Application of Error Control Differential Equations Software to Information Flow Models

By
Evan Lucas-Currie

A Thesis Submitted to

Saint Mary's University, Halifax, Nova Scotia

in Partial Fulfillment of the Requirements for

the Degree of Bachelor of Science Honours

June 01, 2023, Halifax, Nova Scotia

Copyright © Evan Lucas-Currie

Approved:	Dr. Paul Muir (Supervisor)
Approved:	Dr. Yasushi Akiyama (Reader)
Date:	June 01, 2023

Application of Error Control Differential Equations Software to Information Flow Models

By

Evan Lucas-Currie

Abstract

In this thesis, we consider three scientific computing projects. The first project investigates a mathematical model [16] associated with the study of misinformation from the Twitter feed associated with the death of George Floyd. This investigation includes using an error control initial value ODE solver from Python to attempt to verify the results from [16]. The second project investigates mathematical models [19] associated with information diffusion during epidemics. This investigation involves a careful study of how well several of the initial value ODE solvers from Python are able to solve one of the models given the presence of a discontinuity in the definition of the model. Finally, we describe some new work involving the development of a GUI, called G-ODE-PDE, that allows a user to access the suite of error control initial value ODE solvers available in scipy.integrate.solve_ivp and the FORTRAN error control PDE solver, BACOLI, available through the Bacoli_py Python interface.

June 01, 2023

Contents

List of Figures

Acknowledgement

I began the process of writing this thesis with no idea what I was doing. Before this, the longest paper I had written was about 10 pages and I thought it was impossible to write any more than that. However, Dr. Paul Muir showed me how to succeed when I didn't think I could. He took me on as a research assistant over the summer which allowed me to learn how to do research and how to write a thesis; he took the time and effort to help me above and beyond what any other professor has ever done and I will be forever grateful for his impact on my past, present, and future. I'm an older student with a family and Dr. Muir was always understanding and able to motivate me to keep pushing through the process over the last year. He has taught me everything I know about Numerical Analysis, ODEs, and PDEs. I also want to take a moment to thank Dr. Yashushi Akiyama for taking time out of their busy schedule to read the thesis. I would like to also thank the strong women who have been a huge impact on my life – my grandmother and my wife. My wife pushed me to go to university to pursue my dreams. When my grandmother passed away, she gave up her life to raise me and I know she would be proud of what I have achieved so far.

1 Introduction

In this thesis, we will consider error control software for the numerical solution of the initial value ordinary differential equations (ODEs) and time-dependent one spatial dimension, partial differential equations (PDEs). Error control means that, in addition to computing an approximate solution to the ODEs or PDEs, the software also computes an estimate of the error of the approximate solution and adapts the computation until it obtains an approximate solution for which the corresponding error estimate satisfies a user-defined tolerance.

Our first investigation will involve an examination of the paper [16] that looks at an analysis of the spread of misinformation associated with the Twitter feed #DCBlackout which was associated with the riots that occurred after the murder of George Floyd [7]. The authors use a mathematical model to represent the spread of misinformation through various populations. This mathematical model is represented by a system of ODEs. The model is sufficiently complicated that it can not be solved by hand and the authors use numerical software to obtain a solution. In this thesis we will examine the questions that arise in this paper including directly accessing the data available on Twitter and then using a least squares approach to fit the parameters of the ODE model so that the solution to the model gives an optimal fit to the data in a least squares sense. We will use

state-of-the-art error control initial value ODE software to compute solutions to the model. We then compare our results with those given in the paper.

The second investigation involves examining the work undertaken in the paper [19], where the authors consider the spread of information diffusion using epidemiological models. The amount of information in these models is based on how many people are currently well-informed and how many people are poorly informed. These terms will be more carefully defined later in the thesis. The investigation the authors undertook involves a mathematical model for which is it not possible to obtain a close form solution. Therefore, the authors use numerical software to compute a solution. In this thesis we will examine the questions that are considered in the paper and in particular apply error control initial value ODE software to solve the initial value model that arises. An interesting aspect of this work is that the model contains a discontinuous parameter which leads to discontinuities in the definition of the right hand side of the ODE system. Such problems are well known to be challenging for standard initial value solvers. In order to investigate the performance of a number standard initial value solvers when faced with a initial value ODE that has a discontinuity, we apply all of the ODE solvers that are available in the Python Scipy [12] library and investigate how they perform when applied to this problem.

