CARLETON UNIVERSITY

SCHOOL OF
MATHEMATICS AND STATISTICS

HONOURS PROJECT

TITLE: Review and Implementation of an epidemiological model for COVID-19

AUTHOR: James Britton

SUPERVISOR: Prof. Emmanuel Lorin

DATE: August 19th, 2021

Contents

1 Introduction

Since the onset of the **COVID-19** epidemic caused by the novel **SARS-CoV2** coronavirus, peoples' lives around the world have been severely impacted by the disease directly and by restrictions placed on them by governments around the world to limit the spread of the disease and prevent healthcare systems from collapsing. The goal of the restrictions is to limit the spread of the **COVID-19** virus. The virus is spread from water droplets containing the virus which are produced from an infected person's: cough, sneeze or simple respiration. It is of interest to governments to model the pandemic's evolution, within their populations, and attempt to put in place relevant **interventions** (i.e. restrictions) to minimize deaths while also minimizing other costs under consideration, such as: lowering the GDP (gross domestic product), economic challenges to families and business, mental health and other more insidious effects on persons health due to delayed routine or elective procedures/surgeries.

Governments have utilized many models to gain insights of how the disease will spread in their populations. Models can be used to analysis the 'what-ifs', how well does one intervention work versus another and what are the costs for each intervention. They can also help us to understand what would happen if the disease mutates into a more contagious one.

In this work we will be examining a model [9] that was published in March 2020 (early stage of the pandemics first wave), and apply it to the population of the Province of Ontario. In Ontario there were interventions that were imposed and lifted based off of the case load; hence, we modified the model so that the interventions would change based off of the case load. These modifications produced waves in the case loads.

2 Extended SEIR Model

The standard epidemiological model is called a SEIR model if the populations are divided into the following sub-populations: **S**usceptible, **E**xposed, **I**nfectious, and **R**ecovered [1]. The model being presented is an extended SEIR model with the ad-

ditional sub-populations: **H**ospitalized (severely ill), **C**ritical (i.e. in ICU, possibly needing a ventilator) and **D**eceased. This can be represented by the finite state machine in Figure (1) or by the set of equations (1).

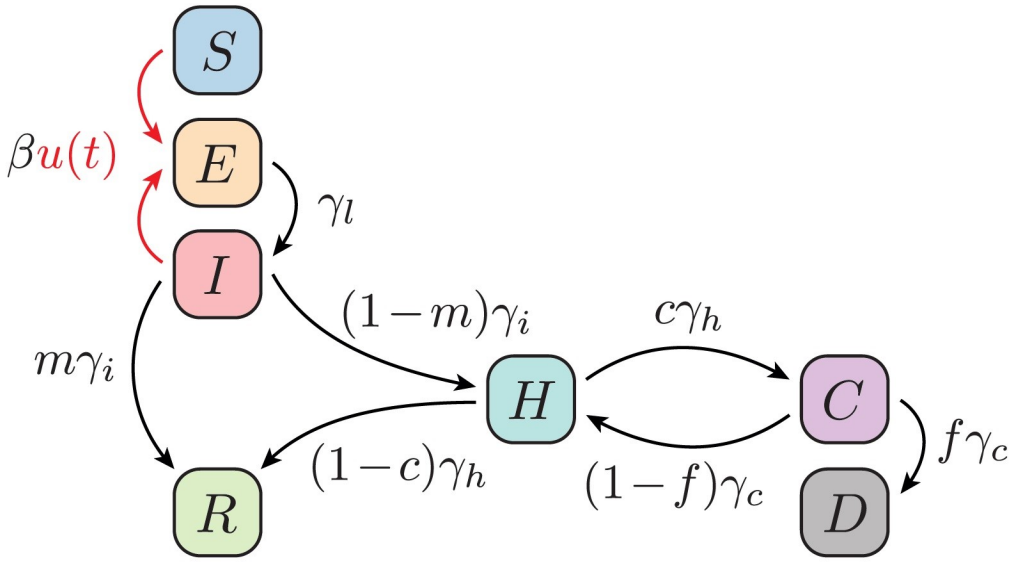


Figure 1: Finite State Machine describing the extended SEIR model. Initially all of the population is in the Susceptible state with a small number of people in the Exposed or Infectious states. After a period of time γ_l a fraction of the Exposed population moves to the Infectious state, represented by the black arrow γ_l. The red arrows from states S and I labeled with $\beta u(t)$ represent the interactions between the Susceptible and Infectious states which can be controlled my the intervention $u(t)$. All the other arrows represent the paths from one state to another, with their labels being the rates associated with each transition. The system of equations representing the finite state machine is written as (1). Image taken from [9]

$$
\begin{aligned}
\dot{S} &= -\beta u(t)\frac{I}{N}S\,,\\
\dot{E} &= \beta u(t)\frac{I}{N}S - \gamma_l E\,,\\
\dot{I} &= \gamma_l E - \gamma_i I\,,\\
\dot{H} &= (1-m)\gamma_i I + (1-f(C/C_o))\gamma_c C - \gamma_h H\,,\\
\dot{C} &= c\gamma_h H - \gamma_c H\,,\\
\dot{R} &= m\gamma_i I + (1-c)\gamma_h H\,,\\
\dot{D} &= f(C/C_o)\gamma_c C\,.
\end{aligned}
\tag{1}
$$

Where the many variables are defined in Table 1 with the initial data for the model for the population of Germany, as was provided in [9]. More succinctly,

$$
\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})\,, \tag{2}
$$

where $\boldsymbol{x} = (S, E, I, H, C, R, D)^T$, the current state, and $\boldsymbol{f}(\boldsymbol{x})$ represents the right hand side (1). The model works in steps that are described by (1), where the dot (derivative in time, e.g. $\dot{x}$) represents the instantaneous change in the sub-populations.

The sub-population of ***S***usceptible people (S) are those that do not have have immunity and are currently not infected, and that may become infected from interactions with the infectious (I) population. This interaction occurs with the probability of β multiplied by the ratio of the infectious population to the total population multiplied by the intervention function u, and then this portion of the population moves to the exposed (E) group i.e. the rate that S changes each day is $\dot{S}$. The rate of change in the ***E***xposed population is described by the expression $\dot{E}$ where latency period γ_l^{-1} is the average period of time required for a (E) exposed person to become infectious themselves after being exposed. The $\dot{I}$ is the change in the ***I***nfectious population where γ_i^{-1} is the average period before recovery or that of an infectious person requiring hospitalization. The remaining daily changes in each population ($\dot{H}$, $\dot{C}$,

symbol	value	description
R_o	2.7	basic reproduction number
N	$83,000,000$	total population size of Germany
γ_l^{-1}	$2.6d$	average latency time between exposure and infectious period
γ_i^{-1}	$2.35d$	average infectious period before recovery or hospitalization
γ_h^{-1}	$4.0d$	average period before severely ill patients turn critical or recover
γ_c^{-1}	$7.5d$	average period before critical patients recover or die
β	$1.15(d)^{-1}$	transmission rate
m	0.92	fraction of infected with at most mild symptoms
c	0.27	fraction of hospitalized patients that turn critical
f	see Eq. (3)	fraction of critical patients that turn fatal
f_o	0.31	mortality of a critical patient with ICU
f_1	$2f_0 = 0.62$	mortality of a critical patient without ICU
C_o	$30,000$	number of ICUs/max. number of simultaneously critical cases.
T	10 years	final time of the simulation

Table 1: List of parameters [9] used in verification of the implementation of model. For simulations specific to the Province of Ontario: $N = 14;745,040$[2] with ICU capacity $C_o = 2,064$ [8] was used with all other data remaining the same.

$\dot{R}$, $\dot{D}$) are dependent on: m the proportion of the population that has at most mild symptoms, the average period γ_h^{-1} that it takes for a person to recover or become critically ill, γ_c^{-1} the average period of time for a patient to either die or stabilize and return to the critical state (H), and on the fraction f of the ICU patients that dies. The fraction f is defined by a piece-wise function (3) that is dependent on the ICU capacity C_o and the current number of ICU beds needed C:

$$f = f\left(\frac{C}{C_o}\right) = \begin{cases} f_o & \text{for } C \leq C_0, \\ f_1 - \frac{C_o}{C}(f_1 - f_o) & \text{for } C \leq C_0\,. \end{cases} \tag{3}$$

In the model, the intervention $u(t)$ may vary from 1, where there is no intervention, to 0, where there is an absolute lockdown and no person has any interactions with the outside world. Note that the basic reproduction number is defined:

$$R_o \coloneqq \frac{\beta}{\gamma_i}\,, \tag{4}$$

and that this matches the data given in Table 1 (using β and γ_i). However, with the intervention $u(t)$ we have the effective reproduction number that is time dependent:

$$R_{eff} = R_o u(t)\,. \tag{5}$$

3 Verification of Model Implementation

Using the 4th order Runge-Kutta method described in Appendix A, a C++ program was written (see Appendix C.2) to implement model (1). To test the program we ran it with the same initial conditions that were proposed in [9], i.e. using initial conditions $E(t = 0) = 20$ of exposed individuals with data in Table 1 and no intervention (i.e. $u(t) = 1$ for all t). We see that Figure 2 matches well Figure 2(a) of [9]. By numerically searching for maximum values for hospitalizations and ICU admissions,

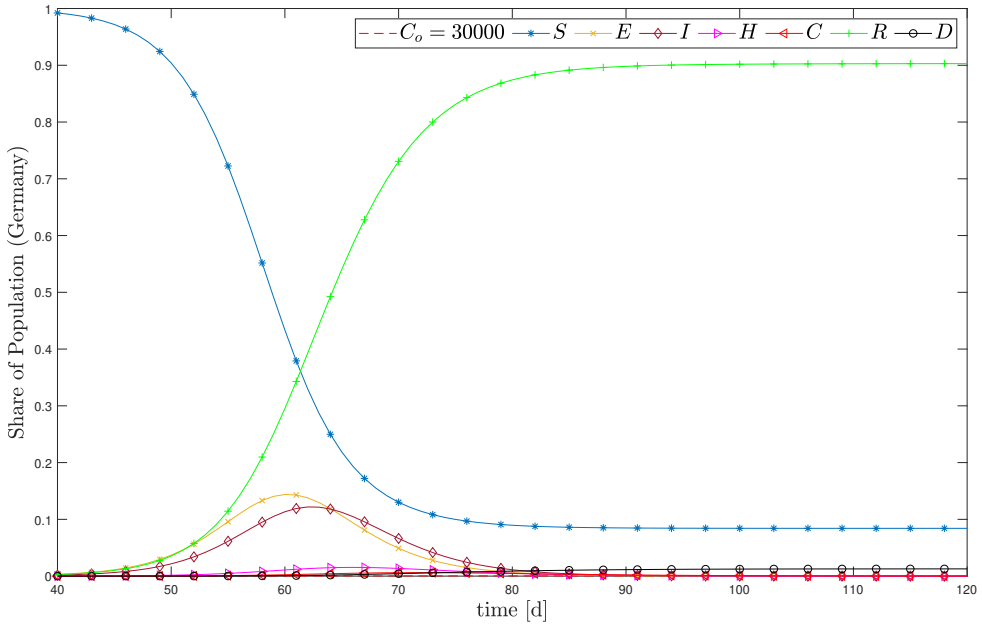


Figure 2: Graphical representation of results of program implementation with the same initial conditions of [9]. We see that it matches well Fig. 2(a) [9].

we obtained:

$$
\begin{aligned}
I_{max} &\approx 10.1 \times 10^6\,, \\
C_{max} &\approx 5.05 \times 10^5\,, \\
\frac{C_{max}}{C_o} &\approx 16.8\,, \\
D(T) &\approx 1.05 \times 10^6\,.
\end{aligned}
$$

Noting that C_{max} & hence C_{max}/C_o, and $D(T)$ (total deaths at the end of the simulation) are slightly different than in [9]. This could be caused by: not using the regularization proposed in the paper.

The number of deaths at the end of the epidemic is very high, and is attributed to the ICU being over capacity for approximately 58 days (see figure 3) leading to the higher fatality rate. From Figure (3), we can see the steep increase in the number of deaths from when the ICU hits capacity until the deaths start to plateau when the

ICU is within its' capacity limits again.

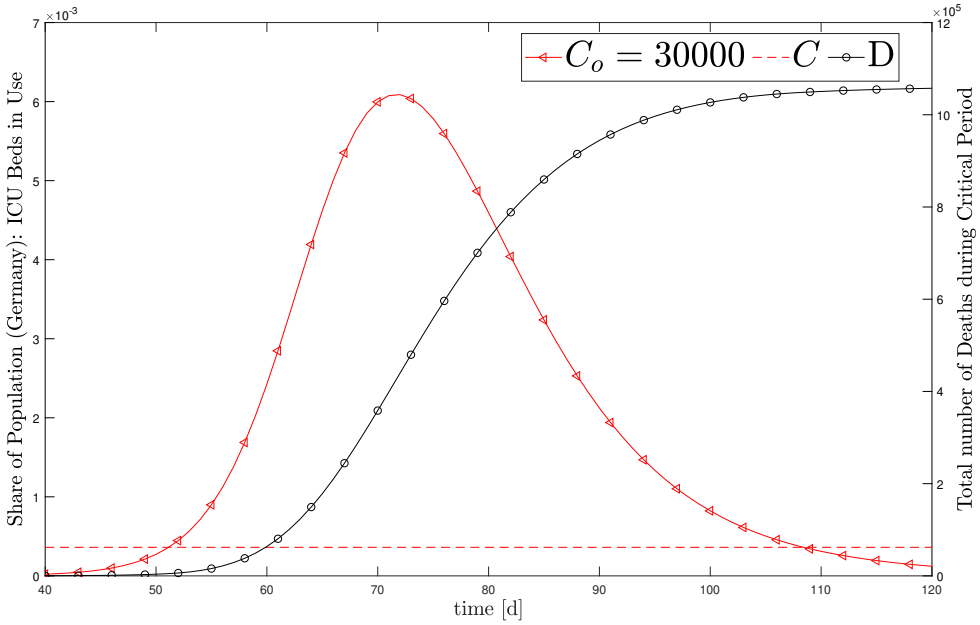


Figure 3: The curve on the left (red) represents the demand on the ICUs, the dotted line is the ICU capacity. This corresponds to the inserted figure within Fig. 2(a) from [9]. The curve on the right (black) is the cumulative deaths during the critical time of the ICU being over capacity. One can see that the deaths grow quickly once the ICU passes capacity and plateau's once the ICU is below capacity, again.

