, \alpha, \phi)$. Mathematically, this can be written in the following way:

$$p(I_{observed,t}|\theta) \sim NegBin(I_{observed,t}|I_{actual,t}, \phi)$$

3.4 Prior distribution

Now, the prior distributions have to be chosen. This has to be done for the two parameters in the SIR-model and one is needed for the ϕ in the likelihood. For the α and β a normal distribution is chosen and for ϕ an exponential distribution. Specifically, the following distributions:

$$\beta \sim N(2, 1)$$

$$\alpha \sim N(0.4, 0.5)$$

$$\phi \sim exp(5)$$

These are chosen based on the prior belief that the β has a value of approximately 2, the α of approximately 0.4 and the ϕ of a value around 0. The beliefs about the values of α, β and ϕ are based on the information from [9]. These priors show the belief that the expectation of these variables is to be larger than zero. The prior distribution helps with determining the posterior distribution, but the more data there is available, the less influence the prior distribution has. So, it is necessary to have enough data in order for the posterior distribution to become less subjective.

3.5 Summary

In order learn more about the distribution of the parameters, a Bayesian statistics approach is used. Firstly, the sample distribution, a negative binomial, was chosen. Then, the used prior distributions for the parameters were discussed; a normal distribution for α and β and an exponential for ϕ. Now that the model is described, in the following chapter the generation of artificial data will be discussed.

Chapter 4

Data generation

4.1 Introduction

A model used on data should give the correct results. In order to check this, artificial data can be generated. This will give an indication on how good the model can determine the parameters. In this section it will be explained how data, based on the stochastic SIR-model as described in section 2.3, can be generated.

4.2 Method

The programming language used to create the data set is R. The steps to create this data set are explained in this section. First, all the initial values need to be chosen, the parameters β and α and the population size N. In this example the following values are used: $\alpha = 0.5$, $\beta = 4$ and $N = 100$. As described in the model, the algorithm starts with a susceptible population size of $N - 1$ and an infected population size of 1.

The used algorithm is explained below. This algorithm is repeated until there are 0 people left in the group of susceptible. Then people can not become ill anymore, just recover. The same algorithm is used, but the possibility for recovery is left out. Which means that ΔT will be ΔT_2. This continues until there is 1 individual left in the infected group. A total of 199 data points is created for the population of $N = 100$.

- $X_0 = N - 1, Y_0 = 1$, $T_0 = 0$
- $\Delta T_1 \sim Exp(\frac{\beta}{N} X_0 Y_0)$, $\Delta T_2 \sim Exp(\alpha Y_0)$
- $\Delta T = \min(\Delta T_1, \Delta T_2)$
- If $\Delta T = \Delta T_1$ someone becomes ill or if $\Delta T = \Delta T_2$ someone recovers.
- If someone becomes ill:

$$X_1 = \begin{cases} X_0 & , t < \Delta T \\ X_0 - 1 & , t = \Delta T \end{cases}$$

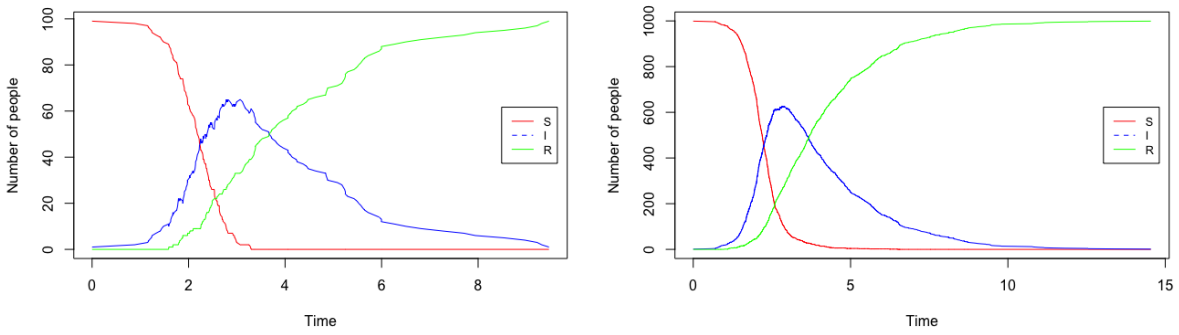
$$Y_1 = \begin{cases} Y_0 & , t < \Delta T \\ Y_0 + 1 & , t = \Delta T \end{cases}$$

- If someone recovers:

$$X_1 = \begin{cases} X_0 & ,t < \Delta T \\ X_0 & ,t = \Delta T \end{cases}$$
$$Y_1 = \begin{cases} Y_0 & ,t < \Delta T \\ Y_0 - 1 & ,t = \Delta T \end{cases}$$

- $T_1 = T_0 + \Delta T$

The same is done for a population of 1000 persons in total. This gives 1995 data points. The result of these computations can be seen in Figure 4.1a and Figure 4.1b. As can be seen in these Figures, the graph for the larger population size looks more smooth because there are more data points. The programmed algorithm can also be found in Appendix A.1.



(a) Results for population size $N = 100$

(b) Results for population size $N = 1000$

Figure 4.1: Stochastic SIR-model, with $\alpha = 0.5$ and $\beta = 4$. On the x-axis the time is displayed and on the y-axis the number of infected people.

As we will see in Figures 5.2a and 5.2b for the larger population size, the generated data will converge to the solution of the deterministic model.

4.3 Summary

Data can be generated based on the stochastic SIR-model. This was done with an algorithm in R. The algorithm was used to create data for two population sizes, $N = 100$ and $N = 1000$. The results were plotted for the two different population sizes and can be seen in Figure 4.1.

Chapter 5

MCMC on artificial data

5.1 Introduction

Now that there is a data set, the Bayesian method can be used to estimate the parameters. This is done using the `Stan` package in R. This is a package which uses Bayesian modelling to obtain samples from the posterior distribution. The package uses Markov Chain Monte Carlo (MCMC), specifically Hamiltonian Monte Carlo. This method will be explained in the upcoming section.

5.2 What is MCMC?

First, it is important to understand the basics about the Markov Chain Monte Carlo (MCMC) method. With MCMC, a sample is drawn and this sample is accepted or rejected based on some acceptance criterion. This results in a large number of samples, which in the end should resemble the distribution from which is drawn, the posterior distribution. However, what makes MCMC different is that it works better than other algorithms for problems in spaces with higher dimensions, because of the fact that it chooses the next sample based on the previous one instead of an independent sample. This allows to sample directly from the not normalized part of the posterior [12]. Because of this, the acceptance probability is a lot higher with higher dimensions than with for example random sampling.

The MCMC algorithm produces a Markov Chain. A Markov Chain is a stochastic process in which the state at $t+1$ only depends on the state at t, so not on the states at $t-1$, $t-2$, etc. To find out which distribution is sampled from, the Markov Chain is useful. To find the distribution, it is needed that the distribution is invariant with respect to the Markov Chain. This means that the distribution does not change in the Markov Chain process, which is called that the chain is homogeneous. Also, the Markov Chain should converge to the actual distribution. This happens when the Markov Chain is irreducible and aperiodic. These two things are ensured when choosing the sampling algorithm.

An example of an MCMC algorithm is Metropolis-Hastings. This algorithm draws a sample z^* from the distribution $q_k(z|z^t)$, where z^t is the current state at time t. This sample is accepted with the following probability:

$$A_k(z^*, z^t) = \min\left(1, \frac{\tilde{p}(z^*)q_k(z^t|z^*)}{\tilde{p}(z^t)q_k(z^*|z^t)}\right) \tag{5.1}$$

In this equation, k represent the possible transitions that are taken into account. The probability distribution that is looked for, $p(z) = \tilde{p}(z)/Z_p$, with normalization constant Z_p and

unnormalized distribution $\tilde{p}(z)$ is invariant. This can be proved by showing that it satisfies detailed balance. The is done by using the acceptance probability.

$$\begin{aligned} p(z)q_k(z|z')A_k(z',z) &= \min(p(z)q_k(z|z'), p(z')q_k(z'|z)) \\ &= \min(p(z')q_k(z'|z), p(z)q_k(z|z')) \\ &= p(z')q_k(z'|z)A_k(z,z') \end{aligned} \tag{5.2}$$

This shows that detailed balance is satisfied and therefore p(z) is an invariant distribution with respect to the Markov Chain produced by this algorithm. This Metropolis acceptance criterion is also used in the MCMC algorithm discussed next.

