BSc Thesis Applied Mathematics

Effect of travel restrictions between provinces in the Netherlands on the spread of COVID-19

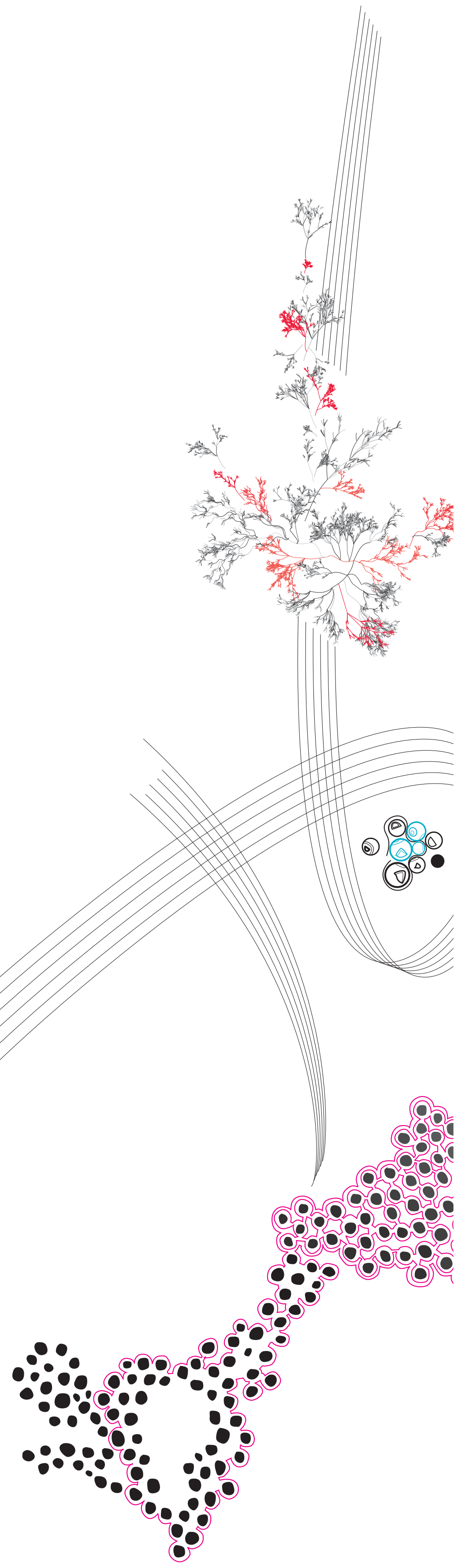
Wout Leemeijer

Supervisor: Matthias Schlottbom

June, 2021

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.



Preface

I want to thank Matthias Schlottbom for his help and advice during my Bachelor Thesis.

Effect of travel restrictions between provinces in the Netherlands on the spread of COVID-19

Wout Leemeijer

June, 2021

Abstract

To look at the effect of travel restrictions in the Netherlands, a model is developed that integrates the effect of commuting into a basic SIR-model, which models the spread of COVID-19 in each of the the Dutch provinces separately and in the entirety of the Netherlands. Based on available commuting data, the commuting parameters between each of the provinces are estimated, and based on the data on COVID-19 in the Netherlands the infection rate and the recovery rate are estimated by making use of a least-squares approximation. This model shows that not allowing any travel between the provinces will lead to less infections and hence less deaths in the entire Netherlands. However, looking at the provinces, the amount of infections shows that there are some differences. Some provinces will have less infections if commuting is not allowed, but there are also some provinces that will have more infections in case commuting is not allowed. This can be explained by the fraction of people commuting in and out of the province and by the fraction of people that are infected in each province. Future research in this model is possible, where vaccination is included into the model or the infection and recovery rate is estimated with a different approach.

Keywords: COVID-19, SIR-model, Travel restrictions, Lockdown, Commuting

1 Introduction

In the last one-and-a-half years, the world has been in a (partial) lockdown caused by the COVID-19 pandemic. This pandemic has left a big impact on the world by forcing governments to enact regulations such as a curfew and travel restrictions. Now that the world has been dealing with this crisis for the past one-and-a-half years it is interesting to see what the effect is of certain measures.

Some research has already been conducted on the effect of travel restrictions on the spread of the COVID-19 virus. For example, the global pandemic and mobility model (GLEAM) has been used to look at the effect of the travel restrictions to and from China. This model divides the world population into subpopulations that are centered around major transportation hubs (usually airports) [2]. However, this research mainly looks at the effects of worldwide travel restrictions. It is also interesting to look at travel restrictions on a smaller scale. Especially, since countries, for example Italy, have enforced regional lockdowns in an attempt to stop the spread of COVID-19 [7]. In Italy, the regional lockdowns still lead to a nationwide lockdown eventually. Would this also be the case in the Netherlands? Because of cultural differences, different population densities and commuting habits, this conclusion can not be immediately drawn for the Netherlands. Therefore, it is interesting to look at the effect of regional lockdowns in the Netherlands. Hence, in this paper the following research question will be answered:

What is the effect of travel restrictions between provinces on the COVID-19 pandemic in the Netherlands, compared to having no travel restrictions?

The goal of this paper is not to give a concrete answer whether travel restrictions between provinces should be implemented in case of a future pandemic, since this decision is subject to more factors, such as economical consequences of these travel restrictions. The goal is to give a clear insight on the effect on the COVID-19 pandemic and whether the travel restrictions will lead to fewer infections and deaths in the Netherlands and what the effect is on the amount of infections and deaths in each of the provinces.

To get insight on how the COVID-19 pandemic develops over time, a SIR-model is constructed. Since the goal is to model the effect of travel restrictions between provinces it is important to create a SIR-model for each of the 12 provinces in the Netherlands. These provinces will be connected by making use of commuting parameters. Furthermore, the infection and recovery parameters will be based on available data on COVID-19 in the Netherlands. Finally, based on simulations of the model with the estimated parameters, results will show what the effect is of having travel restrictions between provinces in Netherlands during the COVID-19 pandemic.

2 Modelling

2.1 The SIR-model

To model the population, the SIR-model divides the population into three different classes: the *susceptibles* S, which is the proportion of the population that can still be infected by the COVID-19 virus, the *infected* I, which is the proportion of the population that is currently infected with COVID-19 and the *removed* R, which is the proportion of people that have died or recovered from COVID-19. Since each person needs to be in one of these classes $S+I+R=1$, with $S, I, R \in [0,1]$. People get infected with an infection rate $\kappa > 0$ and people go to the removed class with a recovery rate $l > 0$. The rate of change for each of the classes can be expressed into the following set of differential equations [9]:

$$\begin{cases} \frac{dS}{dt} = -\kappa SI \\ \frac{dI}{dt} = \kappa SI - lI \\ \frac{dR}{dt} = lI \end{cases} \tag{1}$$

As can be seen from these equations, the rate of change for the susceptible class ($\frac{dS}{dt}$) only depends on the proportion of people that are susceptible (S) or infected (I), but not on the proportion of removed people (R). This is because the assumption is made that people can be infected only once. Hence, once people are in the removed class, they will stay in the removed class. Furthermore, the rate of change of the susceptibles depends on the infection rate κ. From this equation, it can be seen that if nobody is infected, the rate of change for the susceptibles is equal to zero. However, if a lot of people are infected the rate of change for the susceptibles will be higher. Moreover, the proportion of susceptibles is important. If a lot of people are still susceptible to the virus, the rate of change will be higher, but if not a lot of people are susceptible any more the rate of change will be lower. Hence, the rate of change for the susceptibles depends on the infection rate, on the proportion of susceptibles and on the proportion of infected people at that time.

The rate of change for the infected class depends on the amount of people in the susceptible class that get infected. People that will go out of the susceptible class will go into the infected class. However, this rate of change also depends on the amount of people that are removed, with recovery rate l. If a lot of people are infected, more people will recover or die and if not a lot people are infected, less people can recover or die, because they first need to be infected. Hence, the rate of change for the infected people depends on the amount of people that get infected in the susceptible class minus the amount of people that recover or die.

The rate of change for the removed class only depends on the amount of people that are infected and on the recovery rate l. The people that are in the infected class, but have recovered or have died, will move to the removed class, with a rate dependent on the recovery rate and the proportion of infected people at that time.

2.2 The provinces model

To model each of the provinces in the Netherlands, the effect of commuting needs to be taken into account. To do this, assumptions need to be made. First of all, no people will enter the Netherlands from other countries. Also, the effect of new births and deaths not related to COVID-19 are not taken into account. This means that the total population in the Netherlands remains constant. Secondly, people will not move to another province permanently, but only go there for a part of a day. This means that the amount of people living in each province will remain constant. Lastly, the model will make time steps of one day, since the available data makes time steps of one day.

To understand how the SIR-models for the provinces interact with each other, the interaction between two provinces will be considered, in this case Groningen and Friesland. Considering the susceptibles in Groningen during one day, it can be reasoned that this depends on the amount of susceptibles that live in Groningen, minus the amount of susceptibles that work in Friesland for a part of the day, plus the amount of susceptibles that live in Friesland but work in Groningen for a part of the day. To look at the effective fraction of susceptibles, the amount of susceptibles need to be normalized by dividing it with the effective population inside Groningen during the day. Taking this into account, the effective fraction of susceptibles in Groningen during a day becomes:

$$S_{G_E} = \frac{S_G + w(-c_{GF}S_G + c_{FG}S_F)}{N_G + w(-c_{GF}N_G + c_{FG}N_F)} \tag{2}$$

Here, S_G is the amount of susceptibles living in Groningen, $w \in [0,1]$ is the fraction of a day people are working, $c_{GF} \in [0,1]$ is the percentage of people living in Groningen that work in Friesland, $c_{FG} \in [0,1]$ is the percentage of people living in Friesland that work in Groningen, S_F is the amount of susceptibles living in Friesland, N_G is the amount of people living in Groningen and N_F is the amount of people living in Friesland. The same way of reasoning can be used for the effective fraction of infected people in Groningen during a day. However, since infected people are urged to stay home, the amount of travel between provinces for infected people should be lower. The amount of travel is not equal to 0, because some people do not notice they are infected and hence still travel. The fraction of infected people that still travel will be represented by q. Hence, the effective fraction of infected people in Groningen during a day becomes:

$$I_{G_E} = \frac{I_G + qw(-c_{GF}I_G + c_{FG}I_F)}{N_G + w(-c_{GF}N_G + c_{FG}N_F)} \tag{3}$$

Here, I_G is the amount of infected people living in Groningen and I_F is the amount of infected people living in Friesland. The same reasoning can be used to determine the effective fraction of removed persons in Groningen in a day. However, since in the removed class people that have died are also counted, the amount of commuting needs to lowered with a fraction $d \in [0, 1]$, which is the fraction of people still alive after having COVID-19. This yields the following expression for the effective fraction of removed people in Groningen during a day:

$$R_{G_E} = \frac{R_G + dw(-c_{GF}R_G + c_{FG}R_F)}{N_G + w(-c_{GF}N_G + c_{FG}N_F)} \tag{4}$$

Here, R_G is the amount of removed people living in Groningen and R_F is the amount of infected people living in Friesland.

This way of reasoning can also be used to determine the expression of the effective fraction of susceptibles during a day in Friesland, which yields:

$$S_{F_E} = \frac{S_F + w(-c_{FG}S_F + c_{GF}S_G)}{N_F + w(-c_{FG}N_F + c_{GF}N_G)} \tag{5}$$

And also, the effective fraction of infected people during a day in Friesland, which yields:

$$I_{F_E} = \frac{I_F + qw(-c_{FG}I_F + c_{GF}I_G)}{N_F + w(-c_{FG}N_F + c_{GF}N_G)} \tag{6}$$

The effective fraction of removed people in Friesland during a day is:

$$R_{F_E} = \frac{R_F + dw(-c_{FG}R_F + c_{GF}R_G)}{N_F + w(-c_{FG}N_F + c_{GF}N_G)} \tag{7}$$

Now, a SIR-model can be constructed for both Groningen and Friesland. For Groningen the infection rate is $\kappa_G > 0$ and the recovery rate is $l_G > 0$. For Friesland the infection rate is $\kappa_F > 0$ and the recovery rate is $l_F > 0$. Substituting equations (2) and (3) into equation (1) gives the SIR-model for Groningen, while substituting equations (5) and (6) into equation (1) gives the SIR-model for Friesland. This yields the following set of differential equations for the effective rates of change that are in each province in a day:

$$\begin{cases} \textit{Groningen:} \\ \frac{dS_{G_E}}{dt} = -\kappa_G S_{G_E} I_{G_E} \\ \frac{dI_{G_E}}{dt} = \kappa_G S_{G_E} I_{G_E} - l_G I_{G_E} \\ \frac{dR_{G_E}}{dt} = l_G I_{G_E} \\ \\ \textit{Friesland:} \\ \frac{dS_{F_E}}{dt} = -\kappa_F S_{F_E} I_{F_E} \\ \frac{dI_{F_E}}{dt} = \kappa_G S_{F_E} I_{F_E} - l_F I_{F_E} \\ \frac{dR_{F_E}}{dt} = l_F I_{F_E} \end{cases} \tag{8}$$

From these SIR-models, it can be seen that the effective rate of change for the susceptibles in Groningen now does not depend on the actual amount of susceptibles in Groningen, but on the effective amount of susceptibles in Groningen. The same holds for rate of change for infected and removed people in Groningen. The commuting parameters c_{FG} and c_{GF} determine how big the influence is of one province on the other.