The third aspect of our research is concerned with the development of a graphical user interface (GUI), called G-ODE-PDE, for error control ODE solvers and an error control PDE solver. The initial value solvers which can be accessed via the GUI are the suite of ODE solvers provided by scipy.integrate.solve_ivp [12]. The PDE solver that can be accessed via the GUI is a Python interface, bacoli_py [3], to a FORTRAN PDE solver called BACOLI [2].

Throughout the first two investigations, discussed in Chapter 2 and Chapter 3, we use solvers available through the suite from Scipy.integrate.solve_ivp. These solvers are also all available through G-ODE-PDE. These are as follows:

- BDF - This initial value ODE solver is based on a family of backward differentiation formulas (BDFs). These formulas are implicit multi-step methods. Such methods can be used to take a sequence of steps from the beginning of the time domain to the end. A solution approximation is computed at the end of each time step. An implicit method uses the current and past values of the solution to compute an estimate of the value of the solution at the end of the current time step. Implicit methods are often used for solving differential equations that are stiff; such ODEs have solutions that vary rapidly over different time scales. In Scipy, the BDF solver dynamically use BDF methods of 1 to 5 orders of accuracy [12]. When the error of the approxi-

mate solution on a single time step is $O(h^{p+1})$ for some positive integer p, the numerical method is said to be of order p.

- RK45 - This solver is based on a Runge-Kutta pair of order 5(4) and it is an explicit method used to update a solution at each time step based on the approximate value of the solution at the beginning of the step and several other intermediate solution approximations within the step. We say it is explicit because the calculation of the next solution approximation depends only on previously available solution information. The notation "5(4)" that appears above has the following meaning. The 5^{th} order Runge-Kutta method provides an accurate estimate of the solution at the end of the time step. The solver has a built-in 4^{th} order Runge-Kutta method that is used to estimate the error of the 5^{th} order method. RK45 is often not the best choice for stiff systems, as it can require much smaller step sizes in order to get the required accuracy, which can vastly increase computation time. The steps are dynamically adjusted based on the error estimate and the tolerance in order to obtain the required accuracy.

- RK23 - This solver is based on a Runge-Kutta pair of order 3(2), and is similar to the RK45 method described above.

- Radau - The Radau solver is used to solve stiff ODEs [4]. It is part of the Radau IIA family of methods of order 5, and is an implicit Runge-Kutta method. It includes an error estimate based on a third-order Runge-Kutta method. The solver chooses the step size so that the estimate error is less than the user provided tolerance [12].

- DOP853 - The DOP853 solver is based on an explicit Runge-Kutta method of order 8 [12]. Overall, it is similar to the previously mentioned Runge-Kutta methods.

- LSODA - The Livermore Solver for Ordinary Differential Equations with Automatic Method Switching (LSODA) is a numerical method for solving initial value ODEs. The Scipy implementation for LSODA is a wrapper for the FORTRAN solver called LSODA, which provides users with the speed associated with FORTRAN within a Python environment. The software is based on two families of methods, the Adams methods for non-stiff ODEs [12], and the BDF methods for stiff ODEs. This algorithm is distinguished by its ability to switch between the two families of methods based on detection of a stiff problem. LSODA generally begins with a non-stiff Adams method, which involves lower cost per step. If the problem becomes stiff,

the method will switch to the BDF family of methods, preserving performance and accuracy.

As mentioned above the G-ODE-PDE GUI also provides access through the bacoli_py interface, to BACOLI [2]. In BACOLI, the spatial domain is discretized using B-spline Gaussian collocation. This means that the numerical solution is represented in terms of a B-spline basis [1]. In particular, the solution is a sum of B-spline basis functions defined over the spatial domain with unknown time-dependent coefficients. Application of the collocation method means that the approximate solution is required to exactly satisfy the PDE at certain points within each sub-interval of a mesh which partitions the spatial domain. The points on each sub-interval where the approximate solution is required to satisfy the PDE are called collocation points, and in BACOLI they are chosen to be images of Gauss points [5] on each sub-interval. After the collocation process is applied, the PDEs are approximated by a system of time dependent ODEs. These together with the boundary conditions represent a system of time-dependent differential algebraic equations (DAEs). In BACOLI, the DAE system is solved using the DAE solver called DASSL [18]. DASSL computes error controlled approximations to the B-spline coefficients using a family of BDF methods. Once the B-spline coefficients are returned from DASSL, these can be used together with the B-Spline