The used MCMC sampling algorithm in `Stan` is Hamiltonian Monte Carlo. This algorithm uses Hamiltonian Dynamics to create the Markov Chain. Hamilton Dynamics takes the state and the momentum into account. The momentum is the rate in which the state changes. The position combined with the momentum is called the phase space. The dynamics of this phase space can be described using the Hamiltonian function H, which is the sum of the kinetic and potential energy of the system:

$$H = E(z) + K(r) \tag{5.3}$$

The potential energy is dependent on the state z and the kinetic energy on the momentum r. With the help of differentiation, it can be seen that the value of H is constant. Next to H being constant, also the volume of the phase space is constant. These two things ensure that Hamiltonian dynamics will not change the distribution, what was a necessary condition. The calculations to show that these two things remain constant and more in dept information about Hamiltonian Dynamics can be found in the book 'Pattern Recognition and Machine Learning' [6]. Also, in this book the way of discretization of the Hamilton Dynamics is described. This is done using the leapfrog discretization, which involves updating the discrete-time approximations of the momentum and position variable. To overcome any biases from the discretization, the sample is accepted or rejected according to the Metropolis criterion. This criterion leads to this acceptance probability:

$$\min(1, \exp\{H(z,r) - H(z^*, r^*)\} \tag{5.4}$$

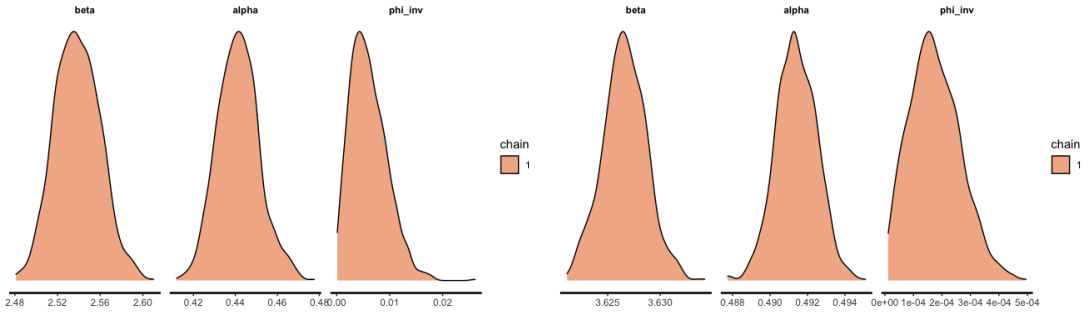
In which r is the momentum variable and r^* is the updated momentum variable. z is the position variable and $z*$ the updated one. This algorithm allows to take large enough steps, but still has a relatively high acceptance chance in comparison with other MCMC algorithms.

5.3 MCMC on the artificial data

Now that the mathematics behind the algorithm are discussed, it can be used on the artificial data set. As mentioned before, the `Stan` package uses Hamiltonian MCMC to draw the samples. The Stan code for the SIR model is mainly retrieved from an article [9]. The program draws 1000 samples as a warm up and then 1000 for the actual sample. The results of these 1000 samples are visualized in Figures 5.1a and 5.1b. For $N = 100$, the result gives an estimation of the distribution of β around 2.53 and for α around 0.44. So, the change in susceptible people is approximately $-\frac{2.53}{100}S(t)I(t)$ and the amount of people recovering is $0.44I(t)$. For ϕ, it is centered around 0.005, which is small. For $N = 1000$, ϕ is distributed around an even smaller number. So, the difference between the actual infected students and the data sample is probably also smaller. This could also be due to the fact that there is more data than with the sample size of 100. Also, the distributions of α and β are narrower. So, the samples are closer to each other. This indicates that the samples are close to the true value.

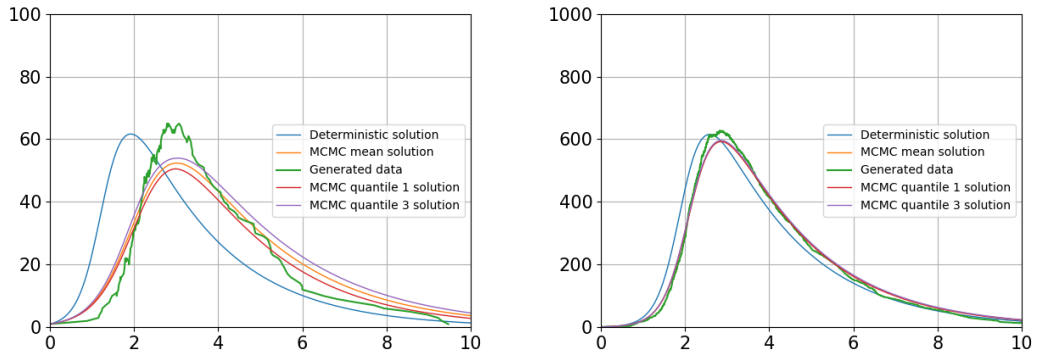
In Figures 5.2a and 5.2b the mean outcome of the MCMC is plotted. With MCMC, 1000 samples for α and β are drawn. These samples are used in the deterministic model. The

deterministic model is numerically integrated using the method of Euler. Taking the mean, first quantile and third quantile of these 1000 outcomes for every time t gives these graphs. As can be seen in Figure 5.2, the MCMC matches the shape of the data better for the larger population size. This was to be expected, since the larger population size has more data points, which should lead to a more accurate determination of the parameters. For the population of 100 persons, the data does not match the MCMC sample mean as good as for the population of 1000 persons.



(a) Result for population size $N = 100$ (b) Result for population size $N = 1000$

Figure 5.1: Result of the MCMC sampling on the artificial data with $\alpha = 0.5$ and $\beta = 4$



(a) Result for population size $N = 100$ (b) Result for population size $N = 1000$

Figure 5.2: Artificial data, mean & quantiles of the 1000 MCMC samples and deterministic solution for $\alpha = 0.5$ and $\beta = 4$. On the x-axis the time is displayed and on the y-axis the number of infected people.

5.4 Summary

Markov Chain Monte Carlo (MCMC) is a method to draw samples from an unknown posterior distribution. The method produces a Markov Chain, where the next sample is only dependent on the current sample and not on earlier samples. This makes the acceptance chances higher. The MCMC algorithm used in the `Stan` package is Hamiltonian Monte Carlo. Based on Hamiltonian Dynamics the next sample is chosen. This algorithm has a relatively high acceptance chance, while still being able to take large enough steps. Using this MCMC on the artificial data, it can be seen that the algorithm predicts the variables well. Also, it can be seen that the stochastic

solution becomes more like the deterministic one, when choosing a larger population size and therefore having more data points. Now, the MCMC algorithm can be used on the real data.

Chapter 6

MCMC on real data

6.1 Introduction

Now that the model is tested on artificial data, it can be applied to real data. As mentioned in the Chapter 1, the spread of Covid-19 in Delft will be researched in the hope to be able to say something about the future spread of the disease. The MCMC algorithm is applied to data about the reported cases and from this hopefully conclusions can be drawn.

6.2 Data

First it is necessary to discuss what data is used. The choice of data can influence the estimated parameters in many ways. There should be enough data points to make the estimation. The external factors influencing the spread of Covid-19 should also be kept in mind. That is why the daily reported Covid-19 cases in Delft from the 25th of February until the 12th of May are chosen. This gives 77 data point. This data is retrieved from the Ministry of Health, Welfare and Sport, see [2]. The 25th of February is chosen as a starting point, because at this date almost every rule to suppress Covid-19 was ended in the Netherlands [5]. This includes, but is not limited to, the end of the 1.5-meter rule, the end of wearing face coverings in shops and the end of the corona test / corona vaccination certificate. The 1.5-meter rule made it mandatory to maintain 1.5 meter distance between two persons, except if they live in the same household. Because of the ending of these rules, the effect of the legislation to prevent Covid-19 on the parameters in the model should be limited. The ending date is chosen, because the data was retrieved at that day, so there was not more data available at that point. Next to this, the Covid-19 restrictions did not change a lot during this time period.

Another factor still has to be considered when interpreting the estimations, namely the vaccination rate. On the 22th of May 2022 83.2% of the population is vaccinated and 63.9% has had their booster vaccination [3]. These vaccination rates ensure that the infection rate is lower, so it affects β.

Next to the Covid-19 cases, the population size should also be chosen correctly. A population size of 104574 residents was measured on 1-1-2022 by the municipality of Delft [4]. This population size is chosen, because the Covid-19 cases were also reported per municipality. Also, the amount of residents is measured once a year by the municipality, so there was no data available for the exact same time period as the Covid-19 cases. From this population, it was assumed that 148 people have Covid-19 at time $t = 0$. This is assumed, because this were the reported infections on the 24th of February and there is no precise information available about how many people have Covid-19 at a point in time. For this I_0, it should be taken into account

that also this value actually is uncertain, which is not added in the model. Since the 25th of February was not the start of the pandemic, the I_0 of this wave of Covid-19 could also be seen as a random variable that could be estimated. This also holds for S_0 and R_0. This is also included as one of the topics that could be looked into for further research.

6.3 Results

Again, using the `Stan` package in R, the MCMC algorithm is applied. In Figure 6.1 the results of the sampling for β, α and ϕ can be seen. For β the results are centered around 1.35 and for α around 1.28. The change in susceptible people is thus approximately $-\frac{1.35}{104574}S(t)I(t)$. This is added to the pool of infected people and approximately $1.28I(t)$ is subtracted and added to the recovered people.