In Section 5 the model will be applied to the Ontario population. We varied the intervention $u(t)$ in (1) and introduced different transmission rates to represent new variants.

4 Analysis of System (1)

We study the stability of the system (1) where the system represents a population in a disease free state ant that the population would be free from interventions that limit the transmission of the disease. This disease free state is an ***equilibrium point*** $\bar{\boldsymbol{x}}$ such that $\boldsymbol{f}(\bar{\boldsymbol{x}}) = \mathbf{0}$, and we study its stability should a small (*spontaneous*) number of exposed or infections persons be introduced into the population (say from travel), i.e. would the population return to a stable disease free state or would exponential growth occur? Under what conditions is this stability maintained? The following is the analysis required to provide answers to these questions.

By inspection one can easily verify that the following meets the definition of an ***equilibrium point***:

$$\bar{x} = \begin{pmatrix} S \\ E \\ I \\ H \\ C \\ R \\ D \end{pmatrix} = \begin{pmatrix} \bar{S} \\ 0 \\ 0 \\ 0 \\ 0 \\ \bar{R} \\ \bar{D} \end{pmatrix},$$

where $\bar{\boldsymbol{x}}$ is an element of the set of stationary states (or equilibrium points). What we seek now is the additional criteria on the set of equilibrium points, required such that with small perturbations $\bar{\boldsymbol{x}} \rightarrow \bar{\boldsymbol{x}} + \epsilon \boldsymbol{x}(\boldsymbol{t})$, that represent *spontaneous* COVID cases, we return to a disease free state. To accomplish this, we take the *Taylor* expansion of (1) about $\bar{\boldsymbol{x}}$, and keep the first non-zero term, which is $Df(\bar{\boldsymbol{x}})$. This is the linearization of $\boldsymbol{f}(\boldsymbol{x})$ about the equilibrium point $\bar{\boldsymbol{x}}$ when the expansion is evaluated at the equilibrium point. We now take the following steps to search for non-negative eigenvalues to insure stability of the stationary states (for further discussion of background material see Appendix B).

Step one is to calculate the first term $A(\bar{\boldsymbol{x}})$ of the the Taylor expansion of (20):

$$\boldsymbol{A}(\bar{\boldsymbol{x}}) = \begin{pmatrix} 0 & 0 & -\beta u(t)\frac{\bar{S}}{N} & 0 & 0 & 0 & 0 \\ 0 & -\gamma_l & \beta u(t)\frac{\bar{S}}{N} & 0 & 0 & 0 & 0 \\ 0 & \gamma_l & -\gamma_i & 0 & 0 & 0 & 0 \\ 0 & 0 & (1-m)\gamma_i & -\gamma_h & (1-f(C/C_o))\gamma_c & 0 & 0 \\ 0 & 0 & 0 & c\gamma_h & -\gamma c & 0 & 0 \\ 0 & 0 & m\gamma_i & (1-c)\gamma_h & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & f(C/C_o) & 0 & 0 \end{pmatrix}.$$

Now we must search for the eigenvalues of $\boldsymbol{A}(\bar{\boldsymbol{x}})$ using the technique in Appendix B.1 as follows:

Let,

$$\tilde{\boldsymbol{A}} = \boldsymbol{A}(\bar{\boldsymbol{x}}) - \lambda \boldsymbol{Id}\,,$$

that is

$$\tilde{\boldsymbol{A}} = \begin{pmatrix} -\lambda & 0 & -\beta u(t)\frac{\bar{S}}{N} & 0 & 0 & 0 & 0 \\ 0 & -\lambda-\gamma_l & \beta u(t)\frac{\bar{S}}{N} & 0 & 0 & 0 & 0 \\ 0 & \gamma_l & -\lambda-\gamma_i & 0 & 0 & 0 & 0 \\ 0 & 0 & (1-m)\gamma_i & -\lambda-\gamma_h & (1-f(C/C_o))\gamma_c & 0 & 0 \\ 0 & 0 & 0 & c\gamma_h & -\lambda-\gamma c & 0 & 0 \\ 0 & 0 & m\gamma_i & (1-c)\gamma_h & 0 & -\lambda & 0 \\ 0 & 0 & 0 & 0 & f(C/C_o) & 0 & -\lambda \end{pmatrix}.$$

Then blocking matrix $\tilde{\boldsymbol{A}}$ as follows,

$$\tilde{\boldsymbol{A}} = \left(\begin{array}{c|c} \mathrm{V} & \mathrm{U} \\ \hline \mathrm{W} & \mathrm{Z} \end{array}\right) = \left(\begin{array}{ccc|ccc|c} -\lambda & 0 & -\beta u(t)\frac{\bar{S}}{N} & 0 & 0 & 0 & 0 \\ 0 & -\lambda-\gamma_l & \beta u(t)\frac{\bar{S}}{N} & 0 & 0 & 0 & 0 \\ 0 & \gamma_l & -\lambda-\gamma_i & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & (1-m)\gamma_i & -\lambda-\gamma_h & (1-f(C/C_o))\gamma_c & 0 & 0 \\ 0 & 0 & 0 & c\gamma_h & -\lambda-\gamma c & 0 & 0 \\ 0 & 0 & m\gamma_i & (1-c)\gamma_h & 0 & -\lambda & 0 \\ 0 & 0 & 0 & 0 & f(C/C_o) & 0 & -\lambda \end{array}\right).$$

And since V, Z are square and U is a zero matrix we can utilize Appendix B.1 and attain,

$$\det(V) = -\lambda\left(\lambda^2 + (\gamma_l + \gamma_i)\lambda + \gamma_l\left(\gamma_i - \beta\frac{\bar{S}}{N}\right)\right)$$

$$\det(Z) = -\lambda^2(\lambda^2 + (\gamma_h + \gamma_c)\lambda + \gamma_h\gamma_c[1 - (1 - f_o)c])$$

which gives the characteristic equation:

$$0 = \det(\tilde{\boldsymbol{A}})\,,$$

$$0 = -\lambda^3\left(\lambda^2 + (\gamma_l + \gamma_i)\lambda + \gamma_l\left(\gamma_i - \beta\frac{\bar{S}}{N}\right)\right)(\lambda^2 + (\gamma_h + \gamma_c) + \gamma_h\gamma_c[1 - (1 - f_o)c])\,.$$

And we determine the eigenvalues (λ's):

$$\lambda_1 = 0, \text{ with multiplicity } 3\,, \tag{6}$$

$$\lambda_{2\pm} = \frac{1}{2}\left[-(\gamma_l + \gamma_i) \pm \sqrt{(\gamma_l - \gamma_i)^2 - 4\gamma_l\beta\frac{u(t)\bar{S}}{N}}\right], \tag{7}$$

$$\lambda_{3\pm} = \frac{1}{2}\left[-(\gamma_h + \gamma_c) \pm \sqrt{(\gamma_h - \gamma_c)^2 - 4\gamma_h\gamma_c(1 - f_o]c}\right]. \tag{8}$$

To ensure stability, we search for the conditions under which the eigenvalues are less or equal to 0. Clearly $\lambda_1 = 0$, and both λ_{2-} and λ_{3-} are negative. So we search for the criteria for λ_{2+} and λ_{3+} being negative. Since $[1-f_o]c < 1$, $\lambda_{3+} < 0$. Substituting (4) into (7) we deduce the requirements for λ_{2+} to be negative, to be:

$$\frac{1}{R_o} > \frac{u(t)\bar{S}}{N}\,.$$

It is desired that $u(t) = 1$, i.e. no intervention, at the final disease free state and we

deduce the maximum $\bar{S}$ for any population.

$$\frac{1}{R_o} > \frac{\bar{S}}{N}\,, \tag{9}$$

$$1 > R_o \frac{\bar{S}}{N}\,, \tag{10}$$

$$1 > \frac{\gamma_i}{\beta}\frac{\bar{S}}{N}\,. \tag{11}$$

Provided Conditions (9)-(11) are maintained for the population with up to date disease parameters β and γ_i, should a small number of spontaneous infections occur, the system will return to a stable disease free state.

It must be noted that the eigenvalues may have both real and imaginary parts especially when γ_l, γ_i are close in value and γ_h, γ_c are close in value. However, as long as the conditions stated above hold, the real parts of the eigenvalues are always negative implying the stability about the equilibrium point.

5 Covid in Ontario

The purpose of any model is to provide predictive information. In this case models have been used to inform various governing bodies on what restrictions are potentially necessary to control or reduce the spread of Covid-19 to a manageable level and which does not overwhelm the healthcare system. The parameters of the simulation have been adjusted to correspond to the Ontario population; also, we have modified the program (see Code C.2) to adjust the intervention $u(t)$ that simulates restrictions that have been enacted in the Province of Ontario; which have affected how the virus has spread, qualitatively, in Ontario. The floor of $u(t)$ and the rules to adjust it up or down have been separated into four time periods:

- The beginning of March 2020, which is when the first wave began up until the beginning of August. However, the population was much more disciplined so the floor $u(t) = 0.30$ was used.

- Starting in Mid August the public discipline started to wane and schools opened up to in-person learning. This combined with the cooling temperatures, and people spending more time in enclosed spaces, the floor $u(t) = 0.40$ was used.

- From mid-March until the stay at home order was put in place on April 9th, there was much more public gatherings (for example large numbers of children and parents gathering at parks) due to the 'Covid-fatigue', and from the schools closing for the delayed March break. So, the floor for $u(t)$ was set much higher, to 0.48

- The floor was set to $u(t) = 0.30$ starting with the Stay-At-Home order April 9th, 2020 and remained in place for the remainder of the simulation.

First, we will examine the results of the simulation with no variants introduced into the population; refer to Fig (4).

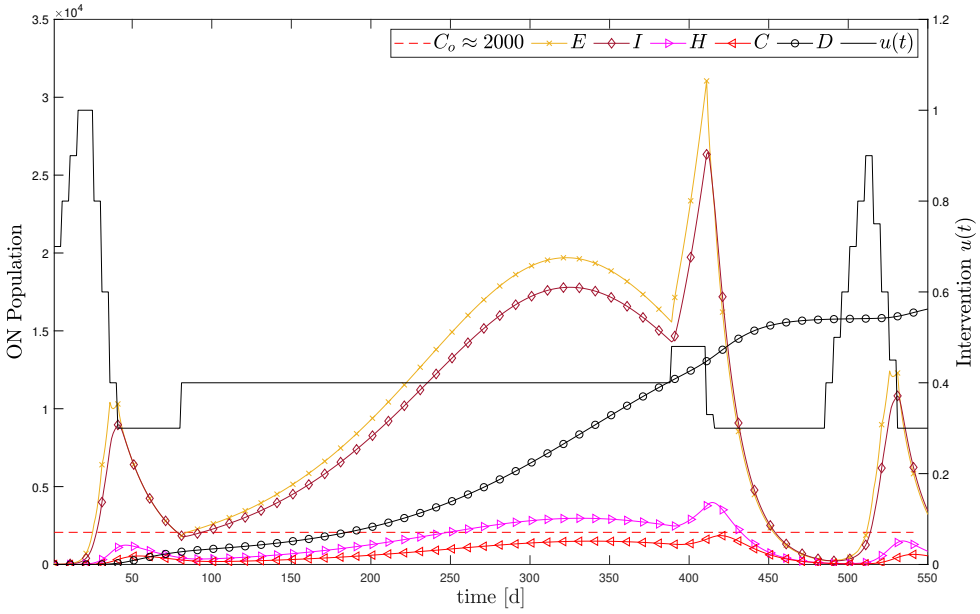


Figure 4: This is the output of the simulation with no variants in Province of Ontario which qualitatively matches the waves that occurred in the Province up to approximately day 440. The left y-axis is the Ontario population, the x-axis is the days since the simulation started, and on the right y-axis is the intervention, on the figure it is the black piece-wise line. The dotted line is the ICU capacity in the province, which it is desirable to stay below. Other quantities are in the legend.

By examining Figure (4), it can been seen that the three waves are simulated and approximately centered about: 90days (March-April 2020), 300 days (December - early February), and 400 days (April - May 2021) respectively. Notice that during the brief relaxation the intervention in March caused a spike in the infections in the population. Examining the simulation past day 440, we could expect a fourth wave if restrictions are relaxed too.

To have more perspective on the stress put on the healthcare system see Figure (5). We can see that Ontario's healthcare system would be very stressed, and this matches qualitatively with what has been happening during the third wave.