But what if there are three provinces in the model, for example Groningen, Friesland and Drenthe. There are also people living in Groningen that work in Drenthe for a part of the day, which need to be subtracted from the amount of susceptibles in Groningen. Moreover, there are people working in Groningen for a part of the day, that live in Drenthe, which need to be added to the amount of susceptibles in Groningen. Hence, equation (2) becomes:

$$S_{G_E} = \frac{S_G + w(-(c_{GF} + c_{GD})S_G + c_{FG}S_F + c_{DG}S_D)}{N_G + w(-(c_{GF} + c_{GD})N_G + c_{FG}N_F + c_{DG}N_D)} \tag{9}$$

Here S_D is the amount of susceptibles in Drenthe, $c_{GD} \in [0, 1]$ is the commuting parameter from Groningen to Drenthe and $c_{DG} \in [0, 1]$ is the commuting parameter from Drenthe to Groningen. With the same reasoning equation (3) becomes:

$$I_{G_E} = \frac{I_G + qw(-(c_{GF} + c_{GD})I_G + c_{FG}I_F + c_{DG}I_D)}{N_G + w(-(c_{GF} + c_{GD})N_G + c_{FG}N_F + c_{DG}N_D)} \tag{10}$$

where I_D is the amount of infected persons in Drenthe. Finally, equation (4) becomes:

$$R_{G_E} = \frac{R_G + dw(-(c_{GF} + c_{GD})R_G + c_{FG}R_F + c_{DG}R_D)}{N_G + w(-(c_{GF} + c_{GD})N_G + c_{FG}N_F + c_{DG}N_D)} \tag{11}$$

where R_D is the amount of removed persons in Drenthe.
Now a general expression can be derived for the effective fraction of susceptible, infected and removed persons in each province, in case there are n provinces. First define the matrix $\boldsymbol{C}$ of size $n \times n$ as follows:

$$\boldsymbol{C} = \begin{bmatrix} 0 & c_{0,1} & c_{0,2} & \cdots & c_{0,n-1} \\ c_{1,0} & 0 & c_{1,2} & \cdots & c_{1,n-1} \\ c_{2,0} & c_{2,1} & 0 & \cdots & c_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & c_{n-1,2} & \cdots & 0 \end{bmatrix} \tag{12}$$

where entry $c_{i,j}$ is the commuting parameter from province i to province j and let $\boldsymbol{C'}$ be the following $n \times n$ matrix:

$$\boldsymbol{C'} = \begin{bmatrix} \sum_{k\neq 0}^{n-1} c_{0,k} & 0 & 0 & \cdots & 0 \\ 0 & \sum_{k\neq 1}^{n-1} c_{1,k} & 0 & \cdots & 0 \\ 0 & 0 & \sum_{k\neq 2}^{n-1} c_{2,k} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sum_{k\neq n-1}^{n-1} c_{n-1,k} \end{bmatrix} \tag{13}$$

From equations (2) and (9) the following equation for the effective fraction of susceptible persons living in province i can be derived:

$$S_{i_E} = \frac{S_i + w\Bigg(- \boldsymbol{C'}(i,i)S_i + \sum_{j=0}^{n-1} \boldsymbol{C}(j,i)S_j\Bigg)}{N_i + w\Bigg(- \boldsymbol{C'}(i,i)N_i + \sum_{j=0}^{n-1} \boldsymbol{C}(j,i)N_j\Bigg)} \tag{14}$$

Here the provinces get an index going from 0 to $n-1$, which means that S_i is the current amount of susceptibles living in province $i \in [0, n-1]$ and N_i is the amount of people living in province i. In equation (14), $\boldsymbol{C'}(i,i)S_i$ is the fraction of susceptible people that live in province i but work in another province for a part of the day and $\sum_{j=0}^{n-1} \boldsymbol{C}(j,i)S_j$ is the fraction of susceptible people that live in another province, but work in province i. Now, define a vector $\boldsymbol{s_E} = [S_{1_E}\ S_{2_E} \cdots\ S_{n_E}]^T$ which stores the effective fraction of susceptibles in each province, define the vector $\boldsymbol{s} = [S_1\ S_2 \cdots\ S_n]^T$ which stores the amount of susceptible people living in each province and define the vector $\boldsymbol{n} = [N_1\ N_2 \cdots\ N_n]^T$ which stores the amount of people living in each province. Then, writing equation (14) as follows, where $\mathcal{I}_n$ is the identity matrix of size $n \times n$, gives:

$$\boldsymbol{s_E} = \frac{\mathrm{s} + w(-\boldsymbol{C'}\boldsymbol{s} + \boldsymbol{C}^T\boldsymbol{s})}{\boldsymbol{n_E}} = \frac{(\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T)\boldsymbol{s}}{\boldsymbol{n_E}} \tag{15}$$

where $\boldsymbol{n_E}$ is a vector that stores the effective population, defined by:

$$\boldsymbol{n_E} = \boldsymbol{n} + w(-\boldsymbol{C'}\boldsymbol{n} + \boldsymbol{C}^T\boldsymbol{n}) = (\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T)\boldsymbol{n} \tag{16}$$

Equation (15) and (16) are both a linear transformation mapping a nonnegative vector to a new nonnegative vector.
With the same reasoning, a general expression for the effective fraction of infections for n provinces can be formed from equations (3) and (10). This yields:

$$I_{i_E} = \frac{I_i + qw\Big(-\boldsymbol{C'}(i,i)I_i + \sum_{j=0}^{n-1} \boldsymbol{C}(j,i)I_j\Big)}{N_i + w\Big(-\boldsymbol{C'}(i,i)N_i + \sum_{j=0}^{n-1} \boldsymbol{C}(j,i)N_j\Big)} \tag{17}$$

where I_i is the amount of people living in province i that are infected. Again, define a vector $\boldsymbol{i_E} = [I_{1_E}\ I_{2_E} \cdots\ I_{n_E}^T]$ which stores the effective fraction of infected people in each province and a vector $\boldsymbol{i} = [I_1\ I_2 \cdots\ I_n]^T$ which stores the actual amount of infections in each province. Now, equation (17) can be written in vector form:

$$\boldsymbol{i_E} = \frac{\boldsymbol{i} + qw(-\boldsymbol{C'}\boldsymbol{i} + \boldsymbol{C}^T\boldsymbol{i})}{\boldsymbol{n_E}} = \frac{(\mathcal{I}_n - qw\boldsymbol{C'} + qw\boldsymbol{C}^T)\boldsymbol{i}}{\boldsymbol{n_E}} \tag{18}$$

This is also a linear transformation mapping a nonnegative vector to a new nonnegative vector.
And again with the same reasoning, a general expression for the effective fraction of removed people can be formed from equations (4) and (11):

$$R_{i_E} = \frac{R_i + dw\Big(-\boldsymbol{C'}(i,i)R_i + \sum_{j=0}^{n-1} \boldsymbol{C}(j,i)R_j\Big)}{N_i + w\Big(-\boldsymbol{C'}(i,i)N_i + \sum_{j=0}^{n-1} \boldsymbol{C}(j,i)N_j\Big)} \tag{19}$$

where R_i is the amount of people living in province i that have been removed. Defining a vector $\boldsymbol{r_E} = [R_{1_E}\ R_{2_E} \cdots\ R_{n_E}^T]$ that stores the effective fraction of removed people in each province during a day and another vector $\boldsymbol{r} = [R_1\ R_2 \cdots\ R_n]^T$ that stores the actual amount of removed people living in each province, gives the following vector form:

$$\boldsymbol{r_E} = \frac{\boldsymbol{r} + dw(-\boldsymbol{C'}\boldsymbol{r} + \boldsymbol{C}^T\boldsymbol{r})}{\boldsymbol{n_E}} = \frac{(\mathcal{I}_n - dw\boldsymbol{C'} + dw\boldsymbol{C}^T)\boldsymbol{r}}{\boldsymbol{n_E}} \tag{20}$$

Again, this is a linear transformation mapping a nonnegative vector to a new nonnegative vector.

Now, a generalized SIR-model for the effective fraction of susceptible, infected and removed persons for each province can be made, as in equation (8).

$$\begin{cases} \frac{d\boldsymbol{s_E}}{dt} = -\kappa \boldsymbol{s_E} \boldsymbol{i_E} \\ \frac{d\boldsymbol{i_E}}{dt} = \kappa \boldsymbol{s_E} \boldsymbol{i_E} - l\boldsymbol{i_E} \\ \frac{d\boldsymbol{r_E}}{dt} = l\boldsymbol{i_E} \end{cases} \tag{21}$$

Here, κ is the vector $[\kappa_0\ \kappa_1 \cdots\ \kappa_{n-1}]$ where $\kappa_i > 0$ is the infection rate in province i and l is the vector $[l_0\ l_1 \cdots\ l_{n-1}]$, where $l_i > 0$ is the recovery rate in province i.

To get the actual amount of susceptible, infected and removed persons, the effective fractions should be transformed again. Equations (15), (18) and (20) give that:

$$\begin{cases} \boldsymbol{s} = \boldsymbol{n_E}(\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T)^{-1}\boldsymbol{s_E} \\ \boldsymbol{i} = \boldsymbol{n_E}(\mathcal{I}_n - qw\boldsymbol{C'} + qw\boldsymbol{C}^T)^{-1}\boldsymbol{i_E} \\ \boldsymbol{r} = \boldsymbol{n_E}(\mathcal{I}_n - dw\boldsymbol{C'} + dw\boldsymbol{C}^T)^{-1}\boldsymbol{r_E} \end{cases} \tag{22}$$

This set of equations only holds if the matrices $\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T$, $\mathcal{I}_n - qw\boldsymbol{C'} + qw\boldsymbol{C}^T$ and $\mathcal{I}_n - dw\boldsymbol{C'} + dw\boldsymbol{C}^T$ are invertible. Banach's Lemma gives us that [19]:

Lemma 2.1 (Banach's Lemma). *Let $\boldsymbol{B}$ be an $n \times n$ matrix. If in some induced matrix norm $\|\boldsymbol{B}\| < 1$, then $\mathcal{I}_n + \boldsymbol{B}$ is invertible and $\|(\mathcal{I}_n + \boldsymbol{B})\|^{-1} \leq \frac{1}{(1-\|\boldsymbol{B}\|)}$*

This lemma gives that the matrix $\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T$ is invertible if $\| - w\boldsymbol{C'} + w\boldsymbol{C}^T\|_1 < 1$. This yields the following:

$$\boldsymbol{B} = -w\boldsymbol{C'} + w\boldsymbol{C}^T = \begin{bmatrix} -w\sum_{k\neq 0}^{n-1} c_{0,k} & wc_{1,0} & wc_{2,0} & \cdots & wc_{n-1,0} \\ wc_{0,1} & -w\sum_{k\neq 1}^{n-1} c_{1,k} & wc_{2,1} & \cdots & wc_{n-1,1} \\ wc_{0,2} & wc_{1,2} & -w\sum_{k\neq 2}^{n-1} c_{2,k} & \cdots & wc_{n-1,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ wc_{0,n-1} & wc_{1,n-1} & wc_{2,n-1} & \cdots & -w\sum_{k\neq n-1}^{n-1} c_{n-1,k} \end{bmatrix}$$

Since $\|\boldsymbol{B}\|_1 := \max_{0\leq j\leq n-1} \sum_{i=0}^{n-1} |B_{i,j}|$, which is equal to the maximum column sum of $\boldsymbol{B}$ we have that $\boldsymbol{B}$,

$$\begin{aligned} \|\boldsymbol{B}\|_1 &= \max_{0\leq j\leq n-1} \left(| - w\sum_{k\neq j}^{n-1} c_{j,k}| + w|c_{j,0}| + w|c_{j,i}| + \cdots + w|c_{j,j-1}| + w|c_{j,j+1}| + \cdots + w|c_{j,n-1}| \right) \\ &= \max_{0\leq j\leq n-1} (w\sum_{k\neq j}^{n-1} |c_{j,k}| + w\sum_{k\neq j}^{n-1} |c_{j,k}|\Big) \\ &= \max_{0\leq j\leq n-1} \left(2w\sum_{k\neq j}^{n-1} c_{j,k} \right) < 1 \end{aligned}$$

Applying the same reasoning to $\mathcal{I}_n - qw\boldsymbol{C'} + qw\boldsymbol{C}^T$ and $\mathcal{I}_n - dw\boldsymbol{C'} + dw\boldsymbol{C}^T$ gives the following lemma:

Lemma 2.2. *For the $n \times n$ identity matrix $\mathcal{I}_n$ and $n \times n$-matrices $\boldsymbol{C}$ and $\boldsymbol{C'}$ as defined in equations (12) and (13), we have that $\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T$ is invertible if:*

$$\max_{0 \le j \le n-1} \left(\sum_{k \neq j}^{n-1} c_{j,k} \right) < \frac{1}{2w} \tag{23}$$

Furthermore, $\mathcal{I}_n - qw\boldsymbol{C'} + qw\boldsymbol{C}^T$ is invertible if:

$$\max_{0 \le j \le n-1} \left(\sum_{k \neq j}^{n-1} c_{j,k} \right) < \frac{1}{2qw} \tag{24}$$

And $\mathcal{I}_n - dw\boldsymbol{C'} + dw\boldsymbol{C}^T$ is invertible if:

$$\max_{0 \le j \le n-1} \left(\sum_{k \neq j}^{n-1} c_{j,k} \right) < \frac{1}{2dw} \tag{25}$$

Since $q, d \in [0,1]$ and since $c_{k,j} \in [0,1]$ is the fraction of people living in province k and working in province j the sum $\sum_{k \neq j}^{n-1} c_{k,j} < 1$ for all k and j, it is clear that for $w < \frac{1}{2}$ equations (23), (24) and (25) always hold. Therefore the set of equations in equation (22) hold if $w < \frac{1}{2}$. Numerically checking the invertibility of $\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T$, $\mathcal{I}_n - qw\boldsymbol{C'} + qw\boldsymbol{C}^T$ and $\mathcal{I}_n - dw\boldsymbol{C'} + dw\boldsymbol{C}^T$ even shows that these matrices are even invertible in the most extreme case, where $w = 1$, $q = 1$, $d = 1$ and $\sum_{k \neq j}^{n-1} c_{k,j} = 1$ for all $j \in [0, n-1]$.

The goal of the model is to get the actual fraction and amount of people for each province for each of the classes *susceptible*, *infected* and *removed*. This means that the rate of change for the susceptible, infected and removed people living in a province needs to be given as output of the model. From equation (22) we therefore get the following expression:

$$\begin{cases} \frac{d\boldsymbol{s}}{dt} = \boldsymbol{n_E}(\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T)^{-1} \frac{\boldsymbol{s_E}}{dt} \\ \frac{\boldsymbol{i}}{dt} = \boldsymbol{n_E}(\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T)^{-1} \frac{\boldsymbol{i_E}}{dt} \\ \frac{d\boldsymbol{r}}{dt} = \boldsymbol{n_E}(\mathcal{I}_n - w\boldsymbol{C'} + w\boldsymbol{C}^T)^{-1} \frac{\boldsymbol{r_E}}{dt} \end{cases} \tag{26}$$

3 Data Analysis

3.1 Commuting Data

For the commuting parameters a data set published by het Centraal Bureau voor Statistiek (CBS) is used [16]. This data set gives information about how many jobs there were in a certain region for people living in a certain region for the years 2014, 2015, 2016, 2017, 2018 and 2019. This also contains data for the amount of jobs there are for people that live in province i and work in province j for all the 12 provinces. All commuting parameters $c_{i,j}$ for $i \in [0, 11]$ and $j \in [0, 11]$ are calculated by taking the average amount of jobs over these 6 years and dividing them by the population of province i. These commuting parameters are then used to define matrices $\boldsymbol{C}$ and $\boldsymbol{C'}$ in equations (12) and (13). Here the provinces are indexed as follows:

- *i=0*: Groningen
- *i=1*: Friesland

- *i=2*: Drenthe
- *i=3*: Overijssel
- *i=4*: Flevoland
- *i=5*: Gelderland
- *i=6*: Utrecht
- *i=7*: Noord-Holland
- *i=8*: Zuid-Holland
- *i=9*: Zeeland
- *i=10*: Noord-Brabant
- *i=11*: Limburg

In the Netherlands men work on average 40.5 hours per week and women work on average 26 hours per week [3]. Assuming that in the Netherlands there are approximately the same amount of men and women, the average person works 33.5 hours per week. Assuming a person works the same amount of hours every day and people work on average $\frac{33.5}{5} = 6.7$ hours in a day, based on a five-day workweek. This means that people work $\frac{6.7}{24} \approx 28\%$ of the day. For simplicity, this is also assumed to be true in the weekends. Then $w = 0.28$, which satisfies the invertibility condition needed for equations (23), (24) and (25).