To be able to say something about the fit of the model the Root Mean Squared Error (RMSE) is calculated. This is done in the following way:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i - \hat{y}}$$

In this calculation y_i is the data point from the actual Covid-19 cases in Delft and $\hat{y}$ is the estimated data point using the mean of the deterministic solutions of the 1000 MCMC samples. The RMSE for this model is approximately 53.9.

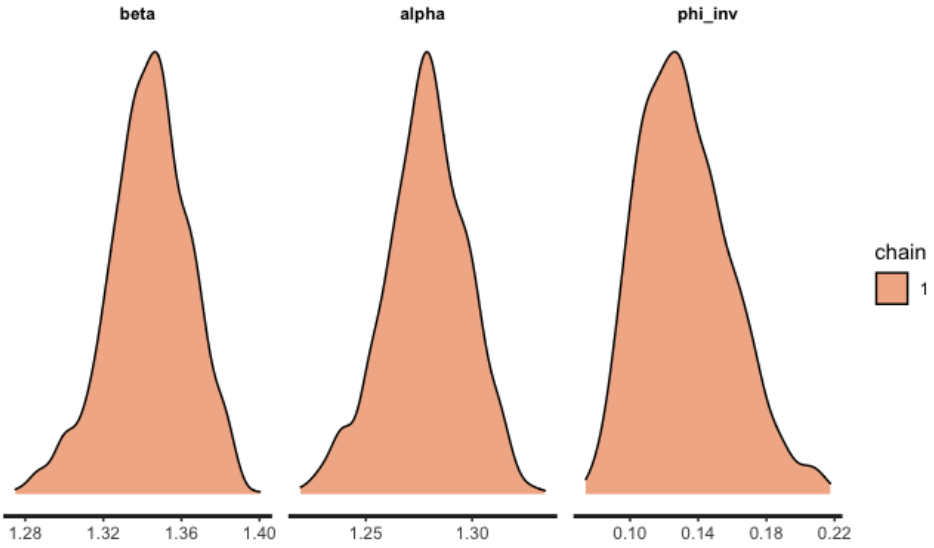


Figure 6.1: MCMC results on the data of reported Covid cases in Delft 25th of February 2022 until 12th of May 2022.

6.4 Summary

The data of reported Covid-19 cases in Delft from the 25th of February until the 12th of May were chosen. The population size N was 104574 as measured on 1-1-2022 by the municipality of Delft. For I_0, it is thought to be 148, since this was the number of reported Covid-19 cases on

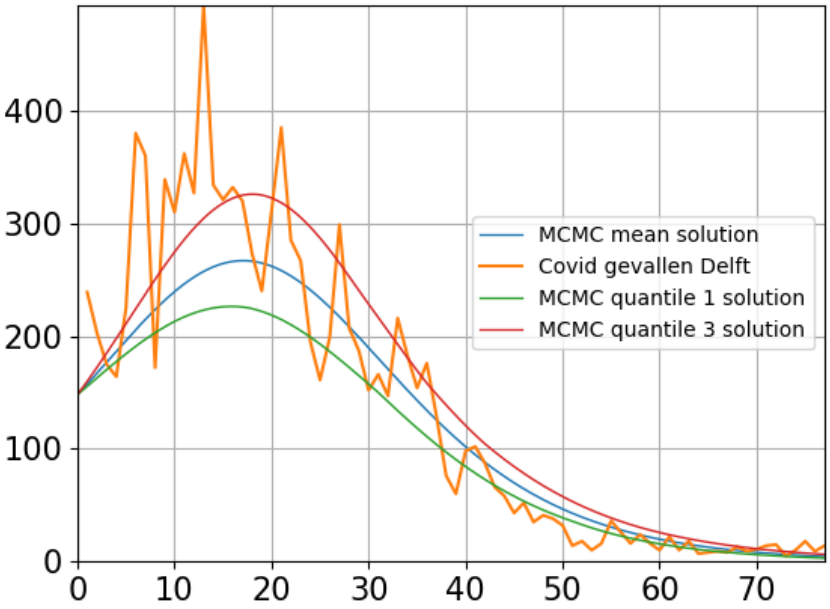


Figure 6.2: Data of reported Covid cases in Delft 25th of February until 12th of May 2022 and the mean, 1st quantile and 3rd quantile of the deterministic solution of the MCMC results in the SIR model.

the 24th of January. By doing the MCMC sampling on the data, this resulted in a β of around 1.35, an α of around 1.28 and an noise term ϕ of around 0.14. The RMSE was calculated to be 53.9. Now that these calculations were done, this can be done as well for the SIRS-model to be able to compare the two models.

Chapter 7

SIRS-model

7.1 Introduction

The Ministry of Health, Welfare and Sport mentions that people can get infected with the coronavirus more than once and that it is unsure how long people stay immune after being infected [1]. This is not taken into account in the SIR-model. In the SIR-model people are immune forever and do not get infected again. To get the model to resemble the true situation more, the decrease in immunity and therefore the possibility to get reinfected, should be added. That is why the SIRS-model is introduced.

7.2 SIRS-model

The new model is the SIRS-model. The difference between this model and the SIR-model is that people can go from recovered to susceptible again. So, the immunity after an infection is not forever like in the SIR-model. This changes one of the assumptions, but the other assumptions made for the SIR-model remain the same. As done in Chapter 2, first the deterministic model and then the stochastic model will be discussed. Then as in Chapter 3 artificial data will be created in order to perform the MCMC algorithm on this data as done in Chapter 4. Finally, the MCMC algorithm will be applied to the real data of Delft.

7.2.1 The deterministic SIRS-model

As mentioned, the difference between this model and the SIR-model is that people can go from recovered to susceptible again. This translates to people not being immune forever, but for some period of time. A parameter ξ is created that represents this loss of immunity. The changes are added to the model by substracting $\xi R(t)$ from $\dot{R}$ and adding it to $\dot{S}$ as can be seen in Equations 7.1, 7.2 and 7.3. This makes it possible for people to go from Recovered to Susceptible again. The rest of the factors are explained in Chapter 2 and again the population remains of constant size.

$$\dot{S} = -\frac{\beta S(t) I(t)}{N} + \xi R(t) \tag{7.1}$$

$$\dot{I} = \frac{\beta S(t) I(t)}{N} - \alpha I(t) \tag{7.2}$$

$$\dot{R} = \alpha I(t) - \xi R(t) \tag{7.3}$$

$$S(t) + I(t) + R(t) = N \tag{7.4}$$

So, S, I and R at time $t+1$ are dependent on the parameters α, β and ξ and on S, I and R on time t. This can be seen as the following relation:

$$(S_{t+1}, I_{t+1}, R_{t+1}) \sim f(\alpha, \beta, \xi, S_t, I_t, R_t)$$

7.2.2 The stochastic SIRS-model

Now, the changes on the stochastic model can be discussed. Instead of two, there are now three possible events that can happen. Someone can go from susceptible to infected, from infected to recovered and from recovered to susceptible. Therefore, like in Chapter 2, the model can be described as process (X_t, Y_t), where X_t are the susceptible people and Y_t the infected people at time t. For the new model, this gives the following rates for the exponential distributions:

$$(i,j) \to (i-1, j+1) : \frac{\beta}{N} X_t Y_t$$
$$(i,j) \to (i, j-1) : \alpha Y_t$$
$$(i,j) \to (i+1, j) : \xi(N - X_t - Y_t)$$

Like in Chapter 2 these rates and given the current history at time t, the transition probabilities are:

$$\mathbb{P}[X_{t+\delta t} - X_t = -1, Y_{t+\delta t} - Y_t = 1 | \mathcal{H}_t] = \frac{\beta}{N} X_t Y_t \delta t + o(\delta t) \tag{7.5}$$

$$\mathbb{P}[X_{t+\delta t} - X_t = 0, Y_{t+\delta t} - Y_t = -1 | \mathcal{H}_t] = \alpha Y_t \delta t + o(\delta t) \tag{7.6}$$

$$\mathbb{P}[X_{t+\delta t} - X_t = 1, Y_{t+\delta t} - Y_t = 0 | \mathcal{H}_t] = \xi(N - X_t - Y_t) + o(\delta t) \tag{7.7}$$

$$\mathbb{P}[X_{t+\delta t} - X_t = 0, Y_{t+\delta t} - Y_t = 0 | \mathcal{H}_t] = 1 - \frac{\beta}{N} X_t Y_t \delta t - \alpha Y_t \delta t - \xi(N - X_t - Y_t) + o(\delta t) \tag{7.8}$$

7.3 Data generation

Like in Chapter 4, first, data is generated based on the stochastic SIRS-model as described in 7.2.2 in order to check whether the model estimates the parameters correctly. Instead of two possible events, there are now three possible events. Someone can get infected, someone can recover and someone can lose their immunity. This gives the algorithm described below.