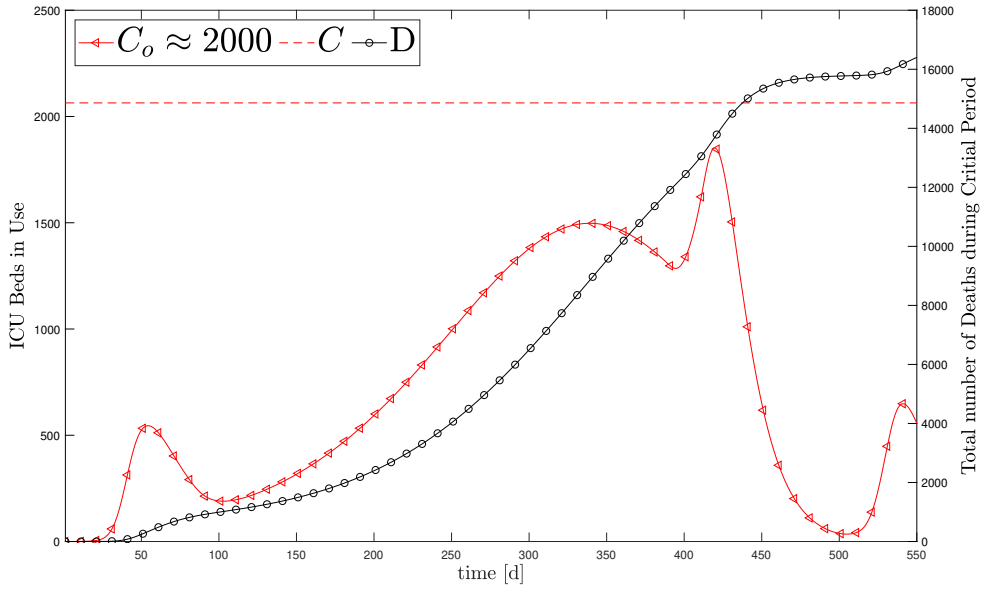
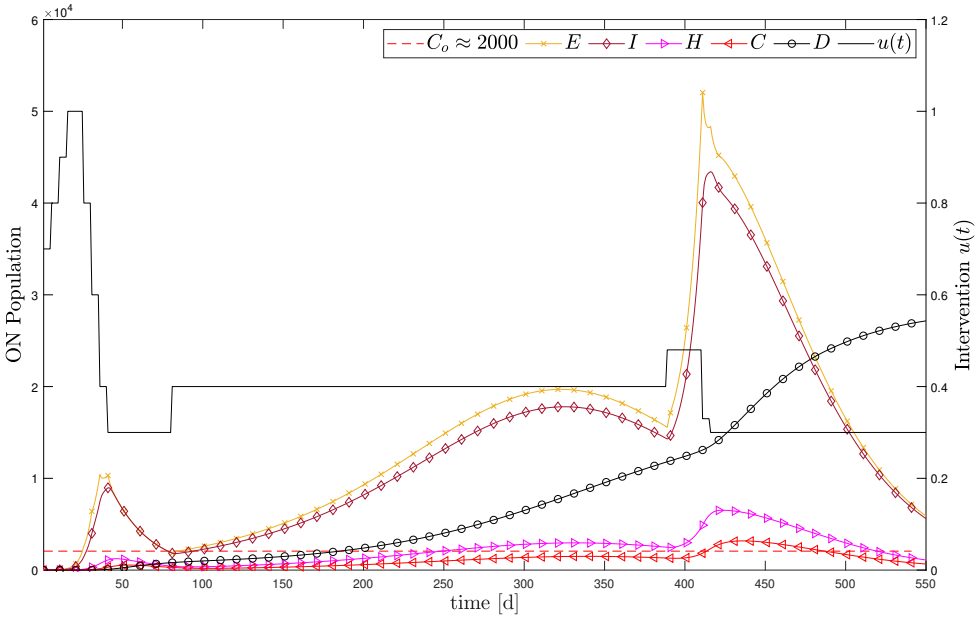


Figure 5: A closure look at the stress placed on the healthcare system, specifically on the ICU capacity, modelled from the same simulation as in figure (4). Again the dotted line is ICU capacity, the red line is the demand on the ICU's and the black line is the total number of deaths.

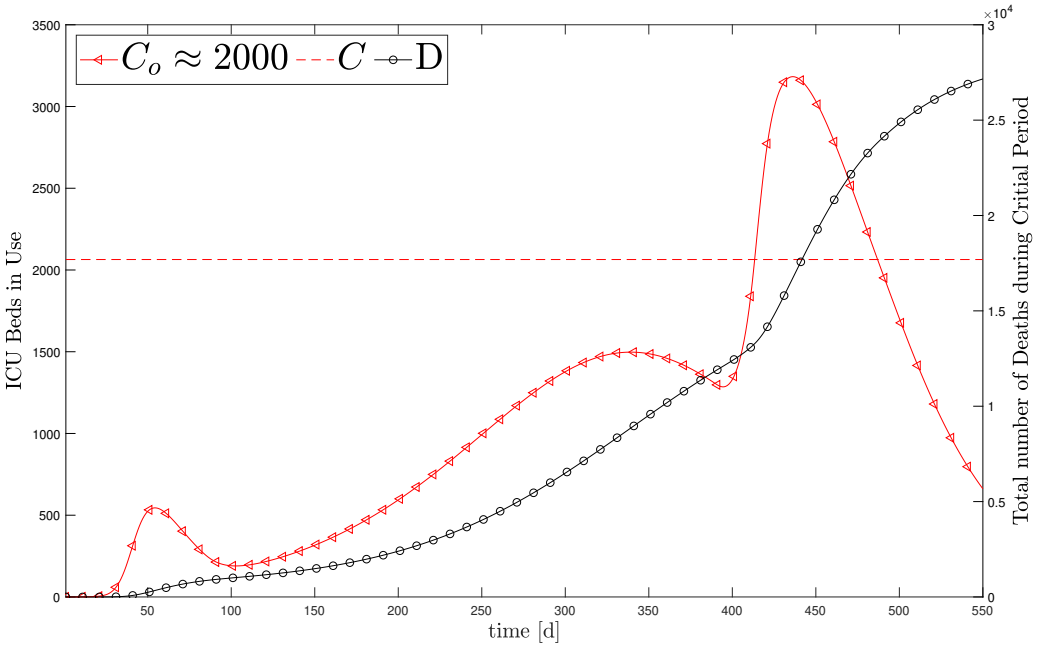
Now we will examine what the model predicts, holding the floor for $u(t) = 0.30$, if a variant is introduced into the population starting on the 390^{th} day and, over a 30 day period, it gradually becomes dominant strain. The variant(s) are known to have transmission rates $40\% - 90\%$ higher than the original COVID strain, under consideration. We ran the simulation under three different scenarios:

- Low - transmission increase of 40% see Figures (6a and 6b),
- Median - transmission increase of 65% see Figures (7a and 7b),
- High - transmission increase of 90% see Figures (8a and 8b).

Examining Fig. (6b), it is evident that even in the low-transmission rate variant case, the Ontario health care system would be overwhelmed. In this simulation the maximum number of infectious people would be $43,425$ with the ICU being at 154% capacity and the total deaths by day (550) would be approximately $3,200$. So, it is



(a) Plot of the simulation of the low transmission increase variant (40%) with overlay of the intervention $u(t)$, and all sub-populations excluding the susceptible and recovered.



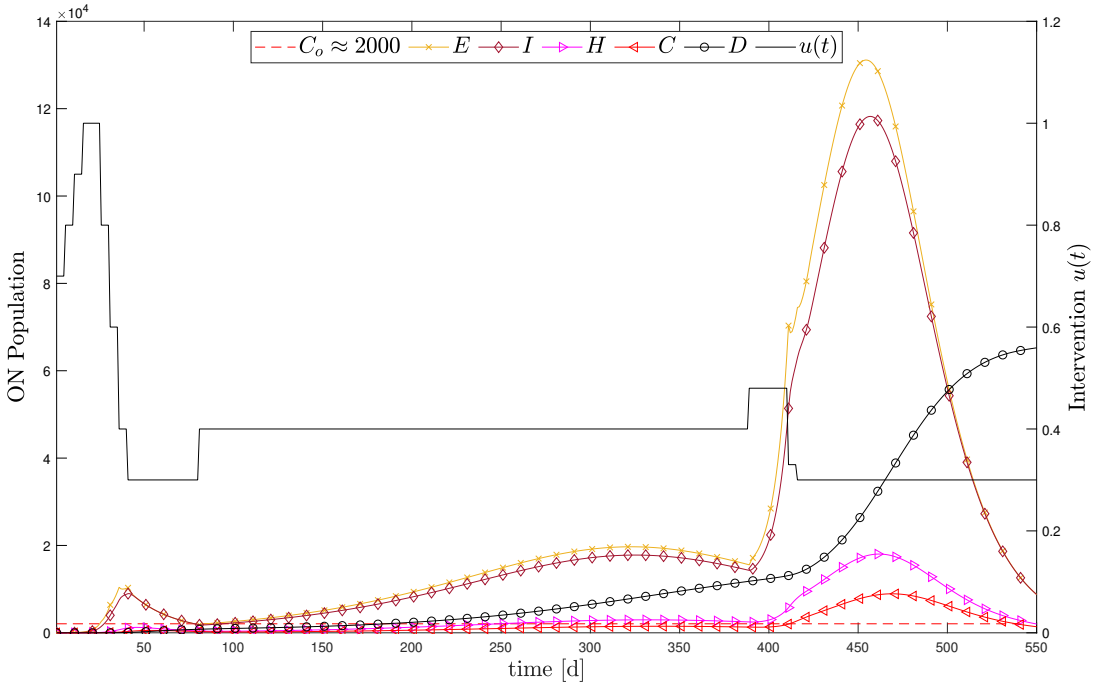
(b) The demand placed on the ICU and total deaths, for the low transmission increase variant (40%)

Figure 6: The Low-transmission variant simulation

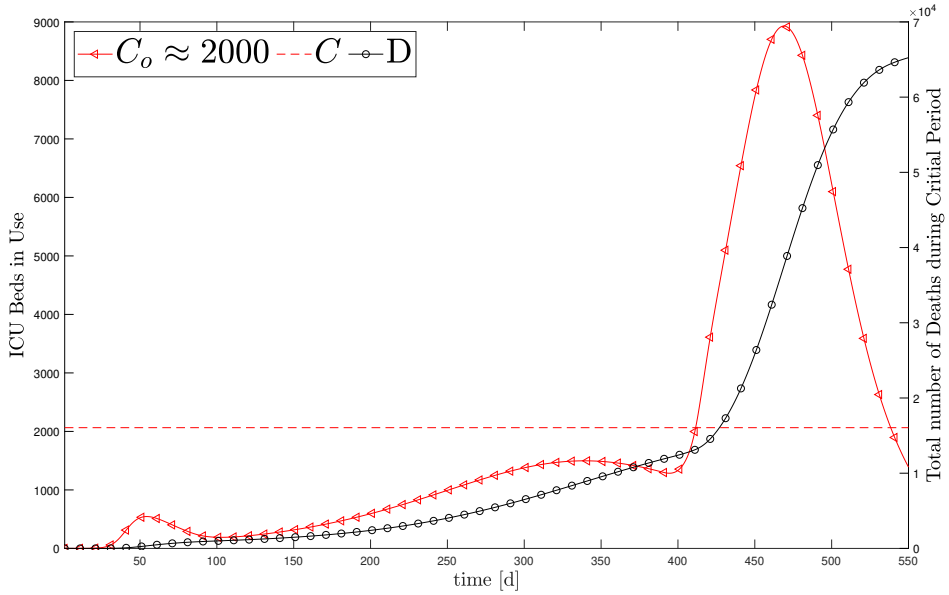
clear that, should this be avoided, the floor must be lowered such that $u(t) < 0.3$. It was not explored how low $u(t)$ must be choosen to avoid the ICUs being at or over capacity.

From Figures, 7a & 7b, and 8a & 8b, it is clear that health care system would be completely overwhelmed. In the median transmission case, there would be $65,245$ deaths and at the peak the ICU capacity would be at 432%. Similarly, for the High transmission case, there would be $70,958$ deaths and at the peak the ICU capacity would be at 498%.

Any of the above scenarios with the variants must be avoided; meaning, from the combination of government interventions and personal choices, the intervention $u(t)$ must stay low enough to insure that the healthcare system does not be over run.

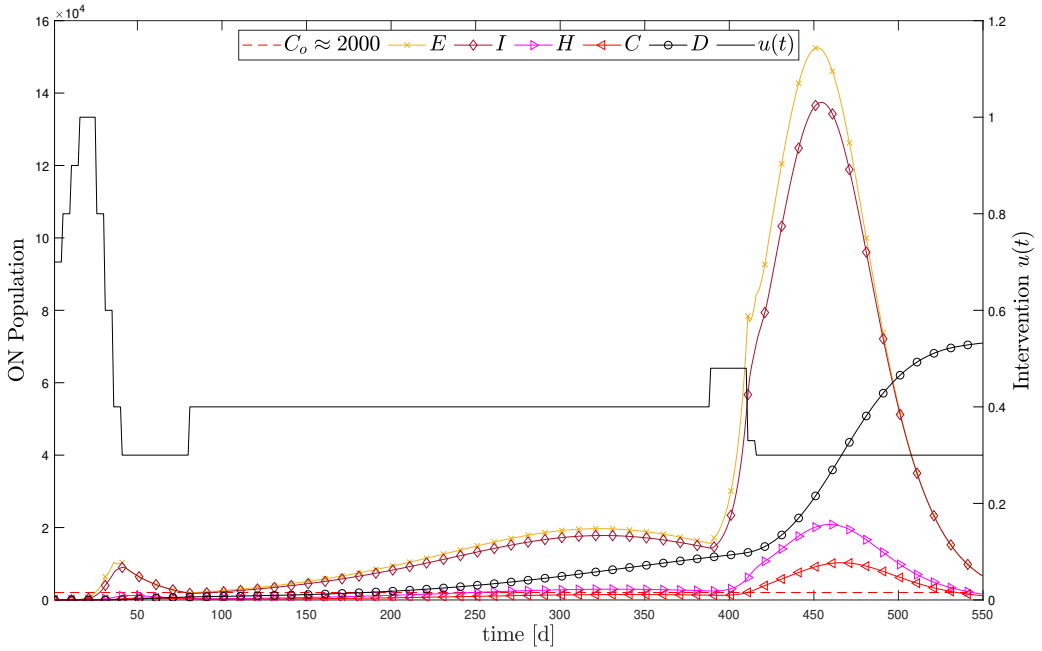


(a) Plot of the simulation of the low transmission increase variant (65%) with overlay of the intervention $u(t)$, and all sub-populations excluding the susceptible and recovered.

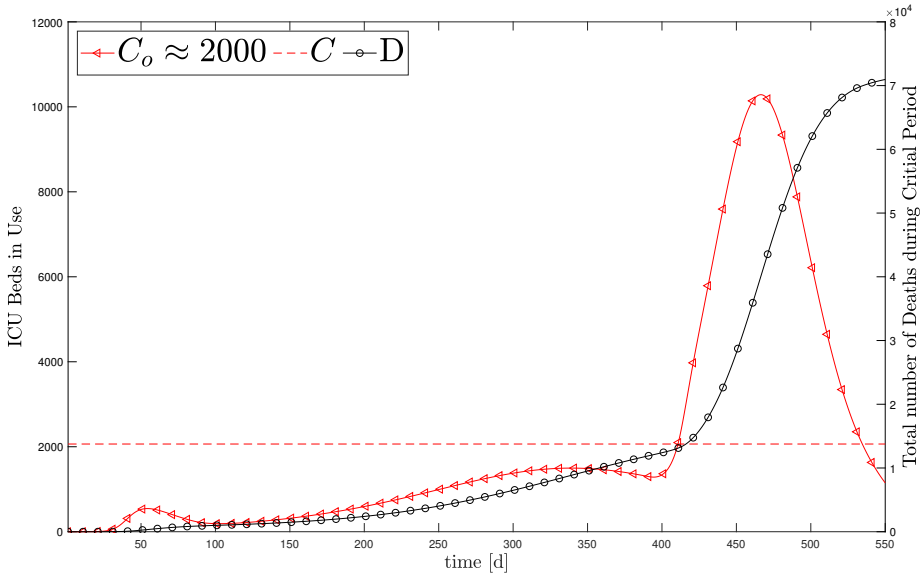


(b) The demand placed on the ICU and total deaths, for the low transmission increase variant (65%)

Figure 7: The Median-transmission variant simulation



(a) Plot of the simulation of the low transmission increase variant (90%) with overlay of the intervention $u(t)$, and all sub-populations excluding the susceptible and recovered.