Furthermore, a study conducted by, the Dutch National Institute for Public Health, the RIVM shows that in the Netherlands 76% of the population stayed at home after being infected with COVID-19 [13]. This means that the fraction of infected people that will still travel despite being infected in the model is set to $q = 1 - 0.76 = 0.24$.

The John Hopkins University and Medicine researched the mortality rate of COVID-19 in all countries. For the Netherlands they determined the death rate to be 1.1% [10]. This means that in the model the amount of removed people that will still travel $d = 1-0.011 = 0.989$.

3.2 COVID-19 Data

For the analysis of the COVID-19 data, a data set is used that collects the data about the amount of new infections, hospital admissions, deaths and vaccinations for each day for each of the provinces and municipalities in the Netherlands [15]. This data is recorded by, among others, the RIVM and the CBS.

The SIR-model in equation (21) depends on the two parameters κ and l, where κ is the infection rate and l is the recovery rate. These will be determined based on the amount of susceptible persons and infections in the data set.

Since the spread of the COVID-19 virus is not constant over time, this means that the infection rate also needs to differ over time. To make sure that this happens in the model, five time periods have been constructed, based on the regulations enforced by the Dutch government at different times [12]. These time periods are defined to be the following:

- **Time period 1:** (*3 March - 31 May:*) The first lockdown in the Netherlands
- **Time period 2:** (*1 June - 29 September:*) The end of the first lockdown
- **Time period 3:** (*30 September - 14 December:*) Partial lockdown
- **Time period 4:** (*15 December - 27 April:*) Full Lockdown
- **Time period 5:** (*28 April - Current Date:*) End of Lockdown

For each of these time periods, the goal is to get an estimate for κ and for l. The available data is not equal to the effective fraction of susceptible and infected people in a province, but is equal to the actual amount of susceptible and infected people in the province. This means that first the amount of susceptible and infected people from the data needs to be expressed as the effective fraction of susceptible and infected people. This is done by making use of equations (15) and (18).

The data collected by the RIVM is not perfect. For example, the amount of new infections recorded each day is not always correct, since they can also include infections of the previous day which had not been recorded yet. Hence, the data contains some noise. To filter out this noise, cubic spline interpolation is used. A cubic spline is a function which is defined piecewise by cubic polynomials. This means that instead of trying to fit one higher order polynomial which estimates all of the data points, the data is estimated by several piecewise cubic polynomials, inbetween each of the data points. This also prevents Runge's phenomenon, which occurs with polynomial interpolation of a high degree, where oscillation in the polynomial occurs at the edges of the interval where the polynomial is defined [18]. This means that between every time point t_i, which is day i in the data set, and t_{i+1} the goal is to find a polynomial p which minimizes the error between the value of p at t_i and the effective fraction of susceptibles at t_i. This leads to the following:

$$\min_p(|p(t_i) - s_E(t_i)|_2^2) \tag{27}$$

And the same holds for the cubic spline to estimate the effective fraction of infected persons at t_i, but now for a polynomial q:

$$\min_q(|q(t_i) - i_E(t_i)|_2^2) \tag{28}$$

Since there is only data available on the amount of infections and susceptible people, but not on the amount of removed people in a day, the infection rate κ and l can be estimated from the following equations from the SIR-model in equation (21):

$$\begin{cases} \frac{d\boldsymbol{s_E}}{dt} = -\kappa \boldsymbol{s_E} \boldsymbol{i_E} \\ \frac{d\boldsymbol{i_E}}{dt} = \kappa \boldsymbol{s_E} \boldsymbol{i_E} - l\boldsymbol{i_E} \end{cases} \tag{29}$$

Making use of the cubic spline for the susceptibles in equation (27) the expression for $\frac{d\boldsymbol{s_E}}{dt}$ in equation (29) at time t_i becomes:

$$p'(t_i) = -\kappa p(t_i) q(t_i) \tag{30}$$

Next, making use of the cubic spline for the infected in equation (28) the expression for $\frac{d\boldsymbol{i_E}}{dt}$ in equation (29) becomes

$$q'(t_i) = \kappa p(t_i) q(t_i) - l q(t_i) \tag{31}$$

To find the optimal value for κ and l at time t_i least-square approximation can be used. Define a matrix $\boldsymbol{A}$ as follows:

$$\boldsymbol{A} = \begin{bmatrix} -p(t_i)q(t_i) & 0 \\ p(t_i)q(t_i) & -q(t_i) \end{bmatrix} \tag{32}$$

Furthermore, define vectors $\boldsymbol{x} = [\kappa\ l]^T$ and $\boldsymbol{b} = [p'(t_i)\ q'(t_i)]^T$. This yields that $\boldsymbol{Ax} = \boldsymbol{b}$ gives the same equations at time t_i as in equation (29), but now using cubic splines. To find κ and l, now the following least square approximation can be used:

$$\min_{\boldsymbol{x}} |\boldsymbol{Ax} - \boldsymbol{b}|_2^2 \tag{33}$$

However, the goal is not to estimate κ at each time point, but to estimate one κ for each time period. This means that for a time period $t = [t_0\ t_1\ t_2 \cdots t_n]$ the matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$ should be extended as follows:

$$\boldsymbol{A} = \begin{bmatrix} -p(t_0)q(t_0) & 0 \\ p(t_0)q(t_0) & -q(t_0) \\ -p(t_1)q(t_1) & 0 \\ p(t_1)q(t_1) & -q(t_1) \\ -p(t_2)q(t_2) & 0 \\ p(t_2)q(t_2) & -q(t_2) \\ \vdots & \vdots \\ -p(t_n)q(t_n) & 0 \\ p(t_n)q(t_n) & -q(t_n) \end{bmatrix} \tag{34}$$

and

$$\boldsymbol{b} = \begin{bmatrix} p'(t_0) \\ q'(t_0) \\ p'(t_1) \\ q'(t_1) \\ p'(t_2) \\ q'(t_2) \\ \vdots \\ p'(t_n) \\ q'(t_n) \end{bmatrix} \tag{35}$$

Now applying the least squares approximation from equation (33) to this matrix $\boldsymbol{A}$ and vectors x and b, gives the optimal value for x and hence κ and l in the time period $t = [t_0$ $t_1\ t_2 \cdots t_n]$.

Since the infection rates in each of the provinces differ from each other, it is interesting to determine κ and l for each of the provinces separately. Applying the method explained above to each time period and each province gives the values for κ as shown in table 1 and the values for l as shown in table 2.

4 Results

For the results the model will be simulated based on the parameters discussed in the previous section. For all tests a comparison will be made between the case that commuting

Time period	1	2	3	4	5
Groningen	0.93	0.96	0.99	1.02	1.06
Friesland	0.93	0.92	0.99	1.02	1.04
Drenthe	0.95	0.93	0.99	1.02	1.06
Overijssel	0.96	0.97	0.99	1.04	1.09
Flevoland	0.92	0.94	0.97	1.03	1.06
Gelderland	0.97	0.97	1.01	1.03	1.09
Utrecht	0.93	0.97	1.01	1.07	1.11
Noord-Holland	0.97	0.98	1.02	1.07	1.12
Zuid-Holland	0.96	0.97	1.02	1.06	1.13
Zeeland	0.94	0.91	0.99	1.02	1.07
Noord-Brabant	0.97	0.99	1.01	1.06	1.12
Limburg	0.94	0.97	1.00	1.05	1.10

Table 1: Values for κ in each of provinces in each of the time periods.

Time period	1	2	3	4	5
Groningen	0.93	0.96	0.97	0.98	1.01
Friesland	0.93	0.96	0.97	0.97	0.99
Drenthe	0.93	0.96	0.98	0.98	0.99
Overijssel	0.93	0.96	0.96	0.97	01.02
Flevoland	0.93	0.96	0.95	0.95	0.97
Gelderland	0.93	0.96	0.95	0.96	0.94
Utrecht	0.93	0.96	0.91	0.95	0.97
Noord-Holland	0.93	0.95	0.95	0.94	1.03
Zuid-Holland	0.93	0.96	0.96	0.95	0.91
Zeeland	0.93	0.96	0.97	1.69	0.99
Noord-Brabant	0.93	0.95	0.95	0.95	0.99
Limburg	0.93	0.96	0.96	0.95	0.96

Table 2: Values for l in each of provinces in each of the time periods.

between provinces is still possible and the case that commuting is not possible anymore which means that all entries in the matrices $\boldsymbol{C}$ and $\boldsymbol{C'}$ in equations (12) and (13) are set to 0. Since the data set on COVID-19 contains data starting from 3 March 2020, the simulations will start at this point, hence $t = 0$ is 3 March 2020. And the simulations run until 24 June 2021, meaning that it ends at $t = 478$. The Python models used for the simulations can be found in appendix A.

In table 3 the amount of infections in each province on 3 March 2020 are shown. Running the model with these starting values gives the trajectory for the fraction of infected persons in each province as shown in figure 1 in case commuting is not allowed. In this figure the dashed lines represent the different time periods.

Since commuting is not allowed and some provinces do not have any infected persons at the start, it can be seen that most provinces won't have any infections at all. However, this case is not realistic. It is not likely that these travel restrictions would have been implemented immediately. Moreover, in some provinces there could be some undetected

Provinces	Amount of infections on 3 March 2020
Groningen	0
Friesland	0
Drenthe	2
Overijssel	0
Flevoland	0
Gelderland	0
Utrecht	3
Noord-Holland	3
Zuid-Holland	3
Zeeland	2
Noord-Brabant	12
Limburg	0

TABLE 3: Amount of infections on 3 March 2020.

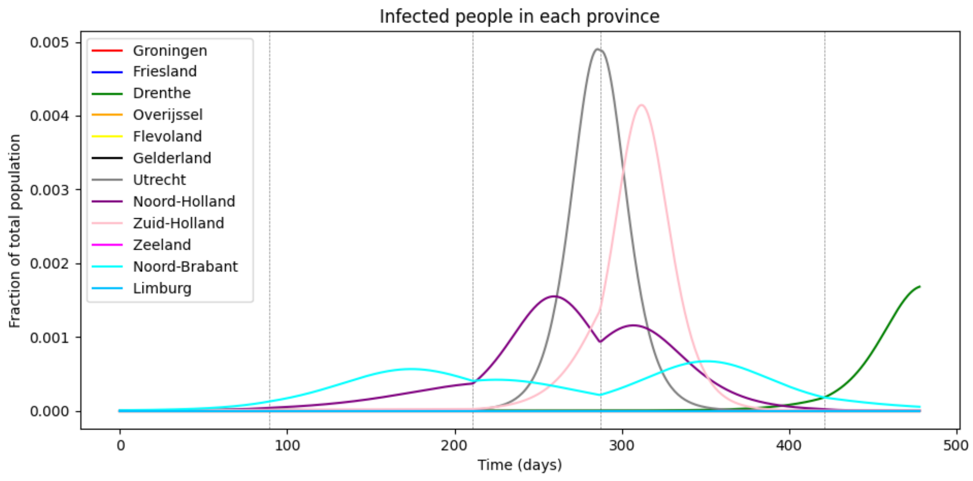


FIGURE 1: Fraction of infected persons in each province over time if commuting is not allowed and starting conditions are based on actual data.

cases of COVID-19, which can still cause people to get infected. Hence, it is not likely that some provinces will not have any infections at all during the COVID-19 pandemic with these travel restrictions. Therefore, to test the model, each province is set to have one infected person from the start. Simulating gives the trajectory of the fraction of infected persons as shown in figure 2.

For this simulation, there are a couple interesting statistics to compare. First of all, the amount of infections in the entire Netherlands and in each of the provinces will be considered. In figure 3 the modelled fraction of infections in the entire Netherlands can be seen for both the case where commuting is allowed and where commuting is not allowed. As can be seen from this figure, the amount of infections when commuting is allowed differs at some points from the case where commuting is not allowed. Therefore, it is interesting to look at the exact number of infections.

In tables 4 and 5 the maximum amount of infections in a day, total infections during the

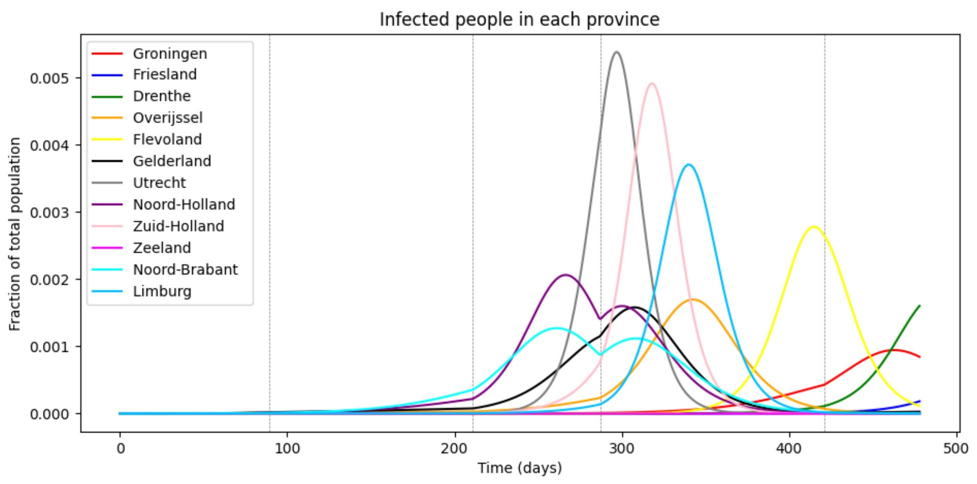


FIGURE 2: Fraction of infected persons in each province over time if commuting is not allowed and each province has one infected person at the start.

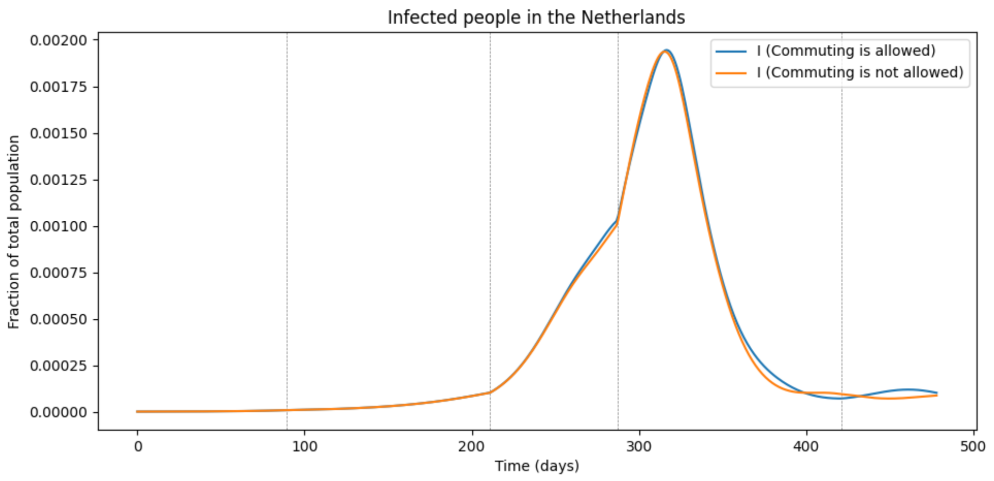


FIGURE 3: Fraction of infections in the Netherlands over time.