- $X_0 = N - 1, Y_0 = 1$, $T_0 = 0$
- $\Delta T_1 \sim Exp(\frac{\beta}{N} X_0 Y_0)$, $\Delta T_2 \sim Exp(\alpha Y_0)$, $\Delta T_3 \sim Exp(\xi(N - X_t - Y_t))$
- $\Delta T = \min(\Delta T_1, \Delta T_2, \Delta T_3)$
- If $\Delta T = \Delta T_1$ someone becomes ill, if $\Delta T = \Delta T_2$ someone recovers and if $\Delta T = \Delta T_3$ someone loses their immunity.
- If someone becomes ill:

$$X_1 = \begin{cases} X_0 & , t < \Delta T \\ X_0 - 1 & , t = \Delta T \end{cases}$$
$$Y_1 = \begin{cases} Y_0 & , t < \Delta T \\ Y_0 + 1 & , t = \Delta T \end{cases}$$

- If someone recovers:

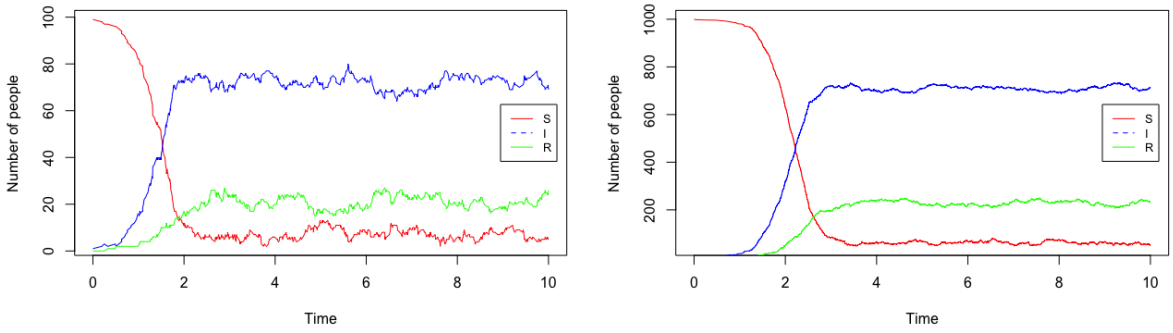
$$X_1 = \begin{cases} X_0 & ,t < \Delta T \\ X_0 & ,t = \Delta T \end{cases}$$
$$Y_1 = \begin{cases} Y_0 & ,t < \Delta T \\ Y_0 - 1 & ,t = \Delta T \end{cases}$$

- If someone loses their immunity:

$$X_1 = \begin{cases} X_0 & ,t < \Delta T \\ X_0 + 1 & ,t = \Delta T \end{cases}$$
$$Y_1 = \begin{cases} Y_0 & ,t < \Delta T \\ Y_0 & ,t = \Delta T \end{cases}$$

- $T_1 = T_0 + \Delta T$

The data generated by this algorithm gives a solution that looks like the SIR-model, but now the model does not converge to a state where everyone is recovered, but to a state where the three parts of the population remain about constant. For $N = 100$ this gives 674 data points and for $N = 1000$ in 5979 data points for t from 0 to 10. In Figures 7.1a and 7.1b the results are plotted for a population of $N = 100$ and $N = 1000$.



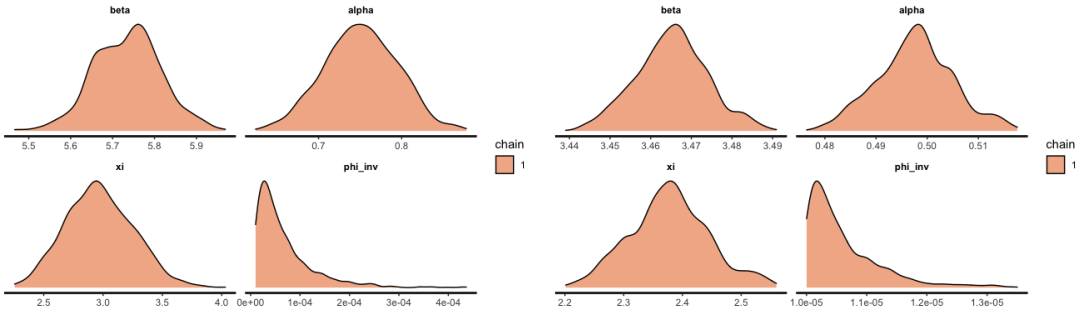
(a) Results for population size $N = 100$

(b) Results for population size $N = 1000$

Figure 7.1: Stochastic SIRS-model, with $\alpha = 0.5$, $\beta = 4$ and $\xi = 0.5$. On the x-axis the time is displayed and on the y-axis the number of infected people.

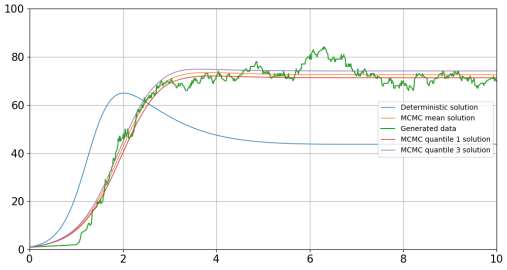
7.4 MCMC results on artificial data

Just like in Chapter 5, the parameters of the generated data are estimated using MCMC. In Figure 7.2 it can be seen that the estimation of the MCMC parameters on the data looks good. However, the generated data seems not to converge to the stochastic model. There is also reviewed whether this is due to the small initial infected population, but when changing this to a larger initial infected population, the model does not seem to be converging to the solution of the differential equations either. This could be due to several reasons, which are not looked into due to time constraints on this research.

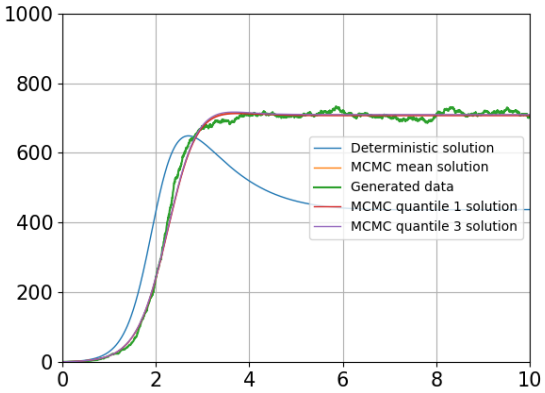


(a) Result for population size $N = 100$ (b) Result for population size $N = 1000$

Figure 7.2: Result of the MCMC sampling on the artificial data with $\alpha = 0.5$, $\beta = 4$ and $\xi = 0.5$



(a) Results for population size $N = 100$



(b) Results for population size $N = 1000$

Figure 7.3: Artificial data, mean & quantiles of the 1000 MCMC samples and deterministic solution for $\alpha = 0.5$, $\beta = 4$ and $\xi = 0.5$. On the x-axis the time is displayed and on the y-axis the number of infected people.

7.5 MCMC results on real data

The SIRS-model is also applied to the real data of Delft. The same data was used as in Chapter 6. The results of the MCMC sampling can be found in Figure 7.4. β is centered around approximately 1.70, α around 1.60 and ξ around 0.012. β and α are both estimated higher than in the SIR-model. This means that more people of the susceptible population get ill and more people of the ill population recover. This looks like a logical consequence of the change in the model, since the population of susceptible people grows with $\xi R(t)$ for every time step t. Also, since $\xi > 0$ there are actually people that lose their immunity. However, the ϕ term is small compared to the α and β term. The noise term ϕ is centered around 0.07. This is smaller than for the SIR-model, which leads to believe that the SIRS-model is a better predictor for the real situation.

To find out whether this model actually fits the data better, the RMSE is calculated again. The RMSE is approximately 42.8, which is lower than for the RMSE for the SIR-model, which was 53.92. From this it could be concluded that the model fits the data better. However, as can be seen from the quantile solutions and from the width of the samples, this model does have

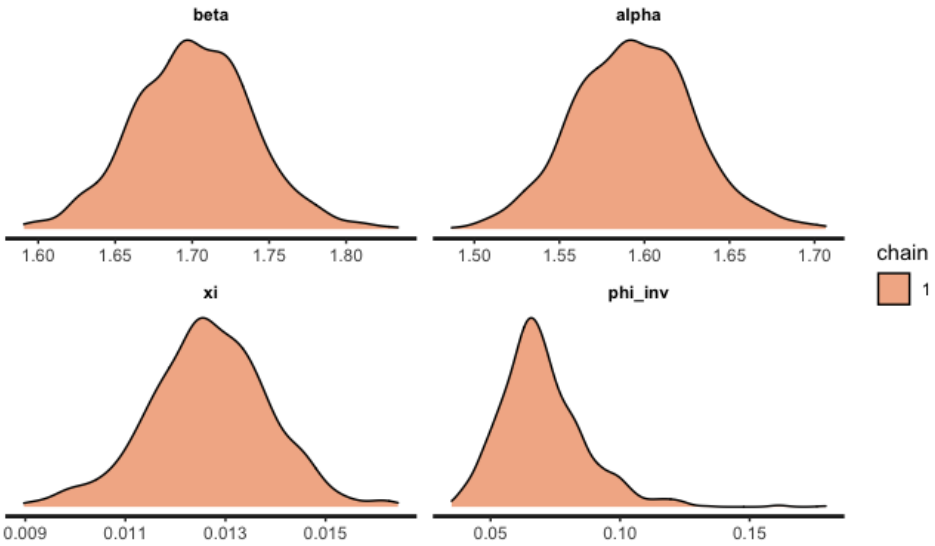


Figure 7.4: MCMC results on the data of reported Covid cases in Delft 25th of February 2022 until 12th of May 2022.