(b) The demand placed on the ICU and total deaths, for the low transmission increase variant (90%)

Figure 8: The High-transmission variant simulation

6 Stability of a Disease Free State

Our model is working under this (reasonable) assumption that once a person has recovered they have immunity; however, only 95% of the recovered population have sufficient antibodies against COVID-19 eight months after recovery [3]. So, it is necessary to strengthen the stability of the disease free state. We introduce the parameter $\delta = 0.95$ into equation (11), which is the proportion of the recovered population $\bar{R}$ that is still immune. Ignoring deaths $\bar{D}$ we attain $\bar{R} = N - \bar{S}$, which gives:

$$1 > \frac{\gamma_i}{\beta} \frac{(1.05\bar{S} - 0.05N)}{N} \,. \tag{12}$$

For illustrative purposes we will look at the city of Ottawa to deduce the upper bound of the susceptible population $\bar{S}$ to maintain a disease free state. Noting that the population of Ottawa is approximately $N = 1,018,000$ [4] the susceptible population $\bar{S}$ must be less than $407,559$. Further, as of April 20th, 2021 the City of Ottawa was administering doses at the optimistic (seven day average) rate of $8,329$doses/day [5]. Focusing on the effectiveness rate of a single dose, being $60\% - 80\%$ [10], and given $18,381$ recovered people in the city; one can preform the following calculation, ignoring the time it takes to develop immunity, to attain the number of days until a stable disease free state could exist.

$$t \geq \frac{(1,018,001 - 18,381) - 407559}{8,329/\text{day} \ \times \ 0.6} = 191.9 \,\text{days} \tag{13}$$

So, if we were currently in a disease free state, it would take at least 192 days to lower the susceptible population (S) such that the city would be in a stable disease free state.

Doing the similar analysis for the Province of Ontario as of May 15th, 2021 with data from [11] and [6], we have the following calculation:

$$t \geq \frac{(14,745,040 - 8,655 - 6,638,361)}{120,000/\text{day} \ \times \ 0.6} = 112.6 \,\text{days} \,. \tag{14}$$

7 Conclusion

In our implementation of the Covid model presented in [9], the intervention $u(t)$ was made to vary in response to Covid case loads in the various sub-populations, this variation in $u(t)$ produced multiple waves. We can deduce that the interventions put in place, followed by the relaxation of these interventions are responsible for multiple waves seen in many jurisdictions around the world and specifically in the Province of Ontario.

We attained the criteria required for any potential equilibrium points (i.e. disease free states) to be stable. This criteria is shown by the inequality in equation (11) where the ratio of susceptible population to the total population must be small enough for the inequality to hold. Taking into consideration that there is a large vaccination campaign focusing primarily on delivering the first dose, we determined that for the City of Ottawa and for the Province of Ontario, it would take 192 days and 113 days, see Equations (14) and (5), respectively, to achieve a potential stable equilibrium point.

For this pandemic to conclude, it is necessary for governments continue to implement interventions and that the public remains willing to follow the interventions. If the population remains vigilant until a sufficient portion of the population is vaccinated, we could see a return to normality in the not too distant future.

A Runge-Kutta (RK) Methods

A large portion of this section is taken from [13]. The general form of Runge-Kutta methods is:

$$y_{n+1} = y_n + hF(t_n, y_n, h; f),\ n \geq 1$$

Where F is defined by:

$$\begin{aligned} F(t_n, y_n, h; f) &= \sum_{i=1}^{s} b_i K_i\,, \\ K_i &= f(t_n + c_i h, u_n + h \sum s_{j=1} a_{ij} K_j). \end{aligned}$$

$i = 1, 2, \ldots, \mathrm{s}$
where s is the number of stages of the method under consideration.

We have the following Butcher tables for explicit Runge-Kutta methods, the general on the left and the one that will be used on the right.

$\mathrm{c}_1 = 0$				
c_2	$\mathrm{a}_{2\,1}$			
c_3	$\mathrm{a}_{3\,1}$	$\mathrm{a}_{3\,2}$		
$\vdots$	$\vdots$	$\vdots$	$\ddots$	
c_s	$\mathrm{c}_{s\,1}$	$\mathrm{c}_{s\,2}$	$\ldots$	$\mathrm{c}_{s\,s}$
	b_1	b_2	$\ldots$	b_s

with, $\sum_{i=1}^{s} b_i = 1$

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	2/6	2/6	1/6

$$\frac{1}{6} + \frac{2}{6} + \frac{2}{6} + \frac{1}{6} = 1$$

The Butcher table, above on the the right, corresponds to the following

$$y_{n+1} = y_n + \frac{h}{2}(K_1 + 2K_2 + 2K_3 + K_4) \tag{15}$$

$$K_1 = f(y_n) \tag{16}$$

$$K_2 = f(y_n + \frac{h}{2}K_1) \tag{17}$$

$$K_3 = f(y_n + \frac{h}{2}K_2) \tag{18}$$

$$K_4 = f(y_n + hK_3) \tag{19}$$

This is the RK method that will be used to implemented the covid model.

The RK method is *consistent* $\iff \sum_{i=1}^{s} b_i = 1$ and with RK methods being single step we have (*adapted from* [13]) we have

$$consistency \bigoplus stability \implies convergence$$

Also, since the system (20) is *autonomous*, so we can extend the RK method discussed above to model the system (20) and maintain order $p = 4$.

B Background Theory [12]

To analysis that dynamical system proposed in [9], which models COVID interventions and spread in Germany, some basic background is needed.

We consider the system:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})\,, \text{ where } \boldsymbol{f} \in C(E) \text{ and } \boldsymbol{x} \in E \subseteq \boldsymbol{R^n} \tag{20}$$

First we have some definitions that are necessary to perform the analysis on our system (20), and the four step *Runge-Kutta* method with order $p = 4$.

Definition B.1. Suppose that $\boldsymbol{f} \in C(E)$ where $E \subseteq \boldsymbol{R^n}$ with E open. Function

symbolx is a ***solution of the differential equation*** (20) on an interval $\boldsymbol{I}$, if $\boldsymbol{x}$ is differentiable on $\boldsymbol{I}$ and if $\forall t \in \boldsymbol{I}$ and satisfies

$$\boldsymbol{x}'(t) = \boldsymbol{f}(\boldsymbol{x}(\boldsymbol{t})),\ t \in \boldsymbol{I}\,,$$
$$\boldsymbol{x}(t_o) = \boldsymbol{x_0}$$

on an interval $\boldsymbol{I}$ if $t_o \in I, \boldsymbol{x}(t_o) = \boldsymbol{x}_o$ and $\boldsymbol{x}(t)$ is a solution of the differential equation (20) on the interval $\boldsymbol{I}$

Definition B.2. Let $E \subseteq \boldsymbol{R^n}$, with E open. A function$\boldsymbol{f} : E \to \boldsymbol{R}^n$ is said to satisfy a ***Lipschitz condition*** on E if there is a positive constant $\boldsymbol{L}$ such that $\forall \boldsymbol{x}, \bar{\boldsymbol{x}} \in E$

$$|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\bar{\boldsymbol{x}})| \leq \boldsymbol{L}|\boldsymbol{x} - \bar{\boldsymbol{x}}|.$$

Then function $\boldsymbol{f}$ is said to be ***locally Lipschitz*** on E if for each point $\bar{\boldsymbol{x}} \in E$ there is an ε-neighbourhood of $\bar{\boldsymbol{x}}$, $N_\varepsilon(\bar{\boldsymbol{x}}) \subseteq E$ and a constant $L_o > 0$ such that $\forall \boldsymbol{x}, \bar{\boldsymbol{x}} \in N_\varepsilon(\boldsymbol{x}_o)$

$$|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\bar{\boldsymbol{x}})| \leq \boldsymbol{K}_o|\boldsymbol{x} - \bar{\boldsymbol{x}}|,$$

with $N_\varepsilon(\boldsymbol{x}_o) = \{\boldsymbol{x} \in \boldsymbol{R}^n |\ |\boldsymbol{x} - \bar{\boldsymbol{x}}| < 0\}$

Lemma B.3. *Let $E \subseteq \boldsymbol{R^n}$, with E open and let $\boldsymbol{f} : E \to \boldsymbol{R}^n$. Then, if $\boldsymbol{f} \in C^1(E)$, $\boldsymbol{f}$ is locally Lipschitz on E.*

Definition B.4. A point $\boldsymbol{x_o} \in \boldsymbol{R}^n$ is called an ***equilibrium point*** or a *critical point* of (20) if $\boldsymbol{f(x_o) = 0}$. An equilibrium point $\boldsymbol{x_o}$ is called a ***hyperbolic equilibrium point*** of (20) if none of the eigenvalues of the matrix $D\boldsymbol{f}(\boldsymbol{x_0})$ have zero real part. The linear system (**??**) with the matrix $A = D\boldsymbol{f}(\boldsymbol{x_0})$ is called the ***the linearization*** of (20) at $\boldsymbol{x_o}$.[12]

To justify the linearization proposed above as being a *good* approximization of (20) we take the Taylor expansion, by Taylor's Theorem ([12]) below, about the equilibrium point $\boldsymbol{x}_o$,

$$\boldsymbol{f}(\boldsymbol{x}) = D\boldsymbol{f}(\boldsymbol{x_o})\boldsymbol{x} + \frac{1}{2}(D^2\boldsymbol{f}(\boldsymbol{x_o})\boldsymbol{x}, \boldsymbol{x}) + \ldots$$

with $(\cdot, \cdot)$ being the standard inner product.

Definition B.5. An equilibrium point $\boldsymbol{x_o}$ of (20) is called a ***sink*** if all the eigenvalues of the matrix $D\boldsymbol{f}(\boldsymbol{x_0})$ have negative real part: it is called a ***source*** if all of the eigenvalues of $D\boldsymbol{f}(\boldsymbol{x_0})$ have positive real part: and it is called a ***saddle*** if it is a hyperbolic equilibrium point and $D\boldsymbol{f}(\boldsymbol{x_0})$ has at least one eigenvalue with a positive real part and at least one with a negative real part.[12]

Theorem B.6. *If $\boldsymbol{x_o}$ is a stable equilibrium point of (20), no eigenvalue of $D\boldsymbol{f}(\boldsymbol{x_0})$ has positive real part. [12]*

Within the text of [12] there is a discussion stating that when $D\boldsymbol{f}(\boldsymbol{x_0})$ has a zero eigenvalue, that $\boldsymbol{x_0}$ is still a stable equilibrium point; however, not asymptotically stable.

Definition B.7. A differential equation for which the right hand side does not contain t explicitly is said to be ***autonomous***. [14]

B.1 Matrix Algebra

For any matrix $\tilde{A} \in M_n R$ we search for the eigenvalues by solving the characteristic equation:

$$0 = det(\tilde{A} - \lambda I) = det(A)$$

Now for any square matrix with $n \geq 4$, solving $det(A)$ becomes quit arduous; however, we can untilize the properties of determinants to limit the effort required. The property of interest was presented in problem 4.6, page 34 of [7]. It follows such:

If we partition A into blocks such that:

$$A = \left[\begin{array}{c|c} V & U \\ \hline W & Z \end{array}\right]$$

We may write:

$$0 = det(A) = det(A^T) = det\left(\begin{array}{c|c} \mathrm{V}^T & \mathrm{W}^T \\ \hline \mathrm{U}^T & \mathrm{Z}^T \end{array}\right)$$

If $U = 0$, and blocks V and Z are square we have:

$$det(A) = det(A^T) = det(V^T)det(Z^T) = det(V)det(Z)$$

C Code

C.1 Learning Code

Lets discuss some concepts of C++ that was needed to implement the code [**code**]

To accomplish this, a knowledge about *objects* was necessary. An object is a instance of a class where you set different parameters of the object. The classic example is that of class *car*. Inside the class *car* there are many aspects (or parameters) of car that need to be *set*, i.e. colour, horsepower, max speed, number of doors.... these would be considered *public* and the user/main program can access and modify them. Other aspects of the class car are *private*, for example: number of tires = 4, it has a engine, it meets the minimum safety standards. And finally there are functions (often called methods) associated with the class, these are things that the car can do i.e. accelerate, brake, turn...