COVID-19 pandemic until today and the total amount of deaths calculated from the death rate of 1.1% are shown in case commuting is allowed and in case commuting is not allowed, respectively.
As can be seen, in the case where commuting is not allowed, the total number of infections, and hence the amount of deaths, will decrease in the Netherlands. However, for some of the provinces the total amount increases when commuting is not allowed. Therefore, it is interesting to compare the relative change in the amount of total infections, between when commuting is allowed and when it is not allowed. This is shown in table 6. In figure 4 a geographical representation of table 6 is shown.

As can be seen from table 6 and figure 4, some provinces will have approximately the same amount of infections, but other provinces will have significantly less or more infections. This can be caused by the amount of commuting that was done before and fraction of the population that is infected in that province. The fraction of the population that is infected

	Commuting Allowed		
	maximum amount of infections	total amount of infections	total amount of deaths
Netherlands	33824	2781523	30597
Groningen	529	37020	407
Friesland	710	43944	483
Drenthe	754	37635	413
Overijssel	1947	141064	1551
Flevoland	1090	60723	667
Gelderland	3286	273506	3009
Utrecht	7156	288532	3173
Noord-Holland	6009	542977	5972
Zuid-Holland	18734	762676	8389
Zeeland	2	46	1
Noord-Brabant	3275	401214	4413
Limburg	4052	192187	2214

Table 4: Infection numbers in case commuting between provinces is allowed.

	Commuting Not Allowed		
	maximum amount of infections	total amount of infections	total amount of deaths
Netherlands	33677	2708962	29799
Groningen	551	36916	406
Friesland	118	2719	30
Drenthe	783	20811	229
Overijssel	1974	142126	1563
Flevoland	1177	63751	701
Gelderland	3298	273017	3003
Utrecht	7291	289764	3187
Noord-Holland	5942	532785	5861
Zuid-Holland	18225	753402	8287
Zeeland	2	191	2
Noord-Brabant	3262	398816	4387
Limburg	4141	194664	2141

Table 5: Infection numbers in case commuting between provinces is not allowed.

in case commuting is allowed and in case commuting is not allowed is shown in table 7.

Looking at the matrix $\mathbf{C}$ as defined in equation (12), it can be seen that the sum of row i is the percentage of the population i commuting out of the province. The amount of people commuting into province i can be found by realising that entry $c_{j,i}$ is the amount of people coming in from province j relative to the population of province j. Therefore, this number should be normalised to the amount of people living in province i. This leads to the following expression for percentage of people commuting into province i, defined to be $\hat{c}_i$:

$$\hat{c}_i = \sum_{j \neq i} c_{j,i} \frac{n(j)}{n(i)} \tag{36}$$

Here, $n(j)$ is the amount of people living in province j and $n(i)$ is the amount of people living in province i. This gives the percentages of people commuting in and out of the different provinces, as shown in table 8.

	Relative change
Netherlands	-2.61 %
Groningen	-0.28 %
Friesland	-93.14 %
Drenthe	-44.75 %
Overijssel	-0.75 %
Flevoland	4.99 %
Gelderland	-0.18 %
Utrecht	0.43 %
Noord-Holland	-1.88 %
Zuid-Holland	-1.22 %
Zeeland	315 %
Noord-Brabant	-0.60 %
Limburg	1.29 %

TABLE 6: Relative change in total amount of infections in case commuting is not allowed anymore.

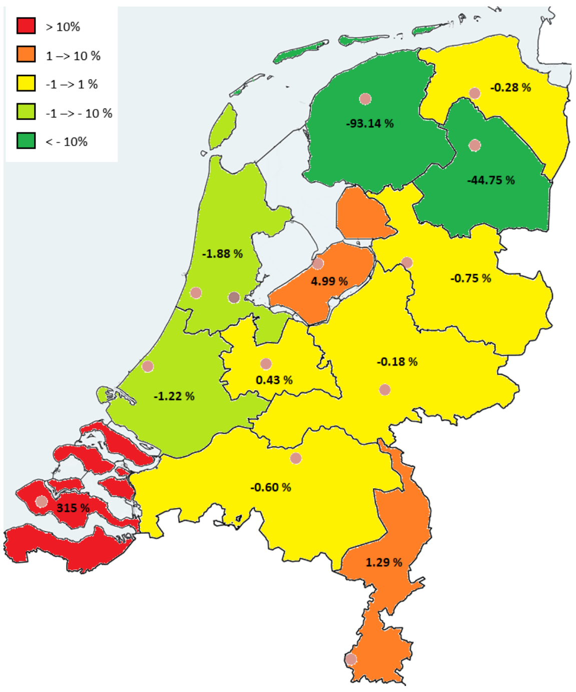


FIGURE 4: Map of the Netherlands showing the relative change in infections in case commuting is not allowed.

As can be seen from tables 6, 7 and 8, Friesland and Drenthe have a significant decrease in infections if commuting is not allowed. This is caused by the fact that in these provinces a low amount of people have been infected and more people are going out to other provinces with a higher fraction of infections than are coming in. This means that if commuting is allowed people in these provinces are more likely to come in contact with people that are infected, since they meet people from other provinces. Hence, if commuting is not allowed they will only meet people in their own province, where the fraction of infections is lower. Meaning that they are less likely to be infected. In Noord-Holland the amount of infections will slightly decrease, this is mainly because a lot more people are coming in

	Commuting allowed	Commuting not allowed
Nederland	16.0 %	15.6 %
Groningen	6.4 %	6.3 %
Friesland	6.8 %	0.42 %
Drenthe	7.7 %	4.3 %
Overijssel	12.1 %	12.2 %
Flevoland	14.4 %	15.1 %
Gelderland	13.1 %	13.1 %
Utrecht	21.3 %	21.4 %
Noord-Holland	18.9 %	18.5 %
Zuid-Holland	20.6 %	20.3%
Zeeland	0.012 %	0.050 %
Noord-Brabant	15.7 %	15.6%
Limburg	17.2 %	17.4 %

Table 7: Percentage of infected persons in each province.

	Going out	Coming in
Groningen	9.8 %	9.3 %
Friesland	8.1 %	4.2 %
Drenthe	15.1 %	11.1 %
Overijssel	9.0 %	8.9 %
Flevoland	23.3 %	11.8 %
Gelderland	11.6 %	9.1 %
Utrecht	14.9 %	19.6 %
Noord-Holland	5.4 %	10.7 %
Zuid-Holland	7.4 %	6.5 %
Zeeland	11.0 %	6.4 %
Noord-Brabant	7.7 %	8.3 %
Limburg	8.6 %	6.4 %

Table 8: Fraction of people commuting into each province or commuting out of each province.

than going out if commuting is allowed. Hence, if commuting is not allowed, there will be less people in a day in Noord-Holland, hence less people can infect other people during a day. In Zuid-Holland there are also slightly less infections. This is likely to be caused by the fact that if commuting is allowed a lot infections are from people in Utrecht, since this is the only province with a higher fraction of infections. Other explanations are less likely, since there are more people leaving the province than coming in, hence if commuting is not allowed there are more people in the province. Moreover, Zuid-Holland has, except for Utrecht, a higher fraction of infections than the rest of the provinces. In Flevoland and Limburg, the amount of infections will slightly increase. This is caused by the fact that when commuting is allowed, more people are leaving the province than coming in. Hence, when commuting is allowed more people are in the province during the day and therefore there are also more infected people in the province during a day. Leading to more infections in these provinces. In Zeeland there is a large increase of infections, but here the result doesn't seem likely. Compared to all other provinces Zeeland has a much lower fraction of infected people. This seems unlikely and therefore likely to be caused by incorrect data and hence and incorrect estimation of κ and l. Because Zeeland only has a small number of infections a small absolute increase quickly leads to a large relative increase, explaining the 315% increase in infections.

Lastly, looking at the amount of susceptible people in the Netherlands, as shown in figure 5, it can be seen that on 24 June 2021 approximately 85% of the population in the Netherlands is still susceptible. This means that a new outbreak of COVID-19 is still possible at a later moment.

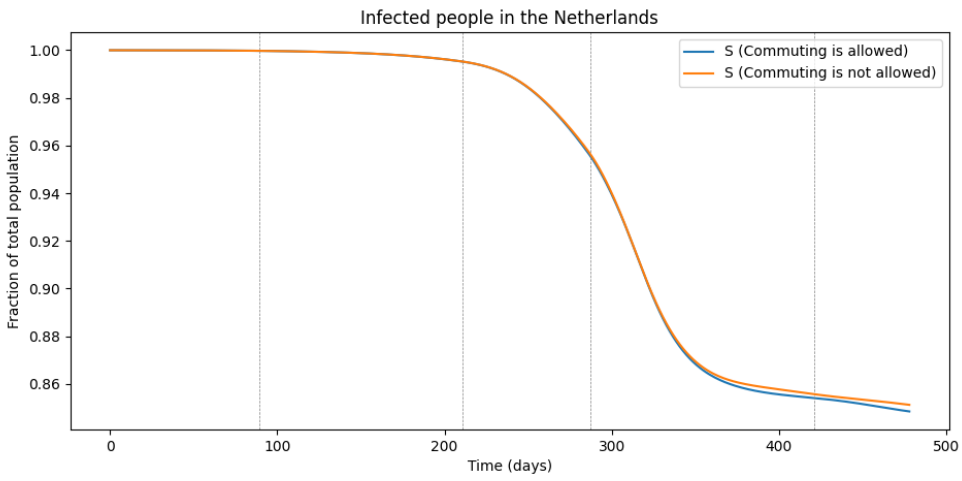


FIGURE 5: Fraction of susceptible people in the Netherlands.

5 Discussion

In the development of this model a few assumptions were made which simplify the real world situation. For future research, these assumptions might be changed. Also, there are other ways that this model can be extended in the future.

First of all, the assumption was made that people cannot be infected twice. While a study by Stokel-Walker shows that reinfection with the COVID-19 virus is unlikely, he states that there are still cases reported where people got reinfected [14]. Hence, to make the model more complete this could be added to the model, which would give a better estimation of the amount of people that are still susceptible at a certain time.

Furthermore, the SIR-model now only consists of the classes *susceptible*, *infected* and *removed*. This can also be extended by adding more classes. For example, adding a class for the deceased people, which separates the class *removed* into *deceased* and *recovered* people. In a study conducted by Giordano et al. on the COVID-19 pandemic in Italy a SIDHARTHE-model is developed, where they have the classes *susceptible*, *infected*, *diagnosed*, *healed*, ailing, *recognized*, *threatened* and *extinct* [6]. In this model, a more clear distinction is made in the population, which is helpful since in a normal SIR-model it is not clear which part of the infected people are hospitalized and in risk of dying. Making the SIR-model more extensive by adding more classes gives more insight in the state of the pandemic, helping governments to make the right decisions. Future research into the model presented in this paper can extend the model by adding more classes.

Moreover, in this model vaccination is not taken into account. Vaccination makes sure that less people are susceptible to attract the virus and therefore there are less infections during the pandemic. There are several studies available that already include this into a SIR-model, either by adding a class *vaccinated* or by changing the parameters κ and l, showing that vaccination can greatly reduce the risk of major outbreaks of the virus [4, 5]. This is also a interesting factor to add into the model developed in this paper for future research.

Next to this, in the model now only the case is tested when there are travel restrictions for everyone, while in real life it is likely that some traffic between provinces is necessary. This can be implemented in the model by adding a constant $h \in [0, 1]$ to the matrices c in (12) and c' in (13). In this case $h = 0$ would mean no commuting between provinces is allowed, and $h = 1$ would mean all commuting is allowed and any number between 0 and 1 would mean h is partially allowed. This could also show the effect of a partial lockdown between provinces.

In the model, the commuting parameters are solely based on travel to work. However, in real life, more traffic is going on between the provinces, but since no data is available on all different kinds of traffic between the provinces and how long people stay in the other province, this is not added into this model. If in the future more data is available, this could be added to give a more complete overview of the effect of travel restrictions between provinces. Moreover, travel time is also not taken into account. For this different types of commuting can be considered. A person is much more likely to get infected in a train than in their car. Hence, a distinction in the model can be made between people who commute by car and those who commute by train and the duration of their commuting trip.

Next to this, the data set of COVID-19 in the Netherlands is not entirely complete. In the beginning of the pandemic, there was not enough test capacity, meaning that a lot of the infected people were not detected. Also, since not everyone gets tested when feeling ill and not everyone gets symptoms of COVID-19, there are undetected cases of COVID-19. This means that these cases are at the moment not included in the model, since the model is purely based on the data available. However, Ivorra et al. have introduced a method that take the undetected cases into account in China [8]. This approach can also be used in this model to get an even better representation of the COVID-19 situation in the Netherlands.

Furthermore, the parameters κ and l are now estimated based on the defined time periods. However, these time periods might not be perfect, and using different time periods might yield better results. This means that more research can be conducted into the estimation of κ and l. Next to this, there are also studies that estimate κ as a continuous function of time. For example, a study conducted by Atkeson considers a function $\kappa(t)$ which also includes the effect of the different seasons and the effect of people less and less willing to comply with the measures taken against the spread of the pandemic [1]. Future research into the model can also include looking at continuous functions of κ, to see if this will yield more accurate results. In many models, the recovery rate l is also kept as a constant, based on the nature of the virus, instead of determining l from the data [17].
Another approach to optimize the estimation for κ and l is to make use of different numerical optimization approaches and regularization, which can prevent overfitting of the

data, to the approximation. This means that if a function $F : (\kappa, l) \longrightarrow (S, I, R)$, the goal is to minimize the following least squares expression with regularization:

$$\min_{\kappa,l} |F(\kappa, l) - S_d, I_d, R_D)|_2^2 + \alpha \frac{w_\kappa \kappa^2 + w_l l^2}{2} \tag{37}$$

Here S_d, I_d, R_d are the values for the amount of susceptibles, infected and removed people from the data, respectively. The parameter α can be determined based on the noise of the data, for example if for

$$|S_d - S| \leq \delta \tag{38}$$

and if the same holds for I_d and R_d, α can be set to δ according to the a-priori choice rule. The parameters w_κ and w_l are weights that can be assigned to κ and l in the regularization. To solve this least squares problem different numerical methods can be used, for example the Gauss-Newton method or the Levenberg-Marquardt method [11].