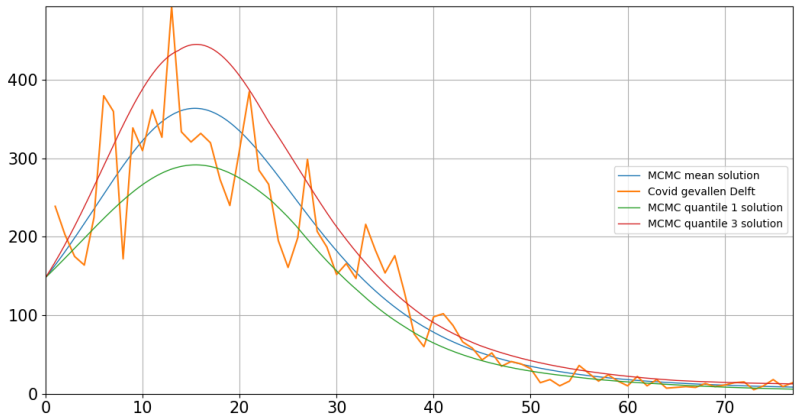


Figure 7.5: Data of reported Covid cases in Delft 25th of February until 12th of May 2022 and the mean, 1st quantile and 3rd quantile of the deterministic solution of the MCMC results in the SIRS model.

more uncertainty than there was in the SIR-model.

Next to the RMSE, to compare the models also the Bayes factor was determined. Since, in this research Bayesian statistics is used to estimate the parameters, this will also be used to compare the two models. This can be done with this Bayes factor. The Bayes factor is a way to compare the likelihoods of two models given the data. It is calculated in the following way:

$$\text{Bayes factor} = \frac{\mathbb{P}[\text{data } | \text{Model 1}]}{\mathbb{P}[\text{data } | \text{Model 2}]}$$

The R package `bridgesampling` was used for this calculation. The Bayes factor of the SIRS-model over the SIR-model is 14931959. This is really high. When looking at the classification

scheme of Jeffreys [10], it can be seen that there is very strong evidence for the SIRS-model instead of the SIR-model.

Finally, since this model is found to be more likely than the SIR-model. The basic reproduction number (R0) is calculated to be able to say something about the spread of Covid-19. This number says how many cases are expected to be caused by an individual during their infectious period in a completely susceptible population. If the number is larger than 1, it means the disease will most likely not die out. A larger R0 indicates a harder to control disease. In another research, this number was also calculated for the coronavirus in Sri Lanka to find out how contagious the disease is [7]. R0 is calculated as follows:

$$\mathrm{R0} = \frac{\beta}{\alpha}$$

The value of R0 is calculated for all the 1000 MCMC samples. The outcomes of these values can be found in Figure 7.6. The mean of these 1000 R0 values is 1.067. This is larger than 1, which means that the spread of Covid-19 will not die out in this population. However, since the value is close to one, it could be said that for this time period of Covid-19 the spread was controllable.

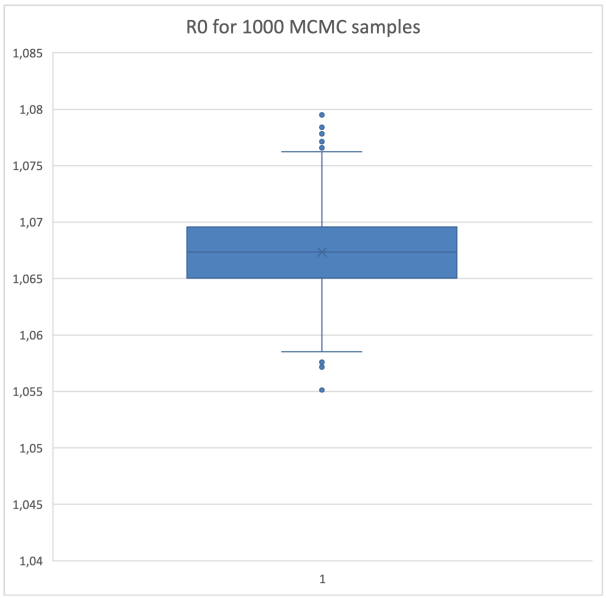


Figure 7.6: The value of R0 for the 1000 MCMC samples shown in a boxplot

7.6 Summary

In attempt to find a model that fits the real situation of Covid-19 better, the SIRS-model was tested. This model is similar to the SIR-model, but people can become susceptible again after recovering. This can be explained by the fact that immunity does not last forever for Covid-19. For this new model there was also data generated stochastically, but this data did not seem to converge to the deterministic model. Next, the MCMC algorithm was applied to this data, which looks to be estimating the data good. Finally, the MCMC algorithm was applied to the real Covid-19 data and a higher α and β were found. Also, the new ξ variable was larger than zero. The noise term ϕ was smaller than in the SIR-model, which could imply that the SIRS-model fits the real situation better. To test this the RMSE and the Bayes factor were calculated.

The RMSE was lower than of the SIR-model, so the error between the model and the data was smaller. The Bayes factor compares the likelihood of the two models and this resulted in strong evidence for the SIRS-model. Finally, the reproduction number R0 was calculated for the 1000 MCMC samples and this gave a number of 1.067, which means it is not likely that the disease will die out.

Chapter 8

Conclusion and Discussion

The goal of this research was to be able to draw a conclusion about the spread of Covid-19. For this purpose two models were used. Both of the models are compartmental models in epidemiology. The statistical methodology behind the parameter estimation was discussed. Next, artificial data was created based on the stochastic versions of these models. The MCMC algorithm was applied to estimate the parameters of this artificial data and also to estimate the parameters of real data of the municipality of Delft. The two models were compared based on the RMSE and the Bayes factor. Finally, the decision for the SIRS-model was made and for this model the R0 was calculated to be able to draw a conclusion about the spread of the disease.

First, based on the stochastic SIR-model, artificial data was created. This data showed that the MCMC algorithm estimated the parameters of the data good, especially for a larger population size. Then, the MCMC algorithm was used on real data, namely the daily reported Covid-19 cases in Delft from the 25th of February until the 12th of May. 2000 samples were drawn, 1000 as a warm-up and 1000 samples were used. This gave as a result a β parameter around 1.35 and a α parameter around 1.28. The noise term of around 0.13 was already small. One thing that should be taken into account is that the current data was chosen because of the minimal restrictions, but the data is approximately one Covid-19 wave and the SIR-model also describes one wave of infections. When the data is chosen differently, this might produce worse results. The 1000 samples were put in to the deterministic solution of the model and then the mean and quantiles were determined of these 1000 solutions. To be able to compare the two models, the RMSE between the data and the mean of the MCMC solutions was also calculated and had the value 53.9.

Next, all the above steps were also taken for the SIRS-model. Artificial data was generated, but this data seemed not to converge to the deterministic model with the same parameters. However, the algorithm does actually estimate the parameters of the SIR-model good and the solutions with the estimated parameters lies close to the data. So, there is assumed that the MCMC does estimate the parameters correctly and that there might be a problem in the data generation. In this model, β is centered around 1.70, α around 1.60 and ξ around 0.012. The RMSE is approximately 42.8, which is lower than the RMSE of the SIR-model. This indicates that the solution of the MCMC for this model compared to the data produces a smaller error than the previous model. Even though the graphs of both solutions looked similar. To compare the models, the Bayes factor was also calculated. It had a value of 14931959, which is really high. This factor provides a way, based on Bayesian statistics, to compare the likelihood of two models. The value of the Bayes factor indicates that the SIRS-model is more likely. Therefore, the SIRS-model should be preferred over the SIR-model to research the spread of Covid-19.