basis functions to obtain the approximate solution. An important feature of BACOLI is that, in addition to computing an approximate solution, it also computes an estimate of the error of that solution. BACOLI has two distinct error estimators; one is based on sampling the collocation solution at a special set of points and then constructing an interpolant that is one order of accuracy higher than the collocation solution. The collocation solution has an error that is O(h^{p+1}) where h is the subinterval size and p is the degree of the B-spline basis functions. The interpolant, because it makes use of special points where the approximate solution is somewhat more accurate, has an error that is O(h^{p+2}); the difference between these two gives an error estimate for the collocation solution. The second type of error estimate is based on constructing an interpolant that is one order lower than the collocation solution; this means that the error of this lower order interpolant (LOI) is O(h^{p-1}). Again, the difference between this interpolant and the collocation solution gives the error estimate.

In either case, the error estimate is used to determine if the solution at the end of each DASSL time step is acceptable; this means the error estimate needs to be less than a user supplied tolerance. If the error estimate is larger than the tolerance, then BACOLI generates a new spatial mesh which is designed to use a sufficient number of points that are clustered in the part of the spatial domain

where the error estimate is the largest. The result of the BACOLI computation is a numerical solution to the PDE for which both the time and spatial errors are controlled adaptively by DASSL and BACOLI respectively.

The thesis is organized as follows. In Chapter 2, we consider the investigation of the spread of misinformation associated with riots following the death of George Floyd. Chapter 3 discusses our investigation of the mathematical model that involves information diffusion associated with epidemics. Chapter 4 is an overview of the G-ODE-PDE GUI. The thesis concludes in Chapter 5 with our summary, conclusions, and suggestions for future work.

2 An Epidemiological Model for the Spread of Misinformation

In this chapter, we consider an ODE-based epidemiological model for the spread of misinformation associated with the real-world Twitter data that has been scraped off the Twitter social media website under the hashtag "DCBlackout" [7]; this data is associated with the misinformation spread during the riots following George Floyd's death [7]. The ODE model includes several parameters; a data fitting algorithm is employed to estimate the parameter values so that the resultant solution

to the ODE system gives an optimal fit to the Twitter data in a least squares sense. In this chapter, we will review the ODE model considered in [16], briefly cover the numerical methods employed by the authors, and then present our results from our own treatment of the model.

2.1 The ODE Model

In [16], the authors use a SEIZ (Susceptible-Exposed-Infected-Skeptic) epidemiological ODE model in an attempt to model the flow of misinformation associated with the hashtag "DCBlackout" on the social media platform Twitter [16]. "Susceptible" in this model refers to individuals who can become either exposed, infected, or skeptical with respect to the misinformation. Susceptible individuals have not yet encountered the misinformation. "Exposed" represents the individuals who have seen a given piece of misinformation but have not yet made a decision to engage with the information. "Infected" represents the individuals who have interacted with a piece of misinformation, either through commenting on it or posting a tweet with a hashtag "DCBlackout". "Skeptic" represents those who do not engage with the misinformation, resulting in them not further spreading the misinformation. The authors state this is the first time this type of model has been applied to misinformation rather than disinformation. The authors define

misinformation as the spread of information that is not known to be false, meaning that the individuals spreading the misinformation believe it to indeed be true. Disinformation is information that is spread by those who know what they are spreading is indeed incorrect [16]. Disinformation therefore needs intent while misinformation does not.