Since vaccination is not included in the model and the estimation of κ and l might not be perfect, the model overestimates the total amount of infections. The model gives 2781523 infections in the Netherlands if commuting is allowed, but according to the real data there are only 1681580 infections reported [15]. Hence, the results in the model are not completely accurate. However, by looking at the relative change in infections the effect of travel restrictions can still be seen. Therefore, because the results are not completely accurate and because the values for κ and l might differ in the future, no prediction is made for the future course of the pandemic, since the results won't be a good representation of reality.

Lastly, this model is now designed for the Dutch provinces. However, it can also be used on a different scale, for example for Dutch municipalities. Since there is also data available on the amount of infections in each municipality in the Netherlands, the parameters κ and l can still be estimated. To make sure this model works, the Python code in appendix A only needs some slight adjustments to store the necessary results, since it is now designed to work for the provinces. Hence, the model could also be used to look at the effect of travel restrictions on a smaller or on a larger scale.

6 Conclusion

The model developed in this paper shows that commuting between provinces does have an effect on the spread of COVID-19 in the Netherlands. If commuting between provinces is not allowed anymore, there will be 2.61% less infections and deaths in the Netherlands up until the 24 June 2021. This accumulates to 72561 less infections and 798 less deaths caused by COVID-19 according to the model. In the provinces the effect of travel restrictions is different, some provinces have less infections when commuting is not allowed, but others have more infections. This is caused by the fact that in some provinces more people are coming into the province when commuting is allowed. This means that the effective population in a day in these provinces is higher than the actual population of these provinces. Not allowing commuting then makes sure that this is not possible, meaning that there are less people in the province and hence reducing the amount of infections.

Looking at the provinces separately shows that provinces like Utrecht and Zuid-Holland have been most affected by the COVID-19 pandemic, since 21.3% and 20.6% of their population have been infected, respectively. This differs a lot from the most northern provinces

Groningen, Friesland and Drenthe, where only 6.4%, 6.8% and 7.7% of the people are infected, respectively. This shows that there is a big difference between provinces and that this model gives a better insight in how COVID-19 has affected the entirety of the Netherlands.

All in all, this model shows that commuting leads to more infections and deaths, but that there are significant differences between the effect of commuting for each province. Since the goal of this paper is only to look at the effect of travel restrictions on COVID-19 infections, there will not be a conclusion drawn whether this measure should be taken in case of a future pandemic. For this decision a lot more factors are relevant, for example the economical consequences of a lockdown. However, this model can be used by the government to make the consideration whether travel restrictions are necessary in case of a future outbreak of COVID-19 or a new pandemic, but let's hope that that is not happening.

References

[1] Andrew Atkeson. A parsimonious behavioral seir model of the 2020 covid epidemic in the united states and the united kingdom. Technical report, National Bureau of Economic Research, 2021.

[2] Matteo Chinazzi, Jessica T Davis, Marco Ajelli, Corrado Gioannini, Maria Litvinova, Stefano Merler, Ana Pastore y Piontti, Kunpeng Mu, Luca Rossi, Kaiyuan Sun, et al. The effect of travel restrictions on the spread of the 2019 novel coronavirus (covid-19) outbreak. *Science*, 368(6489):395–400, 2020.

[3] Christine R Cousins and Ning Tang. Working time and work and family conflict in the netherlands, sweden and the uk. *Work, employment and society*, 18(3):531–549, 2004.

[4] M De la Sen and A Ibeas. On an se (is)(ih) ar epidemic model with combined vaccination and antiviral controls for covid-19 pandemic. *Advances in Difference Equations*, 2021(1):1–30, 2021.

[5] Carolina Fransson and Pieter Trapman. Sir epidemics and vaccination on random graphs with clustering. *Journal of mathematical biology*, 78(7):2369–2398, 2019.

[6] Giulia Giordano, Franco Blanchini, Raffaele Bruno, Patrizio Colaneri, Alessandro Di Filippo, Angela Di Matteo, and Marta Colaneri. Modelling the covid-19 epidemic and implementation of population-wide interventions in italy. *Nature medicine*, 26(6):855–860, 2020.

[7] Giorgio Guzzetta, Flavia Riccardo, Valentina Marziano, Piero Poletti, Filippo Trentini, Antonino Bella, Xanthi Andrianou, Martina Del Manso, Massimo Fabiani, Stefania Bellino, et al. The impact of a nation-wide lockdown on covid-19 transmissibility in italy. *arXiv preprint arXiv:2004.12338*, 2020.

[8] Benjamin Ivorra, M Ruiz Ferrández, Marıa Vela-Pérez, and AM Ramos. Mathematical modeling of the spread of the coronavirus disease 2019 (covid-19) taking into account the undetected infections. the case of china. *Communications in nonlinear science and numerical simulation*, 88:105303, 2020.

[9] William Ogilvy Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character*, 115(772):700–721, 1927.

[10] John Hopkins University & Medicine. Mortality analyses. https://coronavirus.jhu.edu/data/mortality, 2021.

[11] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[12] Rijksoverheid. Coronavirus tijdlijn. https://www.rijksoverheid.nl/onderwerpen/coronavirus-tijdlijn, 2021.

[13] RIVM. Naleven gedragsregels. https://www.rivm.nl/gedragsonderzoek/maatregelen-welbevinden/naleven-gedragsregels#Testen, 2021.

[14] Chris Stokel-Walker. What we know about covid-19 reinfection so far. *British Medical Journal*, 372, 2021.

[15] W. van Bijsterveld. Coronagegevens van het rivm, cbs, nice, lcps & ecdc, dagelijks verzameld. `https://service.openinfo.nl/downloads/corona-cijfers-per-dag-op-gemeente-provincie-en-landelijk-niveau/`, 2021.

[16] CBS (Centraal Bureau voor Statistiek). Banen van werknemers naar woon- en werkregio. `https://opendata.cbs.nl/statline/portal.html?_la=nl&_catalog=CBS&tableId=83628NED&_theme=244`, 2021.

[17] Jacco Wallinga and Marc Lipsitch. How generation intervals shape the relationship between growth rates and reproductive numbers. *Proceedings of the Royal Society B: Biological Sciences*, 274(1609):599–604, 2007.

[18] Kai Wang. A study of cubic spline interpolation. *InSight: Rivier Academic Journal*, 9(2), 2013.

[19] Dirk Werner. *Funktionalanalysis*. Springer, 2018.

A Python code

A.1 Commuting Data

```python
import pandas as pd
import numpy as np

#defining the population vector
N = np.zeros(12)
N[0] = 582649   # 0: Groningen
N[1] = 646092   # 1: Friesland
N[2] = 488871   # 2: Drenthe
N[3] = 1162406  # 3: Overijssel
N[4] = 423021   # 4: Flevoland
N[5] = 2085952  # 5: Gelderland
N[6] = 1354834  # 6: Utrecht
N[7] = 2879527  # 7: Noord-Holland
N[8] = 3708696  # 8: Zuid-Holland
N[9] = 383488   # 9: Zeeland
N[10] = 2562955 # 10: Noord-Brabant
N[11] = 1117201 # 11: Limburg

x1 = pd.read_excel('CommutingData.xlsx')

T=[] #initializing T vector
c=np.zeros((len(N), len(N))) #initializing commuting matrix

#Selecting the right data from the table
Gr = x1.iloc[1, 1]
Fr = x1.iloc[7, 2]
Dr = x1.iloc[13, 2]
Ov = x1.iloc[19, 2]
Fl = x1.iloc[25, 2]
Ge = x1.iloc[31, 2]
Ut = x1.iloc[37, 2]
NH = x1.iloc[43, 2]
ZH = x1.iloc[49, 2]
Ze = x1.iloc[55, 2]
NB = x1.iloc[61, 2]
Li = x1.iloc[67, 2]
print(Ze)
provinces = [Gr, Fr, Dr, Ov, Fl, Ge, Ut, NH, ZH, Ze, NB, Li] #Vector ...
    with all provinces in them


i=0
while i<len(x1): #total length of the table
    j=0
    while j≤11: #iterating all provinces
        if x1.iloc[i,1] == provinces[j]:
            k=0
            while k≤11: #iterating all other provinces
                if k is not j:  #making sure that the commuting zero ...
                    parameter within the province stays zero
                    if x1.iloc[i,2] == provinces[k]:
                        T.append(x1.iat[i,4])
                        if len(T)==6:
                            c[j,k]=(sum(T)/6 * 1000)/N[j]  #Calculating ...
                                commuting parameter.
                            T=[]
                k += 1
        j+=1
    i+=1

cprime = np.zeros((len(N), len(N))) #Initializing the c prime matrix
i=0
while i< len(N):
    cprime[i, i] = sum(c[i,]) #calculating the elements of c'
    i+=1

i=0
while i < len(N):
    print('outgoing=', sum(c[i,:]))
    j=0
    incom =[]
    while j< len(N):
        if j is not i:
            incoming = c[j,i]*N[j]/N[i]
            incom.append(incoming)
        j+=1
    print('incoming=', sum(incom))
    i += 1
```