Now that the SIRS-model is chosen over the SIR-model, a conclusion about the spread

of Covid-19 can be drawn. As mentioned above, in the SIRS-model a β around 1.70, an α around 1.60 and a ξ around 0.012 were found. This means that the spread of Covid-19 would approximately follow these equations:

$$\dot{S} = -\frac{1.70S(t)I(t)}{N} + 0.012R(t)$$

$$\dot{I} = \frac{1.70S(t)I(t)}{N} - 1.60I(t)$$

$$\dot{R} = 1.60I(t) - 0.012R(t)$$

To be able to say something about the continuation of the spread, R0 was calculated. This had a mean of 1.067 for the 1000 MCMC samples. The disease will therefore not be likely to die out. Hence, Covid-19 in Delft will probably continue to spread. However, there should be considered that this number is calculated for a period of 77 days with a relatively simple model. That is why in the upcoming section the possibilities for further research will be mentioned.

8.1 Further Research

There are several things that could be done for further research. One of these things is to change the assumption of both models that the parameters are constant. This is already done for the SIR-model used on Covid-19 data [11], but not for the SIRS-model. Not only the parameters could be not constant, also the population size could change, since a population is affected by births and deaths. However, one should always watch out for overfitting. Another research that has already been done is using the SIRS-model with distributed delays, which makes the solution for the Infected people oscillating [8]. This could be interesting to research for the Covid-19 data, because the spread of Covid-19 also comes in waves. Next to changing the assumptions on the models, one of the possibilities for further research is to not only include the parameters in the MCMC algorithm, but also the initial values. This could be interesting, because, when choosing a point in time, it is not always known with certainty what the initial values are at that point.

Bibliography

[1] Coronavirus disease covid-19. https://www.rivm.nl/node/157841. Accessed: 28-05-2022.

[2] Covid-19 dataset. https://data.rivm.nl/covid-19/. Accessed: 11-05-2022.

[3] Covid-19-vaccinaties. https://coronadashboard.rijksoverheid.nl/landelijk/vaccinaties. Accessed: 27-05-2022.

[4] Data op maat. https://delft.incijfers.nl/. Accessed: 12-05-2022.

[5] Tijdlijn van coronamaatregelen. https://www.rivm.nl/gedragsonderzoek/tijdlijn-maatregelen-covid. Accessed: 12-05-2022.

[6] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 01 2006.

[7] Samath Dharmaratne, Supun Sudaraka, Ishanya Abeyagunawardena, Kasun Manchanayake, Mahen Kothalawala, and Wasantha Gunathunga. Estimation of the basic reproduction number (r0) for the novel coronavirus disease in sri lanka. *Virology Journal*, 17(1):144, 2020.

[8] Sebastian Gonçalves, Guillermo Abramson, and Marcelo Ferreira da Costa Gomes. pages 363–371, 12 2009.

[9] Léo Grinsztajn, Elizaveta Semenova, Charles C. Margossian, and Julien Riou. Bayesian workflow for disease transmission modeling in stan. *Statistics in Medicine*, 40(6500), 07 2021.

[10] H. Jeffreys. *Theory of Probability*. Oxford University Press, 3rd edition, 1961.

[11] Gustavo A. Muñoz-Fernández, Jesús M. Seoane, and Juan B. Seoane-Sepúlveda. A sir-type model describing the successive waves of covid-19. *Chaos, Solitons Fractals*, 144:110682, 2021.

[12] Joseph Rocca and Baptiste Rocca. Bayesian inference problem, mcmc and variational inference. https://towardsdatascience.com/bayesian-inference-problem-mcmc-and-variational-inference-25a8aa9bce29, 07 2019. Accessed: 06-06-2022.

Appendix A

Code SIR model

A.1 R code data generation

```
X <- c(1:2500)*0
Y <- c(1:2500)*0
deltaT <- c(1:2500)*0


X[1] <- 99
Y[1] <- 1
beta <- 4
alpha <- 0.5


i <-1
while (X[i]>0 & Y[i]>0){
  lamda1 <- beta*X[i]*Y[i] * (1/(X[1]+Y[1]))
  y1_rexp <- rexp(1, lamda1)
  lamda2 <- alpha*Y[i]
  y2_rexp <- rexp(1, lamda2)
  mindeltaT <- min(y1_rexp[1], y2_rexp[1])
  deltaT[i+1] <- deltaT[i]+mindeltaT
  if (y1_rexp[1] < y2_rexp[1]){
    X[i+1]<-X[i]-1
    Y[i+1]<-Y[i]+1
  } else{
    X[i+1]<-X[i]
    Y[i+1]<-Y[i]-1
  }
  i <- i+1
}

while (Y[i]>1 & X[i]==0) {
  lamda2 <- alpha*Y[i]
    y2_rexp <- rexp(1, lamda2)
    X[i+1]<-X[i]
    Y[i+1]<-Y[i]-1
    deltaT[i+1] <- deltaT[i]+y2_rexp
    i <- i+1
}

I <- c(1:i)*0
S <- c(1:i)*0
delta_T <- c(1:i)*0
```

```r
for (z in 1:i) {
  I[z] <- Y[z]
  S[z] <- X[z]
  delta_T[z] <- deltaT[z]
}

plot(delta_T, S, type = "l", col="red", xlab = 'Time', ylab = 'Number of people')
lines(delta_T,I, type = "l", col="blue")
lines(delta_T, X[1]+Y[1]-S-I, type = 'l', col="green")
legend("right", inset = 0.02, legend = c("S", "I", "R"), col=c("red", "blue", "green"), lty=1:2, cex = 0.8)

library("writexl")


df <- data.frame(Time = delta_T
                 )
df2 <- data.frame(Infected = I[-1]
                 )
df3 <- data.frame(Infected = I, Time = delta_T
                 )
write_xlsx(df, "bep_dataT.xlsx")
write_xlsx(df2, "bep_dataY.xlsx")
write_xlsx(df3, "bep_data.xlsx")
```

A.2 R code Stan MCMC

```
functions {
  real[] sir(real t, real[] y, real[] theta,
             real[] x_r, int[] x_i) {

      real S = y[1];
      real I = y[2];
      real R = y[3];
      real N = x_i[1];

      real beta = theta[1];
      real alpha = theta[2];

      real dS_dt = -beta * I * S / N;
      real dI_dt =  beta * I * S / N - alpha * I;
      real dR_dt =  alpha * I;

      return {dS_dt, dI_dt, dR_dt};
  }
}
data {
  int<lower=1> n_days;
  real y0[3];
  real t0;
  real ts[n_days];
  int N;
  int cases[n_days];
}
transformed data {
  real x_r[0];
  int x_i[1] = { N };
}
```

```
parameters {
  real<lower=0> alpha;
  real<lower=0> beta;
  real<lower=0> phi_inv;
}
transformed parameters{
  real y[n_days, 3];
  real phi = 1. / phi_inv;
  {
    real theta[2];
    theta[1] = beta;
    theta[2] = alpha;

    y = integrate_ode_rk45(sir, y0, t0, ts, theta, x_r, x_i);
  }
}
model {
  //priors
  beta ~ normal(2, 1);
  alpha ~ normal(0.4, 0.5);
  phi_inv ~ exponential(5);

  //sampling distribution
  //col(matrix x, int n) - The n-th column of matrix x. Here the number of
    infected people
  cases ~ neg_binomial_2(col(to_matrix(y), 2), phi);
}
generated quantities {
  real R0 = beta / alpha;
  real recovery_time = 1 / alpha;
  real pred_cases[n_days];
  pred_cases = neg_binomial_2_rng(col(to_matrix(y), 2), phi);
}
```

A.3 R code MCMC Sampling

```

library(knitr)
opts_chunk$set(results="show",  # hide results
               fig.show="show", # hide figs
               warning=FALSE,   # do not show warnings
               message=FALSE,   # do not show messages
               eval=TRUE,       # evaluate code
               fig.width=5,     # set figure width, height and positioning
               fig.height=3.5,
               fig.align='center')

library(rstan)
library(gridExtra)
library(outbreaks)
library(tidyverse)
library(deSolve)
library("readxl")


my_dataY <- read_excel("~/Documents/bep_dataY.xlsx")
my_dataT <- read_excel("~/Documents/bep_dataT.xlsx")
my_data <- read_excel("~/Documents/bep_data.xlsx")
head(my_dataY)
```

```
head(my_dataT)
head(my_data)

theme_set(theme_bw())
ggplot(data = my_data) +
  geom_point(mapping = aes(x = Time, y = Infected)) +
  labs(y = "Number of students in bed")

# time series of cases
cases <- my_dataY$Infected # Number of students in bed
length(cases)

# total count
N <- 1000;

# times
n_days <- length(cases)
t <- my_dataT$Time
t0 = 0
t <- t[-1]

#initial conditions
i0 <- 1
s0 <- N - i0
r0 <- 0
y0 = c(S = s0, I = i0, R = r0)

# data for Stan
data_sir <- list(n_days = n_days, y0 = y0, t0 = t0, ts = t, N = N, cases =
    cases)

# number of MCMC steps
niter <- 2000

model <- stan_model(file = "sir_negbin.stan")

fit_sir_negbin <- sampling(model,
                 data = data_sir,
                 iter = niter,
                 chains = 1,
                 seed = 0
                 )

pars=c('beta', 'alpha', 'phi_inv')
print(fit_sir_negbin, pars = pars)
stan_dens(fit_sir_negbin, pars = pars, separate_chains = TRUE)

library(rstan)
library("writexl")
beta_and_gamma <- as.matrix(fit_sir_negbin, pars = c("beta", "alpha"), )
#print(summary(fit_sir_negbin, pars = c("beta", "gamma")))

data_beta_and_gamma <- data.frame(data = beta_and_gamma
                  )
write_xlsx(data_beta_and_gamma, "data_beta_and_gamma.xlsx")
```