2.2 Model Definition

In this section, we give a careful review of the ODE model from [16]. The ODE system has the form (S corresponds to Susceptible individuals, E corresponds to Exposed individuals, I corresponds to Infected individuals, and Z corresponds to people who are Skeptics)

$$\begin{aligned}
\tfrac{dS}{dt} &= -\beta S\tfrac{I}{N} - bS\tfrac{Z}{N},\\
\tfrac{dE}{dt} &= \bar{p}\beta S\tfrac{I}{N} + \bar{\ell}bS\tfrac{Z}{N} - \rho E\tfrac{I}{N} - \epsilon E,\\
\tfrac{dI}{dt} &= p\beta S\tfrac{I}{N} + \rho E\tfrac{I}{N} + \epsilon E,\\
\tfrac{dZ}{dt} &= \ell bS\tfrac{Z}{N}.
\end{aligned}$$

The parameters appearing in this model are as follows:

β	Contact rate between S and I
b	Contact rate between S and Z
ρ	Contact rate between E and I
p	Probability of S becoming I given Contact with I
$\bar{p} \to (p-1)$	Probability of S becoming E given Contact with I
ϵ	Transition rate of E becoming I (incubation rate)
ℓ	Probability of S becoming Z given contact with I
$\bar{\ell} \to (\ell - 1)$	Probability of S becoming E given contact with I

The first ODE

$$\frac{dS}{dt} = -\beta S\frac{I}{N} - bS\frac{Z}{N}, \tag{1}$$

models the change in the number of Susceptible individuals. In the first term, $-\beta S\frac{I}{N}$, $\beta \in [0,1]$ represents the contact rate, S corresponds to the total number of Susceptible individuals, and $\frac{I}{N}$ is the number of infected individuals relative to the total population, N. The parameter β controls the rate at which the Susceptible population will decrease given contact with the Infected population. Regarding the $-bS\frac{Z}{N}$ term in (1), the parameter $b \in [0,1]$, represents the rate at which the Skeptics and Susceptible populations come into contact. We note that we only subtract from the Susceptible population, i.e., both terms on the right-hand side

of (1) are negative, and therefore should not see an increase in the number of Susceptible individuals.

The second ODE is concerned with the change in the number of Exposed individuals, and has the form,

$$\frac{dE}{dt} = \bar{p}\beta S\frac{I}{N} + \bar{\ell}bS\frac{Z}{N} - \rho E\frac{I}{N} - \epsilon E. \tag{2}$$

We consider the first term $\bar{p}\beta S\frac{I}{N}$, which is related to the first term from (1), $\beta S\frac{I}{N}$, and includes an additional parameter, $\bar{p}$. The $\bar{p}$ parameter is associated with the probability that a portion of the $\beta S\frac{I}{N}$ term will become part of the Exposed population. This term is positive, corresponding to an increase in the Exposed population. Moving on to the next term in (2), $\bar{\ell}bS\frac{Z}{N}$, we note that it corresponds to the second term from (1), $bS\frac{Z}{N}$, and focuses on the contact between the Skeptics and the Susceptible population. In this case, the coefficient $\bar{\ell}$ represents the probability of a proportion of these individuals becoming part of the Exposed population. Regarding terms correspond to a decrease in the Exposed population, we begin with the term, $\rho E\frac{I}{N}$, where $\rho \in [0, 1]$. This serves as another contact term, in which the Exposed are in contact with the Infected. Lastly, we have an incubation term, $-\epsilon E$, which takes into consideration the fact that an Exposed individual, after a certain amount of time, may become Infected. The parameter, ϵ,

gives the rate at which Exposed people leave the Exposed population after having contact with Infected people.

The third ODE is

$$\frac{dI}{dt} = p\beta S\frac{I}{N} + \rho E\frac{I}{N} + \epsilon E. \tag{3}$$

This ODE describes how the number of Infected can change. We observe that all the terms of the Infected component are positive, prohibiting any decrease in the number of Infected People. The term $p\beta S\frac{I}{N}$, corresponds to the portion of Susceptible people who directly become Infected rather than simply becoming Exposed. The next term, $\rho E\frac{I}{N}$, corresponds to Exposed people who become Infected through contact with Infected people. Lastly, ϵE represents the incubation period leading to Exposed individuals become Infected.

The last ODE is

$$\frac{dZ}{dt} = \ell bS\frac{Z}{N}. \tag{4}$$

This ODE considers the change in the Skeptics population. Here we only have the term, $\ell bS\frac{Z}{N}$, which is the counterpart to the term, $\bar{\ell}bS\frac{Z}{N}$, that appears in (2), and which is associated with the Exposed component of the population. This allows ℓ to directly control the rate at which people flow from Susceptible into Skeptics.

We note that the term is a positive term which means that the total number of Skeptics can only grow.