This is a simple example of making a *class*:

```
class car{
    public:
        //Default constructor
        car (){
            // variables/parameters to the default settion
            max_speed = Sunday driver; // just for fun
            manual = 0               // most people do not have manual
                transmissions
            }
        // Parameterized constructor
        car(double max_speed, int manual, int transmission...)
        // setting parameters to specified values
            this->max_speed = max_speed;
```

```
            ....
        // destructor
        ~car() {}

        void set_max_speed(int i){
            int max_speed = i;       // cruse control speed
        }
    private:
        // this is where all the variables should be declared that make
            the class up
        int num_Tires = 4;
        int manual = 0;
        int max_speed;
        int speed;
        ...
};

int main (int argc, char * argv{}){
    // making a default object car using pointers
    car* Car_1 = new car();
    // now I would like to change the parameter max_speed; and since I
        have a pointer
    car->max_speed(100)
}
```

C.2 C++ Implementation of Covid model

This is my code from my implementation of the Covid model using the Runge-Kutta 4. It was to adjust the intervention $u(t)$ one would have to adjust the code before running it, this is what I did, slowly complexifiying my selection of $u(t)$ for different times. Similarly, I adjusted the β to reflect the various variants of the covid virus.

```
Text enclosed inside \texttt{verbatim} environment
```

```
is printed directly
and all \LaTeX{} commands are ignored.

//>> fclose(fileID)
//
//ans =
//
//    0
//
//>> fileID=fopen('Test_.txt', 'r');
//>> C = textscan(fileID, '%f%f%f%f%f%f%f%f', 'Delimiter','[;', '
   HeaderLines',2);
//>> D=cell2mat(C)

// T = D(:,1); S = D(:, 2); E=D(:,3); I=D(:,4); H=D(:,5); C=D(:,6); R=D
   (:,7); DD=D(:,8);


#ifndef __CINT__ // for ROOT this would be to run interactively on a LINUX
    maching, doesn't work on windows


#define _USE_MATH_DEFINES

#include <iostream>
//#include <mpi.h>
#include <math.h>
#include <iterator>
#include <vector>
#include <istream>
#include <fstream>

#endif    // for ROOT this would be to run interactively on a LINUX
```

```
   maching, doesn't work on windows
/*     this is my function which only needs to be a mapping
       F: R^5 -> R^7 ; however, I will make it R^7 -> R^7, so that it is
           easier
       to conceptually understand later.
*/
//std::vector<long double> F(std::vector<long double> &);


/*     defining global variables that will be used in the
       function above.
*/
// all set in days

class popAgeGroup {

       public:
              // Default constructor
              popAgeGroup(int T) {

                     /*(int age_low, int age_high, double frac, int N(=
                         Grand_pop), int priority, long double numE, long
                         double numI,
                     int C_o, double beta, double _g_l, double _g_i,
                         double _g_h, double _g_c, double u, double R_o,
                         double m,
                     double c, double f_o)*/

                     h = 1;  // time step is fixed for every group
                     age_low = 0;
                     age_high = 100;
                     frac = 1;
                     N = 83000000; //      German population as of July 21
```

```
    st, 2020 vis https://www.worldometers.info/world-
    population/germany-population/
priority=1;
numE=20;
numI=1;
C_o=30000;
beta = 1.15; // as my COVID_03
/* setting latancy periods*/
double _g_l=2.6;            //  (latency period),
    latency period is the time it takes after being
    infected to become infectious
double _g_i=2.35;           //  (time to R-recovered
     or H-hospitalized) after becoming infected
double _g_h=4.0;            //     (time from H to C
    -critical=in ICU)
double _g_c=7.5;            //     (time from C to H
     or D-death)

ut = 1;//                   no interventions when
    set to 1
R_o = 2.7;
m = 0.92;
c = 0.27;

f_o=0.27;
f_1=2*f_o;



/* vector used for Testing condition conditions*/
IC_S.push_back(T);          // initial conditions,
    used for comparing 04 with 05
IC_S.push_back(double(age_low));
IC_S.push_back(double(age_high));
```

```
IC_S.push_back(frac);
IC_S.push_back(double(N));
IC_S.push_back(double(priority));

nE=double(numE);
nI=double(numI);
IC_S.push_back(nE);
IC_S.push_back(nI);
IC_S.push_back(double(C_o));
IC_S.push_back(beta);
IC_S.push_back(_g_l);
IC_S.push_back(_g_i);
IC_S.push_back(_g_h);
IC_S.push_back(_g_c);
IC_S.push_back(ut);
IC_S.push_back(R_o);
IC_S.push_back(m);
IC_S.push_back(c);
IC_S.push_back(f_o);



num_N = frac * double (N); // number of people in
    this population i.e. N= frac * Grand_pop
g_l = 1/_g_l;
g_i = 1/_g_i;
g_h = 1/_g_h;
g_c = 1/_g_c;

CauchyData.push_back(0.0);
CauchyData.push_back(0.0);                  // Will do
     this in default/specific constructor set to 20
    like paper on page 18
```

```
CauchyData.push_back(0.0);
CauchyData.push_back(0.0);
CauchyData.push_back(0.0);
CauchyData.push_back(0.0);
CauchyData.push_back(0.0);

size = CauchyData.size();

CauchyData[1] = numE;
CauchyData[2] = numI;
CauchyData[0] = num_N - CauchyData[1]; // Seting
    suscepible S = N - E(0)
//      will adjust in default/specific constructor

/* initializing everthing that I will need for the
    step function*/
CD = 0;
for(int j = 0; j < 7; j++) {
        CD = CauchyData[j];
        W.push_back(CD);

        F1.push_back(0);
        F2.push_back(0);
        F3.push_back(0);
        F4.push_back(0);
        dum_F1.push_back(0);
        dum_F2.push_back(0);
        dum_F3.push_back(0);
        dum_F4.push_back(0);
        dumW.push_back(0);

        // std::cout << "in for loop for J at
            iteration: " << j << std::endl;
```

```
        }

        // making vectors the correct size
        for( int i=0; i<T; i++)
        {

                S.push_back(0);
            E.push_back(0);
            I.push_back(0);
            H.push_back(0);
            C.push_back(0);
            R.push_back(0);
            D.push_back(0);
                ICU.push_back(C_o);
                u.push_back(ut);
        }


        // puting cauchy data into vectors
        S[0]=(W[0]);
        E[0]=(W[1]);
        I[0]=(W[2]);
        H[0]=(W[3]);
        C[0]=(W[4]);
        R[0]=(W[5]);
        D[0]=(W[6]);


}

//parameterized constructor
// popAgeGroup(int T, int age_low, int age_high, double frac
    , int N, int priority, long double numE, long double
```

```
   numI,
 //          int C_o, double beta, double _g_l, double _g_i,
     double _g_h, double _g_c, double ut, double R_o, double
     m,
  //         double c, double f_o) {

popAgeGroup(int T, int N, long double numE, long double numI
   ,
              long double numH,                    //
                 hospitalized
              long double numC,                    // ICU
              long double numR,                    //
                 recovered
              long double numD,                    // Deaths
              long double numV1,                   // 1 dose
                 vacinated
              long double numV2,                   // 2nd
                 dose vacinated
              int    ut_OFF,                              //
                   during intervetnions "OFF" 0->OFF
              int ut_OFF_date,                     // day to
                 turn interventions '"OFF"
          int C_o, double beta, double _g_l, double _g_i,
                    double _g_h, double _g_c, double ut,
                       double R_o,
                    double m, double c, double f_o) {
       h = 1;  // time step is fixed for every group

       this->ut_OFF=ut_OFF; // set ut_OFF to 1 for no change
                                              //     in the
                                                 simulations
                                                 ANE
                                              // set to 0 to
```

```
                                        turn OFF
                                        intervention
this->ut_OFF_date=ut_OFF_date; // setting date of OFF

this->N=N;
this->numE=numE;
this->numI=numI;
this->numH = numH;
this->numC = numC;
this->numR = numR;
this->numD = numD;
this->C_o=C_o;


this->beta=beta; // as my COVID_03

/* setting latancy periods*/


this->_g_l=_g_l;            // (latency period),
   latency period is the time it takes after being
   infected to become infectious
this->_g_i=_g_i;            // (time to R-recovered
    or H-hospitalized) after becoming infected
this->_g_h=_g_h;            //     (time from H to C
   -critical=in ICU)
this->_g_c=_g_c;            //     (time from C to H
    or D-death)

this->ut=ut;
this->R_o=R_o;
this->m=m;
this->c=c;
```

```
this->f_o=f_o;
f_1 = 2 * f_o;

                    /* vector used for Testing
                       condition conditions*/
IC_S.push_back(T);          // initial conditions,
   used for comparing 04 with 05
IC_S.push_back(double(age_low));
IC_S.push_back(double(age_high));
IC_S.push_back(frac);
IC_S.push_back(double(N));
IC_S.push_back(double(priority));

nE=double(numE);
nI=double(numI);
IC_S.push_back(nE);
IC_S.push_back(nI);
IC_S.push_back(double(C_o));
IC_S.push_back(beta);
IC_S.push_back(_g_l);
IC_S.push_back(_g_i);
IC_S.push_back(_g_h);
IC_S.push_back(_g_c);
IC_S.push_back(ut);
IC_S.push_back(R_o);
IC_S.push_back(m);
IC_S.push_back(c);
IC_S.push_back(f_o);



num_N = double(N);
```

```
g_l = 1/_g_l;
g_i = 1/_g_i;
g_h = 1/_g_h;
g_c = 1/_g_c;

CauchyData.push_back(0.0);
CauchyData.push_back(0.0);
CauchyData.push_back(0.0);
CauchyData.push_back(0.0);
CauchyData.push_back(0.0);
CauchyData.push_back(0.0);
CauchyData.push_back(0.0);

size = CauchyData.size();

CauchyData[1] = numE;
CauchyData[2] = numI;
CauchyData[0] = num_N - CauchyData[1]; // Seting
    suscepible S = N - E(0)
//      will adjust in default/specific constructor

/* initializing everthing that I will need for the
    step function*/
CD = 0;
for(int j = 0; j < 7; j++) {
        CD = CauchyData[j];
        W.push_back(CD);

        F1.push_back(0);
        F2.push_back(0);
        F3.push_back(0);
        F4.push_back(0);
        dum_F1.push_back(0);
```

```
        dum_F2.push_back(0);
        dum_F3.push_back(0);
        dum_F4.push_back(0);
        dumW.push_back(0);

        // std::cout << "in for loop for J at
            iteration: " << j << std::endl;
}

// used for testing
// std::cout<<"ut is: "<< ut<<std::endl;
// std::cout<<"size of u vector "<< u.size()<<std::
    endl;


// making vectors the correct size
for( int i=0; i<T; i++)
{

        S.push_back(0);
    E.push_back(0);
    I.push_back(0);
    H.push_back(0);
    C.push_back(0);
    R.push_back(0);
    D.push_back(0);
        ICU.push_back(C_o);
        u.push_back(ut);
}

// puting cauchy data into vectors
S[0]=(CauchyData[0]);
E[0]=(numE);
```

```
        I[0]=(numI);
        H[0]=(numH);
        C[0]=(numC);
        R[0]=(numR);
        D[0]=(numD);

}


// destructor
~popAgeGroup() {}


void get_ut(int i){
        std::cout<<"At day "<<i<<" the intervention u is: "<<
            u[i]<<std::endl;
}

void set_Beta(double beta){
        this->beta=beta;
}
void step (int i) {

        double udown1, udown2, up1, up2;
if (i>1){
        u[i] = u[i-1];

        //for first wave
        if(i<80&& i %5==0){
                if((C[i]/C_o)>=0.02 || I[i]>0.10*C_o){
                //      std::cout<<u[i-1]<<std::endl;
                        udown1 = u[i-1] -0.20;//*(1+C[i]/C_o);
                        u[i]=udown1;
```

```
        }else if(I[i]>0.125*C_o){
                u[i]=u[i-1]-0.3;
        }

        //floor
        if(u[i]<0.300){
                u[i]=0.30;
        }
        if((C[i]/C_o)<0.04 && I[i]<=0.7*C_o){
                u[i] = u[i-1] + 0.10;
//              std::cout<<"Relax ut if statement at
    step: "<< i<<std::endl;
                //std::cout<<"ut value at step "<<i<<"
                    is: "<<u[i]<<std::endl;
        }
        // max
        if (u[i]>1.00){
                u[i] =1.00;
        }
//      std::cout<<"ut value at step "<<i<<" is: "<<u[
    i]<<std::endl;

}

//working into second wave
if(i>=80 && i % 4==0 && i<385){
        if((C[i]/C_o)>=0.017 || I[i]>0.09*C_o){
        //      std::cout<<u[i-1]<<std::endl;
                udown1 = u[i-1] -0.15;//*(1+C[i]/C_o);
                u[i]=udown1;
        }else if(I[i]>0.12*C_o){
                u[i]=u[i-1]-0.32;
        }
```

```
        //floor
        if(u[i]<0.4){//schools open
                u[i]=0.40;
        }

        if((C[i]/C_o)<0.05 && I[i]<=1.1*C_o){
                u[i] = u[i-1] + 0.10;
//              std::cout<<"Relax ut if statement at
    step: "<< i<<std::endl;
                //std::cout<<"ut value at step "<<i<<"
                    is: "<<u[i]<<std::endl;
        }
        // max
        if (u[i]>1.00){
                u[i] =1.00;
        }
//      std::cout<<"ut value at step "<<i<<" is: "<<u[
    i]<<std::endl;
}

// working on Start third wave (parks open up)
if(i>=385 && i % 4==0 && i<410){
        if((C[i]/C_o)>=0.017 || I[i]>0.09*C_o){
        //      std::cout<<u[i-1]<<std::endl;
                udown1 = u[i-1] -0.15;//*(1+C[i]/C_o);
                u[i]=udown1;
        }else if(I[i]>0.12*C_o){
                u[i]=u[i-1]-0.32;
        }
        //floor
        if(u[i]<0.48){//parks
                u[i]=0.48;
        }
```

```
        if((C[i]/C_o)<0.05 && I[i]<=1.1*C_o){
                u[i] = u[i-1] + 0.10;
//              std::cout<<"Relax ut if statement at
    step: "<< i<<std::endl;
                //std::cout<<"ut value at step "<<i<<"
                    is: "<<u[i]<<std::endl;
        }
        // max
        if (u[i]>1.00){
                u[i] =1.00;
        }
//      std::cout<<"ut value at step "<<i<<" is: "<<u[
    i]<<std::endl;
}
// working on April 9th stay at home, schools closed,
     parks closed


if(ut_OFF==1){ // runing with the interventions still
     on
        if(i>=410 && i % 5==0){
                if((C[i]/C_o)>=0.019 || I[i]>0.11*C_o)
                    {
                //      std::cout<<u[i-1]<<std::endl;
                        udown1 = u[i-1] -0.15;//*(1+C[i
                            ]/C_o);
                        u[i]=udown1;
                }else if(I[i]>0.14*C_o){
                        u[i]=u[i-1]-0.32;
                }
                //floor
                if(u[i]<0.30){//schools open
```

```
                                        u[i]=0.30;
                                }

                                if((C[i]/C_o)<0.05 && I[i]<=1.1*C_o){
                                        u[i] = u[i-1] + 0.10;
                        //              std::cout<<"Relax ut if
                            statement at step: "<< i<<std::endl;
                                        //std::cout<<"ut value at step
                                            "<<i<<"is: "<<u[i]<<std::
                                            endl;
                                }
                                // max
                                if (u[i]>1.00){
                                        u[i] =1.00;
                                }
                        //      std::cout<<"ut value at step "<<i<<"
                            is: "<<u[i]<<std::endl;
                        }
                } else if(ut_OFF==0 && i> ut_OFF_date) {
                        u[i] = 1; // no more intervention
                }


//                      }
//                      int     ut_OFF,                         // during
    intervetnions "OFF" 0->OFF
//                              int ut_OFF_date,

        }
```

```
dum_F1[0]=S[i];
dum_F1[1]=E[i];
dum_F1[2]=I[i];
dum_F1[3]=H[i];
dum_F1[4]=C[i];
dum_F1[5]=R[i];
dum_F1[6]=D[i];

//std::cout << "in for loop for i at iteration: " <<
    i << std::endl;

F1 = F(dum_F1, i);

//     std::cout << "in for loop for i at iteration:
    " << i << std::endl;

dum_F2[0] = S[i] + h * F1[0]/2;
dum_F2[1] = E[i] + h * F1[1]/2;
dum_F2[2] = I[i] + h * F1[2]/2;
dum_F2[3] = H[i] + h * F1[3]/2;
dum_F2[4] = C[i] + h * F1[4]/2;
dum_F2[5] = R[i] + h * F1[5]/2;
dum_F2[6] = D[i] + h * F1[6]/2;
F2 = F(dum_F2,i);

dum_F3[0] = S[i] + h * F2[0]/2;
dum_F3[1] = E[i] + h * F2[1]/2;
dum_F3[2] = I[i] + h * F2[2]/2;
dum_F3[3] = H[i] + h * F2[3]/2;
dum_F3[4] = C[i] + h * F2[4]/2;
dum_F3[5] = R[i] + h * F2[5]/2;
dum_F3[6] = D[i] + h * F2[6]/2;
F3 = F(dum_F3,i);
```

```
        dum_F4[0] = S[i] + h * F3[0];
        dum_F4[1] = E[i] + h * F3[1];
        dum_F4[2] = I[i] + h * F3[2];
        dum_F4[3] = H[i] + h * F3[3];
        dum_F4[4] = C[i] + h * F3[4];
        dum_F4[5] = R[i] + h * F3[5];
        dum_F4[6] = D[i] + h * F3[6];
        F4 = F(dum_F4,i);


        ss=2;
        S[i+1]=S[i] + h * (F1[0] +2* F2[0] + 2*F3[0] + F4[0])
            /6;
        E[i+1]=E[i] + h * (F1[1] + 2*F2[1] + 2*F3[1] + F4[1])
            /6;
        I[i+1]=I[i] + h * (F1[2] + 2*F2[2] + 2*F3[2] + F4[2])
            /6;
        H[i+1]=H[i] + h * (F1[3] + 2*F2[3] + 2*F3[3] + F4[3])
            /6;
        C[i+1]=C[i] + h * (F1[4] + 2*F2[4] + 2*F3[4] + F4[4])
            /6;
        R[i+1]=R[i] + h * (F1[5] + 2*F2[5] + 2*F3[5] + F4[5])
            /6;
        D[i+1]=D[i] + h * (F1[6] + 2*F2[6] + 2*F3[6] + F4[6])
            /6;

}



void write_initial_conditons(){
```

```
        int size = IC_S.size();
        for(int i = 0; i < size; i++){
                std::cout<< IC_S[i]<<std::endl;
        }
}


void finalize(std::string savename1, std::string
    savename_read, int T) {
        // need int T for total time steps, need for the for
            loop in writing to file.

        /*      Printing out vectors T, S, E, I, H, C, R, D
                seperated buy 'tab' and then by endl after D
                For example:
                T    S      E      I      H      C      R
                          D
                0    9      9      9      9      9      9
                          9
                1    9      9      9      9      9      9
                          9
                ...                ...
                            ...
                T    9      9      9      9      9      9
                          9

                I hope that this would make it easy to do
                graphing in Matlab
          */
        int vS = T;

        std::fstream out_COVID;
```

```
std::fstream out_COVID_READ;