A.2 COVID-19 Data

```
import numpy as np
from scipy import optimize
import pandas as pd
import xlrd
import matplotlib.pyplot as plt
import numpy as np
from DataProvincie import c, N, cprime
from scipy import interpolate

#reading the necessary data sheets
x1 = pd.read_excel('Coronatest.xlsx', sheet_name='Besmettingen_verschil')
x2 = pd.read_excel('Coronatest.xlsx', sheet_name='Besmettingen')

#deleting unneccessary info from the data sheet
del x1['Regiocode']
del x1['Regiosoort']
del x1['Regionaam']
del x1['Provincienaam']
del x2['Regiocode']
del x2['Regiosoort']
del x2['Regionaam']
del x2['Provincienaam']

#initializing w and q
w = (33.5/7)/24         #work parameter
q= 1-0.76               #quarantaine paramter

#t=range(26*7)

#Deleting more unneccesary info from the data sheet
i=13
while i ≤ 380:
    x1=x1.drop(i)
    x2=x2.drop(i)
    i += 1

#Transpose and flipping the order of x1 and x2
x1=x1.T
x1=x1[::-1]
x2=x2.T
x2=x2[::-1]

#Initializing vectors for the susceptibles in each province
NederlandS=[]
DrentheS=[]
FlevolandS=[]
FrieslandS=[]
GelderlandS=[]
GroningenS=[]
LimburgS=[]
NoordBrabantS=[]
NoordHollandS=[]
OverijsselS=[]
UtrechtS=[]
ZeelandS=[]
ZuidHollandS=[]

#Initializing vectors for the infected in each province
NederlandI=[]
DrentheI=[]
FlevolandI=[]
FrieslandI=[]
GelderlandI=[]
GroningenI=[]
LimburgI=[]
NoordBrabantI=[]
NoordHollandI=[]
OverijsselI=[]
UtrechtI=[]
ZeelandI=[]
ZuidHollandI=[]

i=0
while i < len(x1):
    #adding the amount susceptibles per time step to the susceptible ...
        vector of each province
    NederlandS.append(sum(N)-x2.iat[i, 0])
    DrentheS.append(N[2]-x2.iat[i, 1])
    FlevolandS.append(N[4]-x2.iat[i, 2])
    FrieslandS.append(N[1]-x2.iat[i, 3])
    GelderlandS.append(N[5]-x2.iat[i, 4])
    GroningenS.append(N[0]-x2.iat[i, 5])
    LimburgS.append(N[11]-x2.iat[i, 6])
    NoordBrabantS.append(N[10]-x2.iat[i, 7])
    NoordHollandS.append(N[7]-x2.iat[i, 8])
    OverijsselS.append(N[3]-x2.iat[i, 9])
    UtrechtS.append(N[6]-x2.iat[i, 10])
    ZeelandS.append(N[9]-x2.iat[i, 11])
    ZuidHollandS.append(N[8]-x2.iat[i, 12])

    #adding the amount infected per time step to the infected vector of ...
        each province
    NederlandI.append(x1.iat[i, 0])
    DrentheI.append(x1.iat[i, 1])
    FlevolandI.append(x1.iat[i, 2])
    FrieslandI.append(x1.iat[i, 3])
    GelderlandI.append(x1.iat[i, 4])
    GroningenI.append(x1.iat[i, 5])
    LimburgI.append(x1.iat[i, 6])
    NoordBrabantI.append(x1.iat[i, 7])
    NoordHollandI.append(x1.iat[i, 8])
    OverijsselI.append(x1.iat[i, 9])
    UtrechtI.append(x1.iat[i, 10])
    ZeelandI.append(x1.iat[i, 11])
    ZuidHollandI.append(x1.iat[i, 12])
    i += 1


#initialzing matrices storing all susceptibles and infected persons for ...
    each province
DataS = np.zeros((12, len(x1)))
DataI = np.zeros((12, len(x1)))
#initialzing matrices storing all effective susceptibles and infected ...
    persons for each province
EffectiveI = np.zeros((12, len(x1)))
EffectiveS = np.zeros((12, len(x1)))
#initialzing matrices storing the derivatives of the amount ...
    susceptibles and infected persons for each province
dS = np.zeros((12, len(x1)))
dI = np.zeros((12, len(x1)))

#Defining the 5 different time periods
time=np.arange(len(x1))
Time1 = range(0,89)
Time2= range(89, 211)
Time3= range(211, 287)
Time4 = range(287, 421)
Time5= range(421, len(x1))

#Filling the susceptible matrix
DataS[0, :] = GroningenS
DataS[1, :] = FrieslandS
DataS[2, :] = DrentheS
DataS[3, :] = OverijsselS
DataS[4, :] = FlevolandS
DataS[5, :] = GelderlandS
DataS[6, :] = UtrechtS
DataS[7, :] = NoordHollandS
DataS[8, :] = ZuidHollandS
DataS[9, :] = ZeelandS
DataS[10, :] = NoordBrabantS
DataS[11, :] = LimburgS

#Filling the infected matrix
DataI[0, :] = GroningenI
DataI[1, :] = FrieslandI
DataI[2, :] = DrentheI
DataI[3, :] = OverijsselI
DataI[4, :] = FlevolandI
DataI[5, :] = GelderlandI
DataI[6, :] = UtrechtI
DataI[7, :] = NoordHollandI
DataI[8, :] = ZuidHollandI
DataI[9, :] = ZeelandI
```

```
DataI[10, :] = NoordBrabantI
DataI[11, :] = LimburgI

#Calculating the matrices to go from actual to effective amount of ...
    susceptibles and infected
M_S = np.identity(len(N)) - w * cprime + w * np.transpose(c)
M_I = np.identity(len(N)) - q*w * cprime + q*w * np.transpose(c)

#Calculating the effective amount of population
EffectiveN =M_S.dot(N.T)

#Calculating the effective amount of susceptibles and infected at each ...
    time step
i=0
while i < len(x1):
    EffectiveS[:,i] = M_S.dot(DataS[:,i])/EffectiveN
    EffectiveI[:,i] = M_I.dot(DataI[:,i])/EffectiveN
    i += 1

#Cubic spline interpolation for the effective susceptibles
csSGr = interpolate.CubicSpline(time, EffectiveS[0,:])
csSFr = interpolate.CubicSpline(time, EffectiveS[1,:])
csSDr = interpolate.CubicSpline(time, EffectiveS[2,:])
csSOv = interpolate.CubicSpline(time, EffectiveS[3,:])
csSFl = interpolate.CubicSpline(time, EffectiveS[4,:])
csSGe = interpolate.CubicSpline(time, EffectiveS[5,:])
csSUt = interpolate.CubicSpline(time, EffectiveS[6,:])
csSNH = interpolate.CubicSpline(time, EffectiveS[7,:])
csSZH = interpolate.CubicSpline(time, EffectiveS[8,:])
csSZe = interpolate.CubicSpline(time, EffectiveS[9,:])
csSNB = interpolate.CubicSpline(time, EffectiveS[10,:])
csSLi = interpolate.CubicSpline(time, EffectiveS[11,:])

#Storing all cubic splines for susceptibles in one vector
csS =[]
csS.append(csSGr)
csS.append(csSFr)
csS.append(csSDr)
csS.append(csSOv)
csS.append(csSFl)
csS.append(csSGe)
csS.append(csSUt)
csS.append(csSNH)
csS.append(csSZH)
csS.append(csSZe)
csS.append(csSNB)
csS.append(csSLi)

#Cubic spline interpolation for the effective infected
csIGr = interpolate.CubicSpline(time, EffectiveI[0,:])
csIFr = interpolate.CubicSpline(time, EffectiveI[1,:])
csIDr = interpolate.CubicSpline(time, EffectiveI[2,:])
csIOv = interpolate.CubicSpline(time, EffectiveI[3,:])
csIFl = interpolate.CubicSpline(time, EffectiveI[4,:])
csIGe = interpolate.CubicSpline(time, EffectiveI[5,:])
csIUt = interpolate.CubicSpline(time, EffectiveI[6,:])
csINH = interpolate.CubicSpline(time, EffectiveI[7,:])
csIZH = interpolate.CubicSpline(time, EffectiveI[8,:])
csIZe = interpolate.CubicSpline(time, EffectiveI[9,:])
csINB = interpolate.CubicSpline(time, EffectiveI[10,:])
csILi = interpolate.CubicSpline(time, EffectiveI[11,:])
```

```

#Storing all cubic splines for infected in one vector
csI =[]
csI.append(csIGr)
csI.append(csIFr)
csI.append(csIDr)
csI.append(csIOv)
csI.append(csIFl)
csI.append(csIGe)
csI.append(csIUt)
csI.append(csINH)
csI.append(csIZH)
csI.append(csIZe)
csI.append(csINB)
csI.append(csILi)

#initialzing matrix filled with -S*I
rightsideS = np.zeros((12,len(x1)))

#Filling the rightsideS matrix and the derivative matrices
i=0
while i<12:
    rightsideS[i,:] = - csS[i](time) * csI[i](time)
    dS[i,:] = csS[i](time, 1)
    dI[i,:] = csI[i](time, 1)
    EffectiveS[i,:] = csS[i](time)
    EffectiveI[i,:] = csI[i](time)
    i+=1

#Timeperiod1: 3 March - 1 June
#Making vectors with all data for timeperiod 1 to 5
dS1 = dS[:,Time1]
dS2 = dS[:,Time2]
dS3 = dS[:,Time3]
dS4 = dS[:,Time4]
dS5 = dS[:,Time5]

dI1 = dI[:,Time1]
dI2 = dI[:,Time2]
dI3 = dI[:,Time3]
dI4 = dI[:,Time4]
dI5 = dI[:,Time5]

rightsideS1 = rightsideS[:,Time1]
rightsideS2 = rightsideS[:,Time2]
rightsideS3 = rightsideS[:,Time3]
rightsideS4 = rightsideS[:,Time4]
rightsideS5 = rightsideS[:,Time5]

leftsideI = np.zeros((12, len(x1)))
leftsideI1 = leftsideI[:,Time1]
leftsideI2 = leftsideI[:,Time2]
leftsideI3 = leftsideI[:,Time3]
leftsideI4 = leftsideI[:,Time4]
leftsideI5 = leftsideI[:,Time5]


EffectiveS1 = EffectiveS[:, Time1]
EffectiveS2 = EffectiveS[:, Time2]
EffectiveS3 = EffectiveS[:, Time3]
EffectiveS4 = EffectiveS[:, Time4]
```

```
EffectiveS5 = EffectiveS[:, Time5]

EffectiveI1 = EffectiveI[:, Time1]
EffectiveI2 = EffectiveI[:, Time2]
EffectiveI3 = EffectiveI[:, Time3]
EffectiveI4 = EffectiveI[:, Time4]
EffectiveI5 = EffectiveI[:, Time5]

#Defining function to fit optimize curve
def func1(x, a):
    y = a*x
    return y

#Initializing vectors for k in each time period
k1 = np.zeros(12)
k2 = np.zeros(12)
k3 = np.zeros(12)
k4 = np.zeros(12)
k5 = np.zeros(12)

#Filling the k vectors by applying a least square curve fit to all vectors
i = 0
while i<12:
    k1[i] = optimize.curve_fit(func1, xdata=rightsideS1[i, :], ...
        ydata=dS1[i, :])[0]
    k2[i] = optimize.curve_fit(func1, xdata=rightsideS2[i, :], ...
        ydata=dS2[i, :])[0]
    k3[i] = optimize.curve_fit(func1, xdata=rightsideS3[i, :], ...
        ydata=dS3[i, :])[0]
    k4[i] = optimize.curve_fit(func1, xdata=rightsideS4[i, :], ...
        ydata=dS4[i, :])[0]
    k5[i] = optimize.curve_fit(func1, xdata=rightsideS5[i, :], ...
        ydata=dS5[i, :])[0]
    i +=1

#Filling the leftsideI matrix with (dI/dt)/I
i =0
while i < 12:
    leftsideI[i,:] = dI[i,:]/EffectiveI[i,:]
    leftsideI1[i,:] = dI1[i,:]/EffectiveI1[i,:]
    leftsideI2[i,:] = dI2[i,:]/EffectiveI2[i,:]
    leftsideI3[i,:] = dI3[i,:]/EffectiveI3[i,:]
    leftsideI4[i,:] = dI4[i,:]/EffectiveI4[i,:]
    leftsideI5[i,:] = dI5[i,:]/EffectiveI5[i,:]
    i += 1

#Defining new function for optimize curve fit
def func2(x, b, c):
    y = b*x-c
    return y

i=0
#Define matrix that stores kappa in first column and l in second column
l1=np.zeros((12,2))
l2=np.zeros((12,2))
l3=np.zeros((12,2))
l4=np.zeros((12,2))
l5=np.zeros((12,2))
while i<12:
    l1[i] = optimize.curve_fit(func2, xdata= EffectiveS1[i,:], ...
        ydata=leftsideI1[i,:])[0]
    l2[i] = optimize.curve_fit(func2, xdata = EffectiveS2[i,:], ...
        ydata=leftsideI2[i,:])[0]
    l3[i] = optimize.curve_fit(func2, xdata = EffectiveS3[i,:], ...
        ydata=leftsideI3[i,:])[0]
    l4[i] = optimize.curve_fit(func2, xdata = EffectiveS4[i,:], ...
        ydata=leftsideI4[i,:])[0]
    l5[i] = optimize.curve_fit(func2, xdata = EffectiveS5[i,:], ...
        ydata=leftsideI5[i,:])[0]
    i+=1


#Initialize empty norm vectors
norm1 = np.zeros(12)
norm2 = np.zeros(12)
norm3 = np.zeros(12)
norm4 = np.zeros(12)
norm5 = np.zeros(12)

#Normalize l to kappa value as calculated before
for i in range(len(N)):
    norm1[i] = l1[i,0]
    l1[i, 0] = k1[i]
    l1[i, 1] = l1[i,1]/norm1[i]*k1[0]

    norm2[i] = l2[i, 0]
    l2[i, 0] = k2[i]
    l2[i, 1] = l2[i, 1] / norm2[i] * k2[0]

    norm3[i] = l3[i, 0]
    l3[i, 0] = k3[i]
    l3[i, 1] = l3[i, 1] / norm3[i] * k3[0]

    norm4[i] = l4[i, 0]
    l4[i, 0] = k4[i]
    l4[i, 1] = l4[i, 1] / norm4[i] * k4[0]

    norm5[i] = l5[i, 0]
    l5[i, 0] = k5[i]
    l5[i, 1] = l5[i, 1] / norm5[i] * k5[0]

#Print kappa vectors
print('k1=', l1[:,0])
print('k2=', l2[:,0])
print('k3=', l3[:,0])
print('k4=', l4[:,0])
print('k5=', l5[:,0])

#Print l vectors
print('l1=', l1[:,1])
print('l2=', l2[:,1])
print('l3=', l3[:,1])
print('l4=', l4[:,1])
print('l5=', l5[:,1])
```

A.3 The Province Model

```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (11, 5)  #set default figure size
```

```
import numpy as np
from numpy import exp
from scipy.integrate import odeint
from DataProvincie import c, N, provinces, cprime
from DataCovid import l1, l2, l3, l4, l5, w, q, x1
d= 0.95 #fraction of people that didn't die

#initializing vectors for the effective rate of change of s, i and r
dseff = np.zeros(12)
dieff = np.zeros(12)
dreff = np.zeros(12)

#Calculating the matrice to go from actual to effective amount of ...
    susceptibles and its inverse
M_S = np.identity(len(N)) - w * cprime + w * np.transpose(c)
inverseM_S = np.linalg.inv(M_S)

#Calculating the matrice to go from actual to effective amount of ...
    infected and its inverse
M_I = np.identity(len(N)) - q*w * cprime + q*w * np.transpose(c)
inverseM_I = np.linalg.inv(M_I)

#Calculating the matrice to go from actual to effective amount of ...
    removed and its inverse
M_R = np.identity(len(N)) - d*w * cprime + d*w * np.transpose(c)
inverseM_R = np.linalg.inv(M_R)

#Calculating effective population for each province
EffectiveN =M_S.dot(N.T)

def F(x, t): #Function to calculate the differential equations ds, di, dr
    global counter
    #Loading in the necessary variables for the function:
    S_Gr, S_Fr, S_Dr, S_Ov, S_Fl, S_Ge, S_Ut, S_NH, S_ZH, S_Ze, S_NB, ...
        S_Li, I_Gr, I_Fr, I_Dr, I_Ov, I_Fl, I_Ge, I_Ut, I_NH, I_ZH, ...
        I_Ze, I_NB, I_Li, R_Gr, R_Fr, R_Dr, R_Ov, R_Fl, R_Ge, R_Ut, ...
        R_NH, R_ZH, R_Ze, R_NB, R_Li = x
    #Defining vector S, I and R
    S=[S_Gr, S_Fr, S_Dr, S_Ov, S_Fl, S_Ge, S_Ut, S_NH, S_ZH, S_Ze, ...
        S_NB, S_Li]
    I=[I_Gr, I_Fr, I_Dr, I_Ov, I_Fl, I_Ge, I_Ut, I_NH, I_ZH, I_Ze, ...
        I_NB, I_Li]
    R=[R_Gr, R_Fr, R_Dr, R_Ov, R_Fl, R_Ge, R_Ut, R_NH, R_ZH, R_Ze, ...
        R_NB, R_Li]

    #Timeperiod 1
    if t≤89:
        k = l1[:,0]
        l = l1[:,1]
    #Timeperiod 2
    elif t<211:
        k = l2[:, 0]
        l = l2[:, 1]
    elif t<287:
    #Timeperiod 3
        k = l3[:, 0]
        l = l3[:, 1]
    #Timeperiod 4
    elif t<421:
        k = l4[:, 0]
        l = l4[:, 1]
    #Timeperiod 5
    else:
        k = l5[:, 0]
        l = l5[:, 1]

    #Calculating effective fraction of suscetible, infected and removed
    Seff = M_S.dot(S)/EffectiveN
    Ieff = M_I.dot(I)/EffectiveN
    Reff = M_R.dot(R)/EffectiveN

    #Calculating the differential equations:
    i=0
    while i ≤ len(N)-1:
        dseff[i] = -k[i]*Seff[i]*Ieff[i]
        dieff[i] = (k[i]*Seff[i]*Ieff[i]) -  l[i]*Ieff[i]
        dreff[i] =  - l[i] * Ieff[i]
        i+=1

    #Calculating actual rate of change of susceptible, infected and removed
    ds = inverseM_S.dot(dseff)*EffectiveN
    di = inverseM_I.dot(dieff)*EffectiveN
    dr = inverseM_R.dot(dreff)*EffectiveN

    #Making sure the final output can be used in odeint
    result = np.reshape([ds, di, dr], len(S)*3)
    return result

# initial conditions of S, I and R (Every province one infection at the ...
    start)
i=0
Sstart=[]
Istart=[]
Rstart=[]
while i ≤ len(N)-1:
    Sstart.append((N[i]-1))  #Making sure everyone but 1 person is ...
        susceptible at the start
    Istart.append(1)          #Making sure only 1 person is infected ...
        at the start
    Rstart.append(0)               #Making sure there are no removed ...
        people at the start
    i += 1

# initial conditions of S, I and R (Every province according to actual ...
    data)
#Sstart=np.ones(12)
#Istart=np.zeros(12)
#Rstart=np.zeros(12)

#Sstart[0] = N[0] #0 infections in Groningen
#Sstart[1] = N[1] #0 infections in Friesland
#Sstart[2] = N[2] - 2 # 2 infections in Drenthe
#Sstart[3] = N[3] #0 infections in Overijssel
#Sstart[4] = N[4] #0 infections in Flevoland
#Sstart[5] = N[5] #0 infections in Gelderland
#Sstart[6] =N[6] - 3 # 3 Infections in Utrecht
#Sstart[7] = N[7] - 3 # 3 Infections in Noord-Holland
#Sstart[8] = N[8] - 3 # 3 Infections in Zuid-Holland
#Sstart[9] = N[9] # 0 Infections in Zeeland
#Sstart[10] = N[10] - 12 # 12 Infections in Noord-Brabant
#Sstart[11] = N[11] #0 infections in Limburg