A.4 Python code MCMC plotting

```

```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from math import *
import xlrd

# initializations

Dt = 0.001
I_init = 1
R_init = 0
S_init = 9
alpha= 0.5
beta= 4
N = S_init + R_init + I_init

t_init = 0                                  # starting time
t_end = 10                                  # stopping time
n_steps = int(round((t_end-t_init)/Dt)) # total number of timesteps
print(n_steps)

t_arr = np.zeros(n_steps + 1)               # create an array of zeros for t
S_arr = np.zeros(n_steps + 1)               # create an array of zeros for P
I_arr = np.zeros(n_steps + 1)
R_arr = np.zeros(n_steps + 1)

t_arr[0] = t_init
S_arr[0] = S_init
I_arr[0] = I_init
R_arr[0] = R_init

#data in plot


df = pd.read_excel ('~/Documents/bep_data.xlsx')
numpy_array = df.to_numpy()


#data samples
datasamples = pd.read_excel('~/Documents/data_beta_and_gamma.xlsx')
datasamples_array = datasamples.to_numpy()


# Euler's method


def SIR_function(beta, alpha):
    for i in range (1, n_steps + 1):
        S = S_arr[i-1]
        I = I_arr[i-1]
        R = R_arr[i-1]
        t = t_arr[i-1]

        dSdt = -beta/N*S*I                  # calculate the derivative
        dIdt = beta/N * S*I-alpha*I
        dRdt = alpha * I

        S_arr[i] = S + Dt*dSdt                # calculate P on the next time step
        I_arr[i] = I + Dt*dIdt
        R_arr[i] = R + Dt*dRdt
```

```
        t_arr[i] = t + Dt
    return  I_arr


# running SIR_function for all samples

SIR_samples = np.zeros((1000, n_steps+1))



for i in range(1000):
    I_array = SIR_function(datasamples_array[i,0],datasamples_array[i,1])
    SIR_samples[i:,] = I_array

column_means = SIR_samples.mean(axis=0)

I1_arr = SIR_function(beta, alpha)

fig = plt.figure()                          # create figure

xaxis = t_arr
y2axis = I1_arr

plt.plot(xaxis, y2axis, linewidth = 1, label = 'Deterministic solution')
plt.plot(xaxis, column_means, linewidth = 1, label = 'MCMC mean solution')
plt.plot(numpy_array[:,1], numpy_array[:,0], label = 'Generated data')
plt.legend(loc='center right')

plt.xticks(fontsize = 15)
plt.yticks(fontsize = 15)
plt.grid(True)                              # show grid
maxS=max(S_arr)
maxI=max(I_arr)
maxR=max(R_arr)
plt.axis([0, t_end, 0, S_init+I_init+R_init])               # define the axes

plt.show()                              # show the plot
```

Appendix B

Code SIRS model

B.1 R code data generation

```
X <- c(1:25000)*0
Y <- c(1:25000)*0
deltaT <- c(1:25000)*0

N <- 1000
X[1] <- 999
Y[1] <- 1
beta <- 4
alpha <- 0.5
xi <- 0.1


i <-1
while (deltaT[i]<10) {
  if (X[i]>0 & Y[i]>0 & (N-X[i]-Y[i])>0){
    lamda1 <- beta*X[i]*Y[i] * (1/N)
    y1_rexp <- rexp(1, lamda1)
    lamda2 <- alpha*Y[i]
    y2_rexp <- rexp(1, lamda2)
    lamda3 <- xi*(N-X[i]-Y[i])
    y3_rexp <- rexp(1, lamda3)
    mindeltaT <- min(y1_rexp[1], y2_rexp[1], y3_rexp[1])
    deltaT[i+1] <- deltaT[i]+mindeltaT
    if (y1_rexp[1] < y2_rexp[1] & y3_rexp[1]){
      X[i+1]<-X[i]-1
      Y[i+1]<-Y[i]+1
    } else if (y3_rexp[1] < y1_rexp[1] & y2_rexp[1]){
      X[i+1]<-X[i]+1
      Y[i+1]<-Y[i]
    } else{
      X[i+1]<-X[i]
      Y[i+1]<-Y[i]-1
  }
    i <- i+1
  } else if (X[i]>0 & Y[i]>0 & (N-X[i]-Y[i])==0){
    lamda1 <- beta*X[i]*Y[i] * (1/N)
    y1_rexp <- rexp(1, lamda1)
    lamda2 <- alpha*Y[i]
    y2_rexp <- rexp(1, lamda2)
    mindeltaT <- min(y1_rexp[1], y2_rexp[1])
    deltaT[i+1] <- deltaT[i]+mindeltaT
```

```
    if (y1_rexp[1] < y2_rexp[1]){
      X[i+1]<-X[i]-1
      Y[i+1]<-Y[i]+1
    } else{
      X[i+1]<-X[i]
      Y[i+1]<-Y[i]-1
    }
    i <- i+1
  } else if(Y[i]==0 & (N-X[i]-Y[i])>0){
    lamda3 <- xi*(N-X[i]-Y[i])
    y3_rexp <- rexp(1, lamda3)
    mindeltaT <- y3_rexp[1]
    deltaT[i+1] <- deltaT[i]+mindeltaT
    X[i+1]<-X[i]+1
    Y[i+1]<-Y[i]
    i <- i+1
  } else if (X[i]==0 & Y[i]>0 &(N-X[i]-Y[i])==0) {
    lamda2 <- alpha*Y[i]
      y2_rexp <- rexp(1, lamda2)
      X[i+1]<-X[i]
      Y[i+1]<-Y[i]-1
      deltaT[i+1] <- deltaT[i]+y2_rexp
      i <- i+1
  } else if (X[i]>0 & Y[i]==0 & (N-X[i]-Y[i])==0) {
    y_rexp = rexp(1,1)
    X[i+1]<-X[i]
    Y[i+1]<- Y[i]
    deltaT[i+1]<- deltaT[i]+y_rexp
    i <- i+1
  } else if (X[i]==0 & Y[i]>0 & (N-X[i]-Y[i])>0){
    lamda2 <- alpha*Y[i]
    y2_rexp <- rexp(1, lamda2)
    lamda3 <- xi*(N-X[i]-Y[i])
    y3_rexp <- rexp(1, lamda3)
    mindeltaT <- min(y2_rexp[1], y3_rexp[1])
    deltaT[i+1] <- deltaT[i]+mindeltaT
    if (y3_rexp[1] < y2_rexp[1]){
      X[i+1]<-X[i]+1
      Y[i+1]<-Y[i]
    } else{
      X[i+1]<-X[i]
      Y[i+1]<-Y[i]-1
    }
  i <- i+1
}
}

# while (Y[i]>1 & X[i]==0) {
#   lamda2 <- alpha*Y[i]
#     y2_rexp <- rexp(1, lamda2)
#     X[i+1]<-X[i]
#     Y[i+1]<-Y[i]-1
#     deltaT[i+1] <- deltaT[i]+y2_rexp
#     i <- i+1
# }

I <- c(1:i)*0
S <- c(1:i)*0
delta_T <- c(1:i)*0
for (z in 1:i) {
```

```
  I[z] <- Y[z]
  S[z] <- X[z]
  delta_T[z] <- deltaT[z]
}

plot(delta_T, S, type = "l", col="red", xlab = 'Time', ylab = 'Number of people')
lines(delta_T,I, type = "l", col="blue")
lines(delta_T, N-S-I, type = 'l', col="green")
legend("right", inset = 0.02, legend = c("S", "I", "R"), col=c("red", "blue", "green"), lty=1:2, cex = 0.8)

library("writexl")


df <- data.frame(Time = delta_T
                 )
df2 <- data.frame(Infected = I[-1]
                 )
df3 <- data.frame(Infected = I, Time = delta_T
                 )
write_xlsx(df, "bep_dataT2.xlsx")
write_xlsx(df2, "bep_dataY2.xlsx")
write_xlsx(df3, "bep_data2.xlsx")
```

B.2 R code Stan