out_COVID.open(savename1.c_str(), std::ios::out);
out_COVID_READ.open(savename_read.c_str(), std::ios::
    out);

//Note: tthe extra 'title' line was needed to get
    matlab to read
// the file properly.... have not figured out way,
    yet.

out_COVID<< "T" << '\t' << "S" << '\t' << "E" << '\t'
     << "I"
         << '\t' << "H" << '\t' << "C" << '\t' << "R"
            << '\t' << "D"
               << '\t' << "ICU"<<std::endl;
out_COVID<< "T" << '\t' << "S" << '\t' << "E" << '\t'
     << "I"
        << '\t' << "H" << '\t' << "C" << '\t' << "R"
           << '\t' << "D"
        << '\t' << "ICU"<<std::endl;

out_COVID_READ<< "T" << '\t'<< '\t'<< '\t' << "S" <<
    '\t'<< '\t'<< '\t' << '\t'
             << "E" << '\t'<< '\t'<< '\t' << "I" <<
                '\t'<< '\t' << '\t'
             << "H" << '\t'<< '\t'<< '\t'<< "C" << '\
                t'<< '\t'<< '\t'<< "R"
             << '\t'<< '\t'<< '\t'<< "D" << '\t' <<
                '\t'<<"ICU"<<std::endl;

for(int i = 0; i < vS; i++) {
       out_COVID<< i+1 << '\t' << S[i] << '\t' << E[i
          ] << '\t' << I[i]
```

```
                        << '\t' << H[i] << '\t' << C[i] << '\
                           t' << R[i] << '\t' << D[i] << '\t
                           '<< ICU[i]<< std::endl;
                out_COVID_READ<< i+1 << '\t' << '\t' << '\t'
                   << S[i] << '\t' << '\t'<< '\t'<< E[i] <<
                   '\t' << '\t'<< '\t' << I[i]
                            << '\t' << '\t'<< '\t' << H[i] <<
                                 '\t' << '\t' << '\t' << C[i]
                            << '\t'<< '\t'<< '\t' << R[i] <<
                               '\t' << '\t'<< '\t' << D[i]
                               << '\t'<< ICU[i]<< std::endl;

                //std::cout<< i << std::endl;
        }

        out_COVID.close();
}


/*      Remember that F: R^7 -> R^7 for understanding, also
    that I will
        have to make a vector of size 7 to have everything
            work well.
          S,    E, I, H, C, R, D     correspond to
        x[0], x[1], ....,    x[6]
*/



void intervention_finalize(std::string savename1, int T) {
// need int T for total time steps, need for the for loop in
     writing to file.
```

```
        int vS = T;

        std::fstream intervention_u;
        intervention_u.open(savename1.c_str(), std::ios::out)
            ;

        intervention_u<< "T" << '\t' << "ut" <<std::endl;
        intervention_u<< "T" << '\t' << "ut" <<std::endl;

        for(int i = 0; i < vS; i++) {
               intervention_u<< i+1 << '\t' << u[i] << std::
                   endl;
        }

        intervention_u.close();
}


/*      Remember that F: R^7 -> R^7 for understanding, also
    that I will
        have to make a vector of size 7 to have everything
            work well.
          S,    E, I, H, C, R, D     correspond to
        x[0], x[1], ....,    x[6]
*/
std::vector<long double> F(std::vector<long double> & K, int
     t) {
        // making u a vector in private ('s), all I have to
            do is pass index into function
        std::vector <long double> Z;
        int size = K.size();
        for(int i = 0; i<size; i++) {
               Z.push_back(i);
```

```
        }

        // setting function for f-ICU mortality
        if(K[4] >= C_o) {
               f = f_1 - (C_o / K[4]) * (f_1 - f_o);
        } else {
               f = f_o;
        }



        Z[0] = -1 * beta * u[t] * K[2] / N * K[0];
                                          //     dS/dt
        Z[1] = beta * u[t] * K[2] / N * K[0] - g_l * K[1];
                                  //     dE/dt
        Z[2] =g_l * K[1] - g_i * K[2] ;
                                                 //     dI/dt
        Z[3] =(1-m) * g_i * K[2] + (1 - f) * g_c * K[4] - g_h
            * K[3];  // dH/dt
        Z[4] =c * g_h * K[3] - g_c * K[4];
                                                 //     dC/dt
        Z[5] = m * g_i * K[2] + (1-c) * g_h * K[3];
                                          //     dR/dt
        Z[6] = f * g_c * K[4];
                                                        //
                 dD/dt

        return Z;
    }

private:
    //declare variables
    // all set in days
```

```
/*(int age_low, int age_high, double frac, int Grand_pop,
   int priority, long double numE, long double numI,
              int C_o, double beta, double _g_l, double _g_i
                  , double _g_h, double _g_c, double u,
                  double R_o, double m,
              double c, double f_o, double f)*/

int    age_low, age_high;
double frac;                        //fraction of the
   population that this group makes up
int            priority;                    // set to 1 if
   top, set to 2, if other (maybe random)
long double numE;                   // exposed
long double numI;                   // infectious
long double numH;                   // hospitalized
long double numC;                   // ICU
long double numR;                   // recovered
long double numD;                   // Deaths
long double numV1;                  // 1 dose vacinated
long double numV2;                  // 2nd dose vacinated
int            C_o;                         //     total
   number of ICU beds avaiable for population

int    ut_OFF;                              // during
   intervetnions "OFF" 0->OFF
int ut_OFF_date;                    // day to turn
   interventions '"OFF"

double _g_l;
double _g_i;
double _g_h;
double _g_c;
```

```
/* for checking Initial conditions for numE and numI, respectively*/
              double nE;
              double nI;

              double beta;  //transmision rate
              double g_l;          //  1/(latency period), latency period
                 is the time it takes after being infected to become
                 infectious
              double g_i;          //  1/(time to R-recovered or H-
                 hospitalized) after becoming infected
              double g_h;          //     1/(time from H to C-critical=in
                  ICU)
              double g_c;          //     1/(time from C to H or D-death)
              double ut;           //     is the modification of
                 transmission rate 1 if nothing is done, 0 if total
                 isolation

              double R_o;          //     basic reproduction number, not
                 sure where this is used YET
              ////////////////////////////////////////////////


              double N;            //     is the total number of people
                 in the population
              double num_N; //  is the total number of people in the group
                  i.e. N * frac = num_N

              double m;            //     share of infected population
                 that are asymtomatic or with mild symptoms
              double c;            //     fraction of hospitalized to
                 become critical C/H -> ICU
```

```
            double f_o;          //     mortality rate of patients in
               ICU
            double f_1; // mortality of critical patients without ICU
               beds
            /*     f below may not need to be global*/
            double f;            //     fraction of critical to become
               dead D/C

            double h;     // time step is fixed for every group

            //     making vectors S, E, I, H, C, R, D to recorde
               populations of each type
            //     at each time step in T
////////////////////////////////
//          for testing and insuring physical states, i.e. non negative
   populations
            double ss;
            double ee;
            double ii;
            double hh;
            double cc;

//////////////


            std::vector <long double> CauchyData; // Y = [

            std::vector <long double> S; //     susceptible
            std::vector <long double> E; //     Exposed
            std::vector <long double> I; //     infectious
            std::vector <long double> H; //     Hospitalized
            std::vector <long double> C; //     ICU
            std::vector <long double> R; //     Recovered
```

```
            std::vector <long double> D; //     deceased
            std::vector <long double> ICU; // ICU capcity
            std::vector <double> u; // intervention

            std::vector <double> IC_S;          // initial conditions,
               used for comparing 04 with 05

////////////////////////////////////////////////////////////////////////
            /* setting all needed variables/vectors for method */
            int size;
            std::vector <long double> W;  // edited out T recall that T
               has been made into a int


            //     RK4

            std::vector<long double> dum_F1;
            std::vector<long double> F1;

            std::vector<long double>dum_F2;
            std::vector<long double> F2;

            std::vector<long double> dum_F3;
            std::vector<long double> F3;

            std::vector<long double> dum_F4;
            std::vector<long double> F4;

            std::vector<long double> dumW;


            long double CD;
```

```
};

int main() {


        double ut = 1.00;     // intervention

        int T = 365 * 100;    // ten years for simulation to run its course
        int N = 14745040; // population Ontario
        long double numE = 20; // exposed initially
        long double numI = 0; // infecticious initially
        long double numH = 0;               // hospitalized
        long double numC = 0;               // ICU
        long double numR = 0;               // recovered
        long double numD = 0;               // Deaths
        long double numV1 = 0;              // 1 dose vacinated
        long double numV2 = 0;              // 2nd dose vacinated
        int    ut_OFF =0;                                       // during
           intervetnions "OFF" 0 ON 1
        int ut_OFF_date = 560;              // day to turn interventions '"
           OFF"
        int C_o = 14*N/100000; // ICU beds available


        // found error in paper, brackets in wrong spot should only be on
           the d-days and not 1/1.15
        double beta = 1.15;   // transmission rate
        double beta_L = beta*1.40; // low end of transmission rate of
           variants
        double beta_M = beta*1.65; // Median
        double beta_H = beta*1.90;// High end of transmission rate of
```

```
        variants
    double beta_step_L=beta; // for changing beta for variants - LOW
    double beta_step_M=beta; // for changing beta for variants - MEDIAN
    double beta_step_H=beta; // for changing beta for variants - HIGH


    double _g_l = 2.6;    //average latency
    double _g_i = 2.35;   // average infectious period before recovery
        or hospitalization
    double _g_h = 4.0;    // average Hospital -> ICU or recover
    double _g_c = 7.5;    // average before ICU -> die or recover
    // double ut = 1;            // initial intervention


    // R_o doesn't do anything!!!!!!
    double R_o = 2.7;     // basic reproduction number
    double m = 0.92;      // fraction of infected with at most mild
        symptoms
    double c = 0.27;      // fraction of hospitialized patients that
        turn to -> ICU
    double f_o = 0.31;    // fortality of critical (ICU) patient with
        ICU


////////////////////////////////////////////////////////////////
//
// JUST TO MAKE EASY TO ENTER DATA/ IC'S
//
////////////////////////////////////////////////////////////////

    //popAgeGroup* Tot_Pop = new popAgeGroup(T);

    /*
```

```
popAgeGroup(int T, int N, long double numE, long double numI,
                            long double numH,                    //
                               hospitalized
                            long double numC,                    // ICU
                            long double numR,                    //
                               recovered
                            long double numD,                    // Deaths
                            long double numV1,                   // 1 dose
                               vacinated
                            long double numV2,                   // 2nd
                               dose vacinated
                            int    ut_OFF,                                //
                                during intervetnions "OFF" 0 ON 1
                            int ut_OFF_date,                     // day to
                               turn interventions '"OFF"
                        int C_o, double beta, double _g_l, double _g_i,
                        int C_o, double beta, double _g_l, double _g_i,
                                  double _g_h, double _g_c, double ut,
                                      double R_o,
                                  double m, double c, double f_o)
      */



      // variant free
popAgeGroup* ON_V_0 = new popAgeGroup(T, N, numE, numI,
                            numH,                // hospitalized
                            numC,                // ICU
                            numR,                // recovered
                            numD,                // Deaths
                            numV1,               // 1 dose vacinated
                            numV2,               // 2nd dose vacinated
```

```
                                ut_OFF,              // turning interventions
                                     "OFF" 0 ON 1
                                ut_OFF_date, // day to turn interventions "
                                    OFF"
                           C_o, beta,  _g_l, _g_i,
                                        _g_h,  _g_c, ut, R_o,
                                        m,  c,  f_o);

// variant low transmission
popAgeGroup* ON_V_L = new popAgeGroup(T, N, numE, numI,
                                numH,                // hospitalized
                                numC,                // ICU
                                numR,                // recovered
                                numD,                // Deaths
                                numV1,               // 1 dose vacinated
                                numV2,               // 2nd dose vacinated
                                ut_OFF,              // turning interventions
                                     "OFF" 0 ON 1
                                ut_OFF_date, // day to turn interventions "
                                    OFF"
                           C_o, beta,  _g_l, _g_i,
                                        _g_h,  _g_c, ut, R_o,
                                        m,  c,  f_o);

// variant median transmission
popAgeGroup* ON_V_M = new popAgeGroup(T, N, numE, numI,
                                numH,                // hospitalized
                                numC,                // ICU
                                numR,                // recovered
                                numD,                // Deaths
                                numV1,               // 1 dose vacinated
                                numV2,               // 2nd dose vacinated
                                ut_OFF,              // turning interventions
```

```
                                    "OFF" 0 ON 1
                                ut_OFF_date, // day to turn interventions "
                                    OFF"
                        C_o, beta,  _g_l, _g_i,
                                      _g_h,  _g_c, ut, R_o,
                                      m,  c,  f_o);
// variant high transmission
popAgeGroup* ON_V_H = new popAgeGroup(T, N, numE, numI,
                                numH,               // hospitalized
                                numC,               // ICU
                                numR,               // recovered
                                numD,               // Deaths
                                numV1,              // 1 dose vacinated
                                numV2,              // 2nd dose vacinated
                                ut_OFF,             // turning interventions
                                    "OFF" 0 ON 1
                                ut_OFF_date, // day to turn interventions "
                                    OFF"
                        C_o, beta,  _g_l, _g_i,
                                      _g_h,  _g_c, ut, R_o,
                                      m,  c,  f_o);

// I don't think i need this function at all!!
//     Test_Tot_Pop->write_initial_conditons();

       for (int t = 0; t < T; t++) {
              ON_V_0->step(t);
              ON_V_L->step(t);
              ON_V_M->step(t);
              ON_V_H->step(t);
//            if (t==240){
//                   Test_Tot_Pop->set_Beta(beta2);
//            }
```

```
        // setting up for variants of different transmission rates.
        if (t>390 && beta_step_L< beta_L){
                beta_step_L = beta_step_L+(beta_L - beta)/30;
                ON_V_L->set_Beta(beta_step_L);
        }
        if (t>390 && beta_step_M< beta_M){
                beta_step_M = beta_step_M+(beta_M - beta)/30;
                ON_V_M->set_Beta(beta_step_M);
        }
        if (t>390 && beta_step_H< beta_H){
                beta_step_H = beta_step_M+(beta_H - beta)/30;
                ON_V_H->set_Beta(beta_step_H);
        }

}

ON_V_0->finalize("ON_VAC_0_01.txt","Test_COVID_READ_Tot_Pop.txt", T
    );
ON_V_L->finalize("ON_VAC_L_01.txt","Test_COVID_READ_Tot_Pop.txt", T
    );
ON_V_M->finalize("ON_VAC_M_01.txt","Test_COVID_READ_Tot_Pop.txt", T
    );
ON_V_H->finalize("ON_VAC_H_01.txt","Test_COVID_READ_Tot_Pop.txt", T
    );

ON_V_0->intervention_finalize("ON_ut_VAC_0_01.txt", T);
ON_V_L->intervention_finalize("ON_ut_VAC_L_01.txt", T);
ON_V_M->intervention_finalize("ON_ut_VAC_M_01.txt", T);
ON_V_H->intervention_finalize("ON_ut_VAC_H_01.txt", T);

////////////////////////////////////////////////////////////////
//
```

```
        //     TO HELP THINKING THROUGH WAVES AND WHEN TO
        //     INTRODUCE VARIANTS IN THE SIMULATION
        //
        // start feb 1st
        // march 30 days
        // april 60
        // may 90
        // june 120
        //     July   150
        // August     180
        // Sept       210
        // Oct        240
        // Nov        270
        // Dec        300
        // jan        330
        // Feb        360
        //     march  390
        //     April  410
        //     May           440
        // june       470
        delete ON_V_0;
        delete ON_V_L;
        delete ON_V_M;
        delete ON_V_H;

        return 0;
}
```