```

```
#Istart[2] = 2 # 2 infections in Drenthe
#Istart[6] = 3 # 3 Infections in Utrecht
#Istart[7] = 3 # 3 Infections in Noord-Holland
#Istart[8] = 3 # 3 Infections in Zuid-Holland
#Istart[10] = 12 # 12 Infections in Noord-Brabant


x_0 = np.reshape([Sstart, Istart, Rstart], len(N)*3)

#Making a time vector for the plots.
#actual time
t_length = len(x1)
grid_size = len(x1)

t_vec = np.linspace(0, t_length, grid_size)


def solve_path(t_vec, x_init=x_0):
    """
    Solve for S(t), I(t) and R(t) via numerical integration,
    given the time path for R0.
    """
    G = lambda x, t: F(x, t)
    S_Gr_path, S_Fr_path, S_Dr_path, S_Ov_path, S_Fl_path, S_Ge_path, ...
        S_Ut_path, S_NH_path, S_ZH_path, S_Ze_path, S_NB_path, ...
        S_Li_path, I_Gr_path, I_Fr_path, I_Dr_path, I_Ov_path, ...
        I_Fl_path, I_Ge_path, I_Ut_path, I_NH_path, I_ZH_path, ...
        I_Ze_path, I_NB_path, I_Li_path, R_Gr_path, R_Fr_path, ...
        R_Dr_path, R_Ov_path, R_Fl_path, R_Ge_path, R_Ut_path, ...
        R_NH_path, R_ZH_path, R_Ze_path, R_NB_path, R_Li_path = ...
        odeint(G, x_init, t_vec).transpose()

    return S_Gr_path, S_Fr_path, S_Dr_path, S_Ov_path, S_Fl_path, ...
        S_Ge_path, S_Ut_path, S_NH_path, S_ZH_path, S_Ze_path, ...
        S_NB_path, S_Li_path, I_Gr_path, I_Fr_path, I_Dr_path, ...
        I_Ov_path, I_Fl_path, I_Ge_path, I_Ut_path, I_NH_path, ...
        I_ZH_path, I_Ze_path, I_NB_path, I_Li_path, R_Gr_path, ...
        R_Fr_path, R_Dr_path, R_Ov_path, R_Fl_path, R_Ge_path, ...
        R_Ut_path, R_NH_path, R_ZH_path, R_Ze_path, R_NB_path, R_Li_path

#Initializing vectors to store the results
S_prov, I_prov, R_prov = [], [], []
S_NL_paths, I_NL_paths, R_NL_paths = [], [], []

#Calculating the results:
S_Gr_path, S_Fr_path, S_Dr_path, S_Ov_path, S_Fl_path, S_Ge_path, ...
    S_Ut_path, S_NH_path, S_ZH_path, S_Ze_path, S_NB_path, S_Li_path, ...
    I_Gr_path, I_Fr_path, I_Dr_path, I_Ov_path, I_Fl_path, I_Ge_path, ...
    I_Ut_path, I_NH_path, I_ZH_path, I_Ze_path, I_NB_path, I_Li_path, ...
    R_Gr_path, R_Fr_path, R_Dr_path, R_Ov_path, R_Fl_path, R_Ge_path, ...
    R_Ut_path, R_NH_path, R_ZH_path, R_Ze_path, R_NB_path, R_Li_path = ...
    solve_path(t_vec)
#Making the vector with the paths of the susceptibles in all provinces.
S_prov.append(S_Gr_path)
S_prov.append(S_Fr_path)
S_prov.append(S_Dr_path)
S_prov.append(S_Ov_path)
S_prov.append(S_Fl_path)
S_prov.append(S_Ge_path)
S_prov.append(S_Ut_path)
S_prov.append(S_NH_path)
S_prov.append(S_ZH_path)
S_prov.append(S_Ze_path)
S_prov.append(S_NB_path)
S_prov.append(S_Li_path)
#Making the vector with the paths of the infected in all provinces.
I_prov.append(I_Gr_path)
I_prov.append(I_Fr_path)
I_prov.append(I_Dr_path)
I_prov.append(I_Ov_path)
I_prov.append(I_Fl_path)
I_prov.append(I_Ge_path)
I_prov.append(I_Ut_path)
I_prov.append(I_NH_path)
I_prov.append(I_ZH_path)
I_prov.append(I_Ze_path)
I_prov.append(I_NB_path)
I_prov.append(I_Li_path)
#Making the vector with the paths of the Removed in all provinces.
R_prov.append(R_Gr_path)
R_prov.append(R_Fr_path)
R_prov.append(R_Dr_path)
R_prov.append(R_Ov_path)
R_prov.append(R_Fl_path)
R_prov.append(R_Ge_path)
R_prov.append(R_Ut_path)
R_prov.append(R_NH_path)
R_prov.append(R_ZH_path)
R_prov.append(R_Ze_path)
R_prov.append(R_NB_path)
R_prov.append(R_Li_path)
#Calculating the total fraction of S, I and R in the Netherlands.
S_NL_path = (S_Gr_path+S_Fr_path+S_Dr_path+S_Ov_path+ S_Fl_path+ ...
    S_Ge_path+S_Ut_path+S_NH_path + S_ZH_path+ S_Ze_path+ S_NB_path + ...
    S_Li_path)
I_NL_path = (I_Gr_path+I_Fr_path+I_Dr_path+I_Ov_path+ I_Fl_path + ...
    I_Ge_path +I_Ut_path +I_NH_path + I_ZH_path + I_Ze_path + I_NB_path ...
    + I_Li_path)
R_NL_path = (R_Gr_path+R_Fr_path+R_Dr_path+R_Ov_path+ R_Fl_path + ...
    R_Ge_path +R_Ut_path+R_NH_path+ R_ZH_path+ R_Ze_path + R_NB_path+ ...
    R_Li_path)/sum(N)

colors = ['red', 'blue', 'green', 'orange', 'yellow', 'black', 'grey', ...
    'purple', 'pink', 'magenta', 'cyan', 'deepskyblue']

##Plotting the susceptibles
#Plotting the time periods
#plt.axvline(89, ls='--', color='grey', lw= 0.5)
#plt.axvline(211, ls='--', color='grey', lw=0.5)
#plt.axvline(287, ls='--', color='grey', lw=0.5)
#plt.axvline(421, ls='--', color='grey', lw=0.5)

#for i in range(len(S_prov)):
#    plt.plot(t_vec, S_prov[i]/N[i], label=provinces[i])
#plt.xlabel('Time (days)')
#plt.ylabel('Fraction of total population')
#plt.title('Susceptibles in each province')
#plt.legend()
#plt.show()

##Plotting the infected
#Plotting the timeperiods
```

```
plt.axvline(89, ls='--', color='grey', lw= 0.5)
plt.axvline(211, ls='--', color='grey', lw=0.5)
plt.axvline(287, ls='--', color='grey', lw=0.5)
plt.axvline(421, ls='--', color='grey', lw=0.5)

for i in range(len(N)):
    plt.plot(t_vec, I_prov[i]/N[i], label=provinces[i], color= colors[i])
plt.xlabel('Time (days)')
plt.ylabel('Fraction of total population')
plt.title('Infected people in each province')
plt.legend()
plt.show()

##Plotting the removed
#Plotting the timeperiods
#plt.axvline(89, ls='--', color='grey', lw= 0.5)
#plt.axvline(211, ls='--', color='grey', lw=0.5)
#plt.axvline(287, ls='--', color='grey', lw=0.5)
#plt.axvline(421, ls='--', color='grey', lw=0.5)
#for i in range(len(R_prov)):
#    plt.plot(t_vec, R_prov[i]/N[i], label=provinces[i])
#plt.xlabel('Time (days)')
#plt.ylabel('Fraction of total population')
#plt.title('Removed people in each province')
#plt.legend()
#plt.show()

##printing the results
print('max I_NL =',max(I_NL_path))
print('sum I_NL =',sum(I_NL_path))
print('max I_Gr =',max(I_Gr_path))
print('sum I_Gr =',sum(I_Gr_path))
print('max I_Fr =',max(I_Fr_path))
print('sum I_Fr =',sum(I_Fr_path))
print('max I_Dr =',max(I_Dr_path))
print('sum I_Dr =',sum(I_Dr_path))
print('max I_Ov =',max(I_Ov_path))
print('sum I_Ov =',sum(I_Ov_path))
print('max I_Fl =',max(I_Fl_path))
print('sum I_Fl =',sum(I_Fl_path))
print('max I_Ge =',max(I_Ge_path))
print('sum I_Ge =',sum(I_Ge_path))
print('max I_Ut =',max(I_Ut_path))
print('sum I_Ut =',sum(I_Ut_path))
print('max I_NH =',max(I_NH_path))
print('sum I_NH =',sum(I_NH_path))
print('max I_ZH =',max(I_ZH_path))
print('sum I_ZH =',sum(I_ZH_path))
print('max I_Ze =',max(I_Ze_path))
print('sum I_Ze =',sum(I_Ze_path))
print('max I_NB =',max(I_NB_path))
print('sum I_NB =',sum(I_NB_path))
print('max I_Li =',max(I_Li_path))
print('sum I_Li =',sum(I_Li_path))

print('deaths NL=', sum(I_NL_path)*0.011)
print('deaths Gr=',sum(I_Gr_path)*0.011)
print('deaths Fr=',sum(I_Fr_path)*0.011)
print('deaths Dr=',sum(I_Dr_path)*0.011)
print('deaths Ov=',sum(I_Ov_path)*0.011)
print('deaths Fl=',sum(I_Fl_path)*0.011)
print('deaths Ge=',sum(I_Ge_path)*0.011)
print('deaths Ut=',sum(I_Ut_path)*0.011)
print('deaths NH=',sum(I_NH_path)*0.011)
print('deaths ZH=',sum(I_ZH_path)*0.011)
print('deaths Ze=',sum(I_Ze_path)*0.011)
print('deaths NB=',sum(I_NB_path)*0.011)
print('deaths Li=',sum(I_Li_path)*0.011)

commutingI = np.zeros(13)
commutingI[0] = sum(I_NL_path)
commutingI[1] = sum(I_Gr_path)
commutingI[2] = sum(I_Fr_path)
commutingI[3] = sum(I_Dr_path)
commutingI[4] = sum(I_Ov_path)
commutingI[5] = sum(I_Fl_path)
commutingI[6] = sum(I_Ge_path)
commutingI[7] = sum(I_Ut_path)
commutingI[8] = sum(I_NH_path)
commutingI[9] = sum(I_ZH_path)
commutingI[10] = sum(I_Ze_path)
commutingI[11] = sum(I_NB_path)
commutingI[12] = sum(I_Li_path)

print('fraction I_NL =', sum(I_NL_path)/sum(N))
print('fraction I_Gr =',sum(I_Gr_path)/N[0])
print('fraction I_Fr =',sum(I_Fr_path)/N[1])
print('fraction I_Dr =',sum(I_Dr_path)/N[2])
print('fraction I_Ov =',sum(I_Ov_path)/N[3])
print('fraction I_Fl =',sum(I_Fl_path)/N[4])
print('fraction I_Ge =',sum(I_Ge_path)/N[5])
print('fraction I_Ut =',sum(I_Ut_path)/N[6])
print('fraction I_NH =',sum(I_NH_path)/N[7])
print('fraction I_ZH =',sum(I_ZH_path)/N[8])
print('fraction I_Ze =',sum(I_Ze_path)/N[9])
print('fraction I_NB =',sum(I_NB_path)/N[10])
print('fraction I_Li =',sum(I_Li_path)/N[11])

##Plotting the entire Netherlands
#Plotting the timezones
#plt.axvline(89, ls='--', color='grey', lw= 0.5)
#plt.axvline(211, ls='--', color='grey', lw=0.5)
#plt.axvline(287, ls='--', color='grey', lw=0.5)
#plt.axvline(421, ls='--', color='grey', lw=0.5)
#plt.plot(t_vec, S_NL_path/sum(N), label ='S (Commuting is allowed)')
#plt.plot(t_vec, I_NL_path/sum(N), label ='I (Commuting is allowed)')
#plt.plot(t_vec, R_NL_path/sum(N), label ='R (Commuting is not allowed)')
#plt.xlabel('Time (days)')
#plt.ylabel('Fraction of total population')
#plt.title('Infected people in the Netherlands')
#plt.legend()
#plt.show()

print('COMMUTING IS NOT ALLOWED')

#Changing c
c=np.zeros((len(N), len(N)))
cprime = np.zeros((len(N), len(N)))

#initializing vectors for the effective rate of change of s, i and r
dseff = np.zeros(12)
dieff = np.zeros(12)
```

```
dreff = np.zeros(12)

#Calculating the matrice to go from actual to effective amount of ...
    susceptibles and its inverse
M_S = np.identity(len(N)) - w * cprime + w * np.transpose(c)
inverseM_S = np.linalg.inv(M_S)

#Calculating the matrice to go from actual to effective amount of ...
    infected and its inverse
M_I = np.identity(len(N)) - q*w * cprime + q*w * np.transpose(c)
inverseM_I = np.linalg.inv(M_I)

#Calculating the matrice to go from actual to effective amount of ...
    removed and its inverse
M_R = np.identity(len(N)) - d*w * cprime + d*w * np.transpose(c)
inverseM_R = np.linalg.inv(M_R)

#Calculating effective population for each province
EffectiveN =M_S.dot(N.T)

def F(x, t): #Function to calculate the differential equations ds, di, dr
    global counter
    #Loading in the necessary variables for the function:
    S_Gr, S_Fr, S_Dr, S_Ov, S_Fl, S_Ge, S_Ut, S_NH, S_ZH, S_Ze, S_NB, ...
        S_Li, I_Gr, I_Fr, I_Dr, I_Ov, I_Fl, I_Ge, I_Ut, I_NH, I_ZH, ...
        I_Ze, I_NB, I_Li, R_Gr, R_Fr, R_Dr, R_Ov, R_Fl, R_Ge, R_Ut, ...
        R_NH, R_ZH, R_Ze, R_NB, R_Li = x
    #Defining vector S, I and R
    S=[S_Gr, S_Fr, S_Dr, S_Ov, S_Fl, S_Ge, S_Ut, S_NH, S_ZH, S_Ze, ...
        S_NB, S_Li]
    I=[I_Gr, I_Fr, I_Dr, I_Ov, I_Fl, I_Ge, I_Ut, I_NH, I_ZH, I_Ze, ...
        I_NB, I_Li]
    R=[R_Gr, R_Fr, R_Dr, R_Ov, R_Fl, R_Ge, R_Ut, R_NH, R_ZH, R_Ze, ...
        R_NB, R_Li]

    #Timeperiod 1
    if t≤89:
        k = l1[:,0]
        l = l1[:,1]
    #Timeperiod 2
    elif t<211:
        k = l2[:, 0]
        l = l2[:, 1]
    elif t<287:
    #Timeperiod 3
        k = l3[:, 0]
        l = l3[:, 1]
    #Timeperiod 4
    elif t<421:
        k = l4[:, 0]
        l = l4[:, 1]
    #Timeperiod 5
    else:
        k = l5[:, 0]
        l = l5[:, 1]

    #Calculating effective fraction of suscetible, infected and removed
    Seff = M_S.dot(S)/EffectiveN
    Ieff = M_I.dot(I)/EffectiveN
    Reff = M_R.dot(R)/EffectiveN

    #Calculating the differential equations:
    i=0
    while i ≤ len(N)-1:
        dseff[i] = -k[i]*Seff[i]*Ieff[i]
        dieff[i] = (k[i]*Seff[i]*Ieff[i]) -  l[i]*Ieff[i]
        dreff[i] =  - l[i] * Ieff[i]