```
functions {
  real[] sir(real t, real[] y, real[] theta,
             real[] x_r, int[] x_i) {

      real S = y[1];
      real I = y[2];
      real R = y[3];
      real N = x_i[1];

      real beta = theta[1];
      real alpha = theta[2];
      real xi = theta[3];

      real dS_dt = -beta * I * S / N + xi * R;
      real dI_dt =  beta * I * S / N - alpha * I;
      real dR_dt =  alpha * I - xi * R;

      return {dS_dt, dI_dt, dR_dt};
  }
}
data {
  int<lower=1> n_days;
  real y0[3];
  real t0;
  real ts[n_days];
  int N;
  int cases[n_days];
}
transformed data {
  real x_r[0];
  int x_i[1] = { N };
}
```

```
parameters {
  real<lower=0> alpha;
  real<lower=0> beta;
  real<lower=0> xi;
  real<lower=0> phi_inv;
}
transformed parameters{
  real y[n_days, 3];
  real phi = 1. / phi_inv;
  {
    real theta[3];
    theta[1] = beta;
    theta[2] = alpha;
    theta[3] = xi;

    y = integrate_ode_rk45(sir, y0, t0, ts, theta, x_r, x_i);
  }
}
model {
  //priors
  beta ~ normal(2, 1);
  alpha ~ normal(0.4, 0.5);
  xi ~ normal(0.4, 0.5);
  phi_inv ~ exponential(5);

  //sampling distribution
  //col(matrix x, int n) - The n-th column of matrix x. Here the number of
    infected people
  cases ~ neg_binomial_2(col(to_matrix(y), 2), phi);
}
generated quantities {
  real R0 = beta / alpha;
  real recovery_time = 1 / alpha;
  real pred_cases[n_days];
  pred_cases = neg_binomial_2_rng(col(to_matrix(y), 2), phi);
}
```

B.3 R code MCMC sampling

```
library(knitr)
opts_chunk$set(results="show",  # hide results
               fig.show="show", # hide figs
               warning=FALSE,   # do not show warnings
               message=FALSE,   # do not show messages
               eval=TRUE,       # evaluate code
               fig.width=5,     # set figure width, height and positioning
               fig.height=3.5,
               fig.align='center')

library(rstan)
library(gridExtra)
library(outbreaks)
library(tidyverse)
library(deSolve)
library("readxl")


# my_dataY <- read_excel("~/Documents/bep_dataY2.xlsx")
# my_dataT <- read_excel("~/Documents/bep_dataT2.xlsx")
```

```r
# my_data <- read_excel("~/Documents/bep_data2.xlsx")
# head(my_dataY)
# head(my_dataT)
# head(my_data)

# theme_set(theme_bw())
# ggplot(data = my_data) +
#   geom_point(mapping = aes(x = Time, y = Infected)) +
#   labs(y = "Number of students in bed")

my_data <- read_excel("~/Documents/Corona_gevallen_Delft_10january12may.xlsx")

head(my_data)


ggplot(data = my_data) +
  geom_point(mapping = aes(x = Date_of_publication, y = Total_reported)) +
  labs(y = "Number of Corona infection in Delft")



cases <- my_data$Total_reported


# total count
N <- 104426;
n_days <- length(cases)
t <- seq(0, n_days, by = 1)
t0 = 0
t <- t[-1]

#initial conditions
i0 <- 148
s0 <- N - i0
r0 <- 0
y0 = c(S = s0, I = i0, R = r0)

# data for Stan
data_sir <- list(n_days = n_days, y0 = y0, t0 = t0, ts = t, N = N, cases =
    cases)

# number of MCMC steps
niter <- 2000

model <- stan_model(file = "sirs_negbin.stan")
model2 <- stan_model(file = "sir_negbin.stan")

fit_sir_negbin <- sampling(model2,
                 data = data_sir,
                 iter = niter,
                 chains = 1,
                 seed = 0
                 )

fit_sirs_negbin <- sampling(model,
                 data = data_sir,
                 iter = niter,
                 chains = 1,
                 seed = 0
                 )
```

```r

pars=c('beta', 'alpha','xi', 'phi_inv')
print(fit_sirs_negbin, pars = pars)
stan_dens(fit_sirs_negbin, pars = pars, separate_chains = TRUE)

library(rstan)
library("writexl")
beta_and_gamma_xi <- as.matrix(fit_sirs_negbin, pars = c("beta", "alpha", "xi")
    , )
print(summary(fit_sirs_negbin, pars = c("beta", "alpha", "xi")))

data_beta_and_gamma_xi <- data.frame(data = beta_and_gamma_xi)
print(data_beta_and_gamma_xi)
write_xlsx(data_beta_and_gamma_xi, "data_beta_and_gamma_xi.xlsx")

library(bridgesampling)


# compute log marginal likelihood via bridge sampling for H0
H0.bridge <- bridge_sampler(fit_sir_negbin, silent = TRUE)

# compute log marginal likelihood via bridge sampling for H1
H1.bridge <- bridge_sampler(fit_sirs_negbin, silent = TRUE)

BF01 <- bf(H0.bridge, H1.bridge)
BF02 <- bf(H1.bridge, H0.bridge)
print(BF01)
print(BF02)
```

B.4 Python code MCMC plotting

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from math import *
import xlrd



# initializations

Dt = 0.01
I_init = 1 #148
R_init = 0
S_init = 99 # 104426
alpha1= 0.5
beta1= 4
xi1= 0.5
N = S_init + R_init + I_init

t_init = 0                                  # starting time
t_end = 10 ##77                                 # stopping time
n_steps = int(round((t_end-t_init)/Dt)) # total number of timesteps

t_arr = np.zeros(n_steps + 1)               # create an array of zeros for t
S_arr = np.zeros(n_steps + 1)               # create an array of zeros for P
I_arr = np.zeros(n_steps + 1)
R_arr = np.zeros(n_steps + 1)

```

```
t_arr[0] = t_init
S_arr[0] = S_init
I_arr[0] = I_init
R_arr[0] = R_init


#Generated data

df = pd.read_excel ('~/Documents/bep_data2.xlsx')
##df = pd.read_excel("~/Documents/Corona_gevallen_Delft_10january12may.xlsx")
numpy_array = df.to_numpy()

#data samples

datasamples = pd.read_excel('~/Documents/BEP/data_beta_and_gamma_xi.xlsx')
datasamples_array = datasamples.to_numpy()


# Euler's method


def SIR_function(beta, alpha, xi):
    for i in range (1, n_steps + 1):
        S = S_arr[i-1]
        I = I_arr[i-1]
        R = R_arr[i-1]
        t = t_arr[i-1]

        dSdt = -beta/N*S*I + xi*R                # calculate the derivative
        dIdt = beta/N * S*I-alpha*I
        dRdt = alpha * I - xi*R

        S_arr[i] = S + Dt*dSdt               # calculate P on the next time step
        I_arr[i] = I + Dt*dIdt
        R_arr[i] = R + Dt*dRdt
        t_arr[i] = t + Dt
    return  I_arr




# running SIR_function for all samples

SIR_samples = np.zeros((1000, n_steps+1))



for i in range(1000):
    I_array = SIR_function(datasamples_array[i,0],datasamples_array[i,1],
    datasamples_array[i, 2])
    SIR_samples[i:,] = I_array

column_means = SIR_samples.mean(axis=0)
quantileSIR1 = np.quantile(SIR_samples, 0, axis=0)
quantileSIR3 = np.quantile(SIR_samples, 1, axis=0)

Determenistic_solution = SIR_function(beta1, alpha1, xi1)

```

```
fig = plt.figure()                          # create figure

xaxis = t_arr

Time = np.arange(1, 78, dtype=int)




# Calculating MAE
MSE = 0
for i in range(1, t_end+1):
    MSE += (1/77)*(column_means[i*100]-numpy_array[i-1,1])**2
print('RMSE is' + str(sqrt(MSE)))




plt.plot(xaxis, Determenistic_solution, linewidth = 1, label = 'Deterministic
    solution')
plt.plot(xaxis, column_means, linewidth = 1, label = 'MCMC mean solution')
plt.plot(numpy_array[:,1], numpy_array[:,0], label = 'Generated data')
## plt.plot(Time, numpy_array[:,1], 'r+',label = 'Covid gevallen Delft')
plt.plot(xaxis, quantileSIR1, linewidth = 1, label = 'MCMC quantile 1 solution'
    )
plt.plot(xaxis, quantileSIR3, linewidth = 1, label = 'MCMC quantile 3 solution'
    )
plt.legend(loc='center right')



plt.xticks(fontsize = 15)
plt.yticks(fontsize = 15)
plt.grid(True)                              # show grid
maxS=max(S_arr)
maxI=max(I_arr)
maxR=max(R_arr)
plt.axis([0, t_end, 0, S_init+I_init])
## plt.axis([0, t_end, 0, max(numpy_array[:,1])])                # define the
    axes

plt.show()                                # show the plot
```