C.3 MatLab Code for Plotting and inspecting data

This MatLab code was used to produce all plots and to inspect the data (i.e. look for max values). Instead of saving different implementations for each simulation, I adjusted the code each time I ran another simulation. In hindsight, it would have

been better to save the code that produce each figure so that it would be simpler to reproduce figures again.

```
function ploting03;
% reads in data from text file, note I must adjust the name of file being
% read in.
% will make 'nice' plot, and I will also normalize the data to the
% population size.

 %fclose(fileID);

 % reading full data sets
 ON_V_00=fopen('ON_VAC_0_01.txt', 'r');
 ON_V_L=fopen('ON_VAC_L_01.txt', 'r');
 ON_V_M=fopen('ON_VAC_M_01.txt', 'r');
 ON_V_H=fopen('ON_VAC_H_01.txt', 'r');

 %reading the intervention
 ON_V_00_ut=fopen('ON_ut_VAC_0_01.txt', 'r');
 ON_V_L_ut=fopen('ON_ut_VAC_L_01.txt', 'r');
 ON_V_M_ut=fopen('ON_ut_VAC_M_01.txt', 'r');
 ON_V_H_ut=fopen('ON_ut_VAC_H_01.txt', 'r');

 ut_C = textscan(ON_V_00_ut, '%f%f', 'Delimiter','[;', 'HeaderLines',2);
 C = textscan(ON_V_00, '%f%f%f%f%f%f%f%f%f%f', 'Delimiter','[;', '
    HeaderLines',2);
 Data_0=cell2mat(C);
 ut_0=cell2mat(ut_C);
 ut_0(1,2)=0;
  C = textscan(ON_V_L, '%f%f%f%f%f%f%f%f%f%f', 'Delimiter','[;', '
     HeaderLines',2);
 Data_L=cell2mat(C);
```

```
 C = textscan(ON_V_M, '%f%f%f%f%f%f%f%f%f%f', 'Delimiter','[;', '
     HeaderLines',2);
Data_M=cell2mat(C);
 C = textscan(ON_V_H, '%f%f%f%f%f%f%f%f%f%f', 'Delimiter','[;', '
     HeaderLines',2);
Data_H=cell2mat(C);
% normalize data
%


% interval of interest
start = 1;
[m n] = size(Data_0)
% fin = 365*3;
fin = 1001;
Tot_Pop = Data_0(1,2);
Data_L=Data_L./Tot_Pop;
% Normalizing ON_V_L

 % ICU capacity
Co=14*Tot_Pop/100000;
%T = Data_0(start:fin,1);
T = Data_0(start:fin,1);
T_max = max(T);


% ut=ut_0(start:fin,2);
%  %% Population NO variants
%         %normalizing
%         %D = D./Tot_Pop;
%
%         % S1=Data_0(1,2)
%         % E1 =Data_0(1,3)
```

```
%         S0 = Data_0(start:fin,2); E0=Data_0(start:fin,3); I0=Data_0(
    start:fin,4);
%         H0=Data_0(start:fin,5); C0=Data_0(start:fin,6); R0=Data_0(start:
    fin,7);
%         DD0=Data_0(start:fin,8);ICU0=Data_0(start:fin,9);
%


%         %% template plot
%         % plot(T, S, T, E, T, I, T, H, T, C, T, R, T, DD, T, ICU)
%
%         %% template legend
%         % legend('$S$','$E$','$I$','$H$','$C$','$R$', '$D$', '$C_{o
    }=30000$','Interpreter','latex');
%         % legend('$S$','$E$','$I$','$H$','$C$','$R$', '$D$', '$C_{o
    }=30000$','Interpreter','latex', 'NumColumns',2);
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %      Going to add ut to each plot, must carefully adjust the scale
% %%%%%%%%%%%%%%%%%%%%%%%%%
%         fig_1=figure('Name','ON Population no Variatants');
%        left_color = [0 0 0];
%         right_color = [0 0 0];
%         set(fig_1,'defaultAxesColorOrder',[left_color; right_color]);
%         pCo = plot(T, ICU0, 'Color', '#FF0000', 'LineStyle','--', '
    DisplayName','C0=30000');
%         hold on
%    %     pS1=plot(T, S0, 'Color', '#0072BD', 'DisplayName', 'S');
%         pT2=plot(T, E0, 'Color', '#EDB120', 'DisplayName', 'E');
%         pI3 = plot(T, I0, 'Color', '#A2142F', 'DisplayName','I');
%         pH4 = plot(T, H0, 'Color', '#FF00FF', 'DisplayName', 'H');
%         pC5 = plot(T, C0, 'Color', '#FF0000', 'DisplayName', 'C');
%     %    pR6 = plot(T, R0, 'Color', '#00FF00', 'DisplayName', 'R');
%         pD7 = plot(T, DD0, 'Color', '#000000', 'DisplayName', 'D');
%         %title('Line Plot of Sine and Cosine Between -2\pi and 2\pi')
```

```
%         xlabel('time [d]')
%         yyaxis left
%         ylabel('ON Population')
%         %pUt = plot(T, ut, 'Color', '#0072BD', 'DisplayName', 'u(t)');
%         yyaxis right
%         xlim([0.0 T_max])
%         ylim([0.0 1.0])
%         ylabel('Intervernt u(t)')
%         pUt = plot(T, ut, 'Color', '#000000', 'DisplayName', 'u(t)');
%         legend('$C_{o}=30000$','$E$','$I$','$H$','$C$', '$D$', '$u(t)$','
    Interpreter','latex', 'NumColumns',7);
%         %legend('$C_{o}=30000$','$S$','$E$','$I$','$H$','$C$','$R$', '$D$
    ','Interpreter','latex', 'NumColumns',8);
%         % pCo = plot(T, ICU, 'Color', '#FF0000', 'LineStyle','--'): %,
    '#FF0000', 'Marker', 'v');
%         %pC0.Marker = '--';
% %         set(fig1,'defaultAxesColorOrder',[left_color; right_color]);
%         hold off
%         Imax = max(I0)% infectious
%         Cmax = max(C0)% ICU
%         CmultiOver = Cmax/Co
%         Dend_T = max(DD0)
%         % savefig('nicefig.pdf');
%         %print -depsc myfig.fig
%         print -depsc 01_TEst_Fig
%
%         fig = figure('Name','ICU-no Variants');
%         left_color = [0 0 0];
%         right_color = [0 0 0];
%         set(fig,'defaultAxesColorOrder',[left_color; right_color]);
%
%         % print(gcf, '-dpdf', 'test.pdf');
%         yyaxis left
```

```
%         xlim([0.0 T_max])
%         pC5 = plot(T, C0, 'Color', '#FF0000', 'DisplayName', 'C');
%         hold on
%         pCo = plot(T, ICU0, 'Color', '#FF0000', 'LineStyle','--', '
    DisplayName','C0=30000');
%
%
%         legend('$C_{o}=30000$','$C$','$D$','Interpreter','latex', '
    NumColumns',3);
%         ylabel('ICU Beds in Use');
%         yyaxis right
%         pD7 = plot(T, DD0, 'Color', '#000000', 'DisplayName', 'D');
%         ylabel('Total number of Deaths during Critical period');
%         xlabel('time [d]')
%
%         hold off
%
 %% Population WITH variant L
        %normalizing
        %D = D./Tot_Pop;
        % S1=Data_0(1,2)
        % E1 =Data_0(1,3)
        S0 = Data_L(start:fin,2); E0=Data_L(start:fin,3); I0=Data_L(start:
            fin,4);
        H0=Data_L(start:fin,5); C0=Data_L(start:fin,6); R0=Data_L(start:
            fin,7);
        DD0=Data_L(start:fin,8);ICU0=Data_L(start:fin,9);

        %% template plot
        % plot(T, S, T, E, T, I, T, H, T, C, T, R, T, DD, T, ICU)

        %% template legend
```

```
 % legend('$S$','$E$','$I$','$H$','$C$','$R$', '$D$', '$C_{o}=30000
    $','Interpreter','latex');
 % legend('$S$','$E$','$I$','$H$','$C$','$R$', '$D$', '$C_{o}=30000
    $','Interpreter','latex', 'NumColumns',2);

 fig_2=figure('Name','ON Population With Variants (L-transmission)
    ');
 left_color = [0 0 0];
 right_color = [0 0 0];
 set(fig_2,'defaultAxesColorOrder',[left_color; right_color]);
% pCo = plot(T, ICU0, 'Color', '#FF0000', 'LineStyle','--', '
   DisplayName','C0=30000');
 hold on
 pS1=plot(T, S0, 'Color', '#0072BD', 'DisplayName', 'S');
 %pT2=plot(T, E0, 'Color', '#EDB120', 'DisplayName', 'E');
 %pI3 = plot(T, I0, 'Color', '#A2142F', 'DisplayName','I');
 %pH4 = plot(T, H0, 'Color', '#FF00FF', 'DisplayName', 'H');
 %pC5 = plot(T, C0, 'Color', '#FF0000', 'DisplayName', 'C');
 pR6 = plot(T, R0, 'Color', '#00FF00', 'DisplayName', 'R');
 pD7 = plot(T, DD0, 'Color', '#000000', 'DisplayName', 'D');
 %title('Line Plot of Sine and Cosine Between -2\pi and 2\pi')
 %xlabel('time [d]')
 xlabel('time [d]')
 %yyaxis left
 ylabel('ON Share of Population')
 %pUt = plot(T, ut, 'Color', '#0072BD', 'DisplayName', 'u(t)');
 %yyaxis right
 xlim([0.0 T_max])
%  ylim([0.0 1.0])
%  ylabel('Intervernt u(t)')
%  pUt = plot(T, ut, 'Color', '#000000', 'DisplayName', 'u(t)');
legend('$S$','$R$', '$D$','Interpreter','latex', 'NumColumns',3);
```

```
%legend('$C_{o}=30000$','$E$','$I$','$H$','$C$', '$D$', 'Interpreter
    ','latex', 'NumColumns',6);
 % legend('$C_{o}=30000$','$E$','$I$','$H$','$C$', '$D$', '$u(t)$','
    Interpreter','latex', 'NumColumns',7);
 %legend('$C_{o}=30000$','$S$','$E$','$I$','$H$','$C$','$R$', '$D$
    ','Interpreter','latex', 'NumColumns',8);
 % pCo = plot(T, ICU, 'Color', '#FF0000', 'LineStyle','--'): %, '#
    FF0000', 'Marker', 'v');
 %pC0.Marker = '--';
%  set(fig1,'defaultAxesColorOrder',[left_color; right_color]);
 hold off

 Imax_L = max(I0)*Tot_Pop
 Cmax_L = max(C0)*Tot_Pop
 CmultiOver_L = Cmax_L/Co
 Dend_T_L = max(DD0)*Tot_Pop
 % savefig('nicefig.pdf');
 %print -depsc myfig.fig
 %print -depsc epsFig

 fig = figure('Name','ICU-Variants(L-transmission)');
 left_color = [0 0 0];
 right_color = [0 0 0];
 set(fig,'defaultAxesColorOrder',[left_color; right_color]);

 % print(gcf, '-dpdf', 'test.pdf');
 xlim([0.0 T_max])
 yyaxis left
 pC5 = plot(T, C0*Tot_Pop, 'Color', '#FF0000', 'DisplayName', 'C');
 hold on
 pCo = plot(T, ICU0*Tot_Pop, 'Color', '#FF0000', 'LineStyle','--',
    'DisplayName','C0=30000');
```

```
        legend('$C_{o}=30000$','$C$','$D$','Interpreter','latex', '
            NumColumns',3);
        ylabel('ICU Beds in Use');
        yyaxis right
        pD7 = plot(T, DD0*Tot_Pop, 'Color', '#000000', 'DisplayName', 'D')
            ;
        ylabel('Total number of Deaths');
        %xlabel('time [d]')
        xlabel('time [d]')