        i+=1

    #Calculating actual rate of change of susceptible, infected and removed
    ds = inverseM_S.dot(dseff)*EffectiveN
    di = inverseM_I.dot(dieff)*EffectiveN
    dr = inverseM_R.dot(dreff)*EffectiveN

    #Making sure the final output can be used in odeint
    result = np.reshape([ds, di, dr], len(S)*3)
    return result

# initial conditions of S, I and R (Every province one infection at the ...
    start)
i=0
Sstart=[]
Istart=[]
Rstart=[]
while i ≤ len(N)-1:
    Sstart.append((N[i]-1))  #Making sure everyone but 1 person is ...
        susceptible at the start
    Istart.append(1)           #Making sure only 1 person is infected ...
        at the start
    Rstart.append(0)                #Making sure there are no removed ...
        people at the start
    i += 1

# initial conditions of S, I and R (Every province according to actual ...
    data)
#Sstart=np.ones(12)
#Istart=np.zeros(12)
#Rstart=np.zeros(12)

#Sstart[0] = N[0] #0 infections in Groningen
#Sstart[1] = N[1] #0 infections in Friesland
#Sstart[2] = N[2] - 2 # 2 infections in Drenthe
#Sstart[3] = N[3] #0 infections in Overijssel
#Sstart[4] = N[4] #0 infections in Flevoland
#Sstart[5] = N[5] #0 infections in Gelderland
#Sstart[6] =N[6] - 3 # 3 Infections in Utrecht
#Sstart[7] = N[7] - 3 # 3 Infections in Noord-Holland
#Sstart[8] = N[8] - 3 # 3 Infections in Zuid-Holland
#Sstart[9] = N[9] # 0 Infections in Zeeland
#Sstart[10] = N[10] - 12 # 12 Infections in Noord-Brabant
#Sstart[11] = N[11] #0 infections in Limburg

#Istart[2] = 2 # 2 infections in Drenthe
#Istart[6] = 3 # 3 Infections in Utrecht
#Istart[7] = 3 # 3 Infections in Noord-Holland
#Istart[8] = 3 # 3 Infections in Zuid-Holland
#Istart[10] = 12 # 12 Infections in Noord-Brabant

x_0 = np.reshape([Sstart, Istart, Rstart], len(N)*3)

#Making a time vector for the plots.
#Actual time
```

```
t_length = len(x1)
grid_size = len(x1)

t_vec = np.linspace(0, t_length, grid_size)


def solve_path(t_vec, x_init=x_0):
    """
    Solve for S(t), I(t) and R(t) via numerical integration,
    given the time path for R0.
    """
    G = lambda x, t: F(x, t)
    S_Gr_path, S_Fr_path, S_Dr_path, S_Ov_path, S_Fl_path, S_Ge_path, ...
        S_Ut_path, S_NH_path, S_ZH_path, S_Ze_path, S_NB_path, ...
        S_Li_path, I_Gr_path, I_Fr_path, I_Dr_path, I_Ov_path, ...
        I_Fl_path, I_Ge_path, I_Ut_path, I_NH_path, I_ZH_path, ...
        I_Ze_path, I_NB_path, I_Li_path, R_Gr_path, R_Fr_path, ...
        R_Dr_path, R_Ov_path, R_Fl_path, R_Ge_path, R_Ut_path, ...
        R_NH_path, R_ZH_path, R_Ze_path, R_NB_path, R_Li_path = ...
        odeint(G, x_init, t_vec).transpose()

    return S_Gr_path, S_Fr_path, S_Dr_path, S_Ov_path, S_Fl_path, ...
        S_Ge_path, S_Ut_path, S_NH_path, S_ZH_path, S_Ze_path, ...
        S_NB_path, S_Li_path, I_Gr_path, I_Fr_path, I_Dr_path, ...
        I_Ov_path, I_Fl_path, I_Ge_path, I_Ut_path, I_NH_path, ...
        I_ZH_path, I_Ze_path, I_NB_path, I_Li_path, R_Gr_path, ...
        R_Fr_path, R_Dr_path, R_Ov_path, R_Fl_path, R_Ge_path, ...
        R_Ut_path, R_NH_path, R_ZH_path, R_Ze_path, R_NB_path, R_Li_path

#Initializing vectors to store the results
S_prov, I_prov, R_prov = [], [], []
S_NL_paths, I_NL_paths, R_NL_paths = [], [], []

#Calculating the results:
S_Gr_path, S_Fr_path, S_Dr_path, S_Ov_path, S_Fl_path, S_Ge_path, ...
    S_Ut_path, S_NH_path, S_ZH_path, S_Ze_path, S_NB_path, S_Li_path, ...
    I_Gr_path, I_Fr_path, I_Dr_path, I_Ov_path, I_Fl_path, I_Ge_path, ...
    I_Ut_path, I_NH_path, I_ZH_path, I_Ze_path, I_NB_path, I_Li_path, ...
    R_Gr_path, R_Fr_path, R_Dr_path, R_Ov_path, R_Fl_path, R_Ge_path, ...
    R_Ut_path, R_NH_path, R_ZH_path, R_Ze_path, R_NB_path, R_Li_path = ...
    solve_path(t_vec)
#Making the vector with the paths of the susceptibles in all provinces.
S_prov.append(S_Gr_path)
S_prov.append(S_Fr_path)
S_prov.append(S_Dr_path)
S_prov.append(S_Ov_path)
S_prov.append(S_Fl_path)
S_prov.append(S_Ge_path)
S_prov.append(S_Ut_path)
S_prov.append(S_NH_path)
S_prov.append(S_ZH_path)
S_prov.append(S_Ze_path)
S_prov.append(S_NB_path)
S_prov.append(S_Li_path)
#Making the vector with the paths of the infected in all provinces.
I_prov.append(I_Gr_path)
I_prov.append(I_Fr_path)
I_prov.append(I_Dr_path)
I_prov.append(I_Ov_path)
I_prov.append(I_Fl_path)
I_prov.append(I_Ge_path)
I_prov.append(I_Ut_path)
I_prov.append(I_NH_path)
I_prov.append(I_ZH_path)
I_prov.append(I_Ze_path)
I_prov.append(I_NB_path)
I_prov.append(I_Li_path)
#Making the vector with the paths of the Removed in all provinces.
R_prov.append(R_Gr_path)
R_prov.append(R_Fr_path)
R_prov.append(R_Dr_path)
R_prov.append(R_Ov_path)
R_prov.append(R_Fl_path)
R_prov.append(R_Ge_path)
R_prov.append(R_Ut_path)
R_prov.append(R_NH_path)
R_prov.append(R_ZH_path)
R_prov.append(R_Ze_path)
R_prov.append(R_NB_path)
R_prov.append(R_Li_path)
#Calculating the total fraction of S, I and R in the Netherlands.
S_NL_path = (S_Gr_path+S_Fr_path+S_Dr_path+S_Ov_path+ S_Fl_path+ ...
    S_Ge_path+S_Ut_path+S_NH_path + S_ZH_path+ S_Ze_path+ S_NB_path + ...
    S_Li_path)
I_NL_path = (I_Gr_path+I_Fr_path+I_Dr_path+I_Ov_path+ I_Fl_path + ...
    I_Ge_path +I_Ut_path +I_NH_path + I_ZH_path + I_Ze_path + I_NB_path ...
    + I_Li_path)
R_NL_path = (R_Gr_path+R_Fr_path+R_Dr_path+R_Ov_path+ R_Fl_path + ...
    R_Ge_path +R_Ut_path+R_NH_path+ R_ZH_path+ R_Ze_path + R_NB_path+ ...
    R_Li_path)/sum(N)

colors = ['red', 'blue', 'green', 'orange', 'yellow', 'black', 'grey', ...
    'purple', 'pink', 'magenta', 'cyan', 'deepskyblue']

##Plotting the susceptibles
#Plotting the time periods
#plt.axvline(89, ls='--', color='grey', lw= 0.5)
#plt.axvline(211, ls='--', color='grey', lw=0.5)
#plt.axvline(287, ls='--', color='grey', lw=0.5)
#plt.axvline(421, ls='--', color='grey', lw=0.5)

#for i in range(len(S_prov)):
#    plt.plot(t_vec, S_prov[i]/N[i], label=provinces[i])
#plt.xlabel('Time (days)')
#plt.ylabel('Fraction of total population')
#plt.title('Susceptibles in each province')
#plt.legend()
#plt.show()

##Plotting the infected
#Plotting the timeperiods
plt.axvline(89, ls='--', color='grey', lw= 0.5)
plt.axvline(211, ls='--', color='grey', lw=0.5)
plt.axvline(287, ls='--', color='grey', lw=0.5)
plt.axvline(421, ls='--', color='grey', lw=0.5)

for i in range(len(N)):
    plt.plot(t_vec, I_prov[i]/N[i], label=provinces[i], color= colors[i])
plt.xlabel('Time (days)')
plt.ylabel('Fraction of total population')
plt.title('Infected people in each province')
plt.legend()
plt.show()

##Plotting the removed
#Plotting the timeperiods
#plt.axvline(89, ls='--', color='grey', lw= 0.5)
#plt.axvline(211, ls='--', color='grey', lw=0.5)
#plt.axvline(287, ls='--', color='grey', lw=0.5)
#plt.axvline(421, ls='--', color='grey', lw=0.5)
#for i in range(len(R_prov)):
#    plt.plot(t_vec, R_prov[i]/N[i], label=provinces[i])
#plt.xlabel('Time (days)')
#plt.ylabel('Fraction of total population')
#plt.title('Removed people in each province')
#plt.legend()
#plt.show()

##printing the results
print('max I_NL =',max(I_NL_path))
print('sum I_NL =',sum(I_NL_path))
print('max I_Gr =',max(I_Gr_path))
print('sum I_Gr =',sum(I_Gr_path))
print('max I_Fr =',max(I_Fr_path))
print('sum I_Fr =',sum(I_Fr_path))
print('max I_Dr =',max(I_Dr_path))
print('sum I_Dr =',sum(I_Dr_path))
print('max I_Ov =',max(I_Ov_path))
print('sum I_Ov =',sum(I_Ov_path))
print('max I_Fl =',max(I_Fl_path))
print('sum I_Fl =',sum(I_Fl_path))
print('max I_Ge =',max(I_Ge_path))
print('sum I_Ge =',sum(I_Ge_path))
print('max I_Ut =',max(I_Ut_path))
print('sum I_Ut =',sum(I_Ut_path))
print('max I_NH =',max(I_NH_path))
print('sum I_NH =',sum(I_NH_path))
print('max I_ZH =',max(I_ZH_path))
print('sum I_ZH =',sum(I_ZH_path))
print('max I_Ze =',max(I_Ze_path))
print('sum I_Ze =',sum(I_Ze_path))
print('max I_NB =',max(I_NB_path))
print('sum I_NB =',sum(I_NB_path))
print('max I_Li =',max(I_Li_path))
print('sum I_Li =',sum(I_Li_path))

print('deaths NL=', sum(I_NL_path)*0.011)
print('deaths Gr=',sum(I_Gr_path)*0.011)
print('deaths Fr=',sum(I_Fr_path)*0.011)
print('deaths Dr=',sum(I_Dr_path)*0.011)
print('deaths Ov=',sum(I_Ov_path)*0.011)
print('deaths Fl=',sum(I_Fl_path)*0.011)
print('deaths Ge=',sum(I_Ge_path)*0.011)
print('deaths Ut=',sum(I_Ut_path)*0.011)
print('deaths NH=',sum(I_NH_path)*0.011)
print('deaths ZH=',sum(I_ZH_path)*0.011)
print('deaths Ze=',sum(I_Ze_path)*0.011)
print('deaths NB=',sum(I_NB_path)*0.011)
print('deaths Li=',sum(I_Li_path)*0.011)

notcommutingI = np.zeros(13)
notcommutingI[0] = sum(I_NL_path)
notcommutingI[1] = sum(I_Gr_path)
```

```
notcommutingI[2] = sum(I_Fr_path)
notcommutingI[3] = sum(I_Dr_path)
notcommutingI[4] = sum(I_Ov_path)
notcommutingI[5] = sum(I_Fl_path)
notcommutingI[6] = sum(I_Ge_path)
notcommutingI[7] = sum(I_Ut_path)
notcommutingI[8] = sum(I_NH_path)
notcommutingI[9] = sum(I_ZH_path)
notcommutingI[10] = sum(I_Ze_path)
notcommutingI[11] = sum(I_NB_path)
notcommutingI[12] = sum(I_Li_path)

change = (notcommutingI-commutingI)/commutingI * 1001

print('percentage change =', change)

print('fraction I_NL =', sum(I_NL_path)/sum(N))
print('fraction I_Gr =',sum(I_Gr_path)/N[0])
print('fraction I_Fr =',sum(I_Fr_path)/N[1])
print('fraction I_Dr =',sum(I_Dr_path)/N[2])
print('fraction I_Ov =',sum(I_Ov_path)/N[3])
print('fraction I_Fl =',sum(I_Fl_path)/N[4])
print('fraction I_Ge =',sum(I_Ge_path)/N[5])
print('fraction I_Ut =',sum(I_Ut_path)/N[6])
print('fraction I_NH =',sum(I_NH_path)/N[7])
print('fraction I_ZH =',sum(I_ZH_path)/N[8])
print('fraction I_Ze =',sum(I_Ze_path)/N[9])
print('fraction I_NB =',sum(I_NB_path)/N[10])
print('fraction I_Li =',sum(I_Li_path)/N[11])

##Plotting the entire Netherlands
#Plotting the timezones
plt.axvline(89, ls='--', color='grey', lw= 0.5)
plt.axvline(211, ls='--', color='grey', lw=0.5)
plt.axvline(287, ls='--', color='grey', lw=0.5)
plt.axvline(421, ls='--', color='grey', lw=0.5)
#plt.plot(t_vec, S_NL_path/sum(N), label ='S (Commuting is not allowed)')
#plt.show()
#plt.plot(t_vec, I_NL_path/sum(N), label ='I (Commuting is not allowed)')
#plt.plot(t_vec, R_NL_path/sum(N), label ='R (Commuting is not allowed)')
plt.xlabel('Time (days)')
plt.ylabel('Fraction of total population')
plt.title('Infected people in the Netherlands')
plt.legend()
#plt.show()
```

A.4 Numerical analysis invertibility

```
from DataProvincie import c, cprime, N
import numpy as np

#setting w, q, and d to their maximum value
w = 1
q = 1
d = 1

#setting cprime and c to their maximum value
for i in range(len(N)):
```

```python
    cprime[i,i] = 1
    for j in range(len(N)):
        if j is not i:
            c[i,j]=1/11

#calculating the inverses
M_S = np.identity(len(N)) - w * cprime + w * np.transpose(c)
inverseM_S = np.linalg.inv(M_S)

M_I = np.identity(len(N)) - q*w * cprime + q*w * np.transpose(c)
inverseM_I = np.linalg.inv(M_I)

M_R = np.identity(len(N)) - d*w * cprime + d*w * np.transpose(c)
inverseM_R = np.linalg.inv(M_R)

print(inverseM_I)
print(inverseM_R)
print(inverseM_S)
```