        hold off

% %% Population WITH variant M
%         %normalizing
%         %D = D./Tot_Pop;
%         % S1=Data_0(1,2)
%         % E1 =Data_0(1,3)
%         S0 = Data_M(start:fin,2); E0=Data_M(start:fin,3); I0=Data_M(
    start:fin,4);
%         H0=Data_M(start:fin,5); C0=Data_M(start:fin,6); R0=Data_M(start:
    fin,7);
%         DD0=Data_M(start:fin,8);ICU0=Data_M(start:fin,9);
%
%         %% template plot
%         % plot(T, S, T, E, T, I, T, H, T, C, T, R, T, DD, T, ICU)
%
%         %% template legend
%         % legend('$S$','$E$','$I$','$H$','$C$','$R$', '$D$', '$C_{o
    }=30000$','Interpreter','latex');
%         % legend('$S$','$E$','$I$','$H$','$C$','$R$', '$D$', '$C_{o
    }=30000$','Interpreter','latex', 'NumColumns',2);
%
```

```
%         fig_3=figure('Name','ON Population With Variants (M-tranmission)
    ');
%        left_color = [0 0 0];
%         right_color = [0 0 0];
%         set(fig_3,'defaultAxesColorOrder',[left_color; right_color]);
%         pCo = plot(T, ICU0, 'Color', '#FF0000', 'LineStyle','--', '
    DisplayName','C0=30000');
%         hold on
%    %     pS1=plot(T, S0, 'Color', '#0072BD', 'DisplayName', 'S');
%         pT2=plot(T, E0, 'Color', '#EDB120', 'DisplayName', 'E');
%         pI3 = plot(T, I0, 'Color', '#A2142F', 'DisplayName','I');
%         pH4 = plot(T, H0, 'Color', '#FF00FF', 'DisplayName', 'H');
%         pC5 = plot(T, C0, 'Color', '#FF0000', 'DisplayName', 'C');
%     %    pR6 = plot(T, R0, 'Color', '#00FF00', 'DisplayName', 'R');
%         pD7 = plot(T, DD0, 'Color', '#000000', 'DisplayName', 'D');
%         %title('Line Plot of Sine and Cosine Between -2\pi and 2\pi')
%         xlabel('time [d]')
%         yyaxis left
%         ylabel('ON Population')
%         %pUt = plot(T, ut, 'Color', '#0072BD', 'DisplayName', 'u(t)');
%         yyaxis right
%         xlim([0.0 T_max])
%         ylim([0.0 1.0])
%         ylabel('Intervernt u(t)')
%         pUt = plot(T, ut, 'Color', '#000000', 'DisplayName', 'u(t)');
%         legend('$C_{o}=30000$','$E$','$I$','$H$','$C$', '$D$', '$u(t)$','
    Interpreter','latex', 'NumColumns',7);
%         %legend('$C_{o}=30000$','$S$','$E$','$I$','$H$','$C$','$R$', '$D$
    ','Interpreter','latex', 'NumColumns',8);
%         % pCo = plot(T, ICU, 'Color', '#FF0000', 'LineStyle','--'): %,
    '#FF0000', 'Marker', 'v');
%         %pC0.Marker = '--';
% %        set(fig1,'defaultAxesColorOrder',[left_color; right_color]);
```

```
%         hold off
%         Imax_M = max(I0)
%         Cmax_M = max(C0)
%         CmultiOver_M = Cmax_M/Co
%         Dend_T_M = max(DD0)
%         % savefig('nicefig.pdf');
%         %print -depsc myfig.fig
%         %print -depsc epsFig
%
%         fig = figure('Name','ICU-Variants(M-transmission)');
%         left_color = [0 0 0];
%         right_color = [0 0 0];
%         set(fig,'defaultAxesColorOrder',[left_color; right_color]);
%
%         % print(gcf, '-dpdf', 'test.pdf');
%         xlim([0.0 T_max])
%         yyaxis left
%         pC5 = plot(T, C0, 'Color', '#FF0000', 'DisplayName', 'C');
%         hold on
%         pCo = plot(T, ICU0, 'Color', '#FF0000', 'LineStyle','--', '
    DisplayName','C0=30000');
%
%
%         legend('$C_{o}=30000$','$C$','$D$','Interpreter','latex', '
    NumColumns',3);
%         ylabel('ICU Beds in Use');
%         yyaxis right
%         pD7 = plot(T, DD0, 'Color', '#000000', 'DisplayName', 'D');
%         ylabel('Total number of Deaths during Critical period');
%         xlabel('time [d]')
%
%         hold off
%
```

```
%
%         %% Population WITH variant H
%         %normalizing
%         %D = D./Tot_Pop;
%         % S1=Data_0(1,2)
%         % E1 =Data_0(1,3)
%         S0 = Data_H(start:fin,2); E0=Data_H(start:fin,3); I0=Data_H(
    start:fin,4);
%         H0=Data_H(start:fin,5); C0=Data_H(start:fin,6); R0=Data_H(start:
    fin,7);
%         DD0=Data_H(start:fin,8);ICU0=Data_H(start:fin,9);
%
%         %% template plot
%         % plot(T, S, T, E, T, I, T, H, T, C, T, R, T, DD, T, ICU)
%
%         %% template legend
%         % legend('$S$','$E$','$I$','$H$','$C$','$R$', '$D$', '$C_{o
    }=30000$','Interpreter','latex');
%         % legend('$S$','$E$','$I$','$H$','$C$','$R$', '$D$', '$C_{o
    }=30000$','Interpreter','latex', 'NumColumns',2);
%
%         fig_4=figure('Name','ON Population With Variants (H-transmission
    )');
%         left_color = [0 0 0];
%         right_color = [0 0 0];
%         set(fig_4,'defaultAxesColorOrder',[left_color; right_color]);
%         pCo = plot(T, ICU0, 'Color', '#FF0000', 'LineStyle','--', '
    DisplayName','C0=30000');
%         hold on
%     %     pS1=plot(T, S0, 'Color', '#0072BD', 'DisplayName', 'S');
%         pT2=plot(T, E0, 'Color', '#EDB120', 'DisplayName', 'E');
%         pI3 = plot(T, I0, 'Color', '#A2142F', 'DisplayName','I');
%         pH4 = plot(T, H0, 'Color', '#FF00FF', 'DisplayName', 'H');
```

```
%         pC5 = plot(T, C0, 'Color', '#FF0000', 'DisplayName', 'C');
%     %    pR6 = plot(T, R0, 'Color', '#00FF00', 'DisplayName', 'R');
%         pD7 = plot(T, DD0, 'Color', '#000000', 'DisplayName', 'D');
%         %title('Line Plot of Sine and Cosine Between -2\pi and 2\pi')
%         xlabel('time [d]')
%         yyaxis left
%         ylabel('ON Population')
%         %pUt = plot(T, ut, 'Color', '#0072BD', 'DisplayName', 'u(t)');
%         yyaxis right
%         xlim([0.0 T_max])
%         ylim([0.0 1.0])
%         ylabel('Intervernt u(t)')
%         pUt = plot(T, ut, 'Color', '#000000', 'DisplayName', 'u(t)');
%         legend('$C_{o}=30000$','$E$','$I$','$H$','$C$', '$D$', '$u(t)$','
    Interpreter','latex', 'NumColumns',7);
%         %legend('$C_{o}=30000$','$S$','$E$','$I$','$H$','$C$','$R$', '$D$
    ','Interpreter','latex', 'NumColumns',8);
%         % pCo = plot(T, ICU, 'Color', '#FF0000', 'LineStyle','--'): %,
    '#FF0000', 'Marker', 'v');
%         %pC0.Marker = '--';
% %        set(fig1,'defaultAxesColorOrder',[left_color; right_color]);
%         hold off
%         Imax_H = max(I0)
%         Cmax_H = max(C0)
%         CmultiOver_H = Cmax_H/Co
%         Dend_T_H = max(DD0)
%         % savefig('nicefig.pdf');
%         %print -depsc myfig.fig
%         %print -depsc epsFig
%
%         fig = figure('Name','ICU-H-transmission');
%         left_color = [0 0 0];
%         right_color = [0 0 0];
```

```
%         set(fig,'defaultAxesColorOrder',[left_color; right_color]);
%
%         % print(gcf, '-dpdf', 'test.pdf');
%         xlim([0.0 T_max])
%         yyaxis left
%         pC5 = plot(T, C0, 'Color', '#FF0000', 'DisplayName', 'C');
%         hold on
%         pCo = plot(T, ICU0, 'Color', '#FF0000', 'LineStyle','--', '
    DisplayName','C0=30000');
%
%
%         legend('$C_{o}=30000$','$C$','$D$','Interpreter','latex', '
    NumColumns',3);
%         ylabel('ICU Beds in Use');
%         yyaxis right
%         pD7 = plot(T, DD0, 'Color', '#000000', 'DisplayName', 'D');
%         ylabel('Total number of Deaths during Critical period');
%         xlabel('time [d]')
%
%         hold off
%
%

 fclose(ON_V_00);
 fclose(ON_V_L);
 fclose(ON_V_M);
 fclose(ON_V_H);
 fclose(ON_V_00_ut);
 fclose(ON_V_L_ut);
 fclose(ON_V_M_ut);
 fclose(ON_V_H_ut);
```

References

[1] URL: https://docs.idmod.org/projects/emod-hiv/en/latest/model-seir.html (visited on 04/07/2021).

[2] URL: https://www.ontario.ca/page/ontario-demographic-quarterly-highlights-first-quarter-2020 (visited on 05/15/2021).

[3] URL: https://www.nih.gov/news-events/nih-research-matters/lasting-immunity-found-after-recovery-covid-19 (visited on 04/20/2021).

[4] URL: https://ottawa.ca/en/living-ottawa/statistics-and-demographics/current-population-and-household-estimates (visited on 04/20/2021).

[5] URL: https://www.ottawapublichealth.ca/en/reports-research-and-statistics/COVID-19_Vaccination_Dashboard.aspx (visited on 04/20/2021).

[6] URL: https://ourworldindata.org/coronavirus-data (visited on 05/25/2021).

[7] Michael Artin. *Algebra: applications, and algorithms 2nd ed.* Boston: Prentice Hall, 2001.

[8] Canadian Institute for Health Information. "Care in Canadian ICUs: Data Tables". Unknown, after 2015.

[9] Markus Kantner and Thomas Koprucki. "Beyond just "flattening the curve": Optimal control of epidemics with purely non-pharmaceutical interventions". Apr. 2020.

[10] Public Health Ontario. "COVID-19 Real-World Vaccine Effectiveness - What We Know So Far". Mar. 2021.

[11] Public Health Ontario. "COVID-19 Vaccine Uptake in Ontario: December 14, 2020 to May 15, 2021". May 2021.

[12] Lawrence Perko. *Diferential Equations and Dynamical Systems, 3nd ed.* New Your, NY: Springer, 2001.

[13] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical Mathematics 2nd ed.* Berlin: Springer, 2007.

[14] Paul Waltman. *A Second Course in Elementary Differential Equations.* Garden City, NY: Dover, 2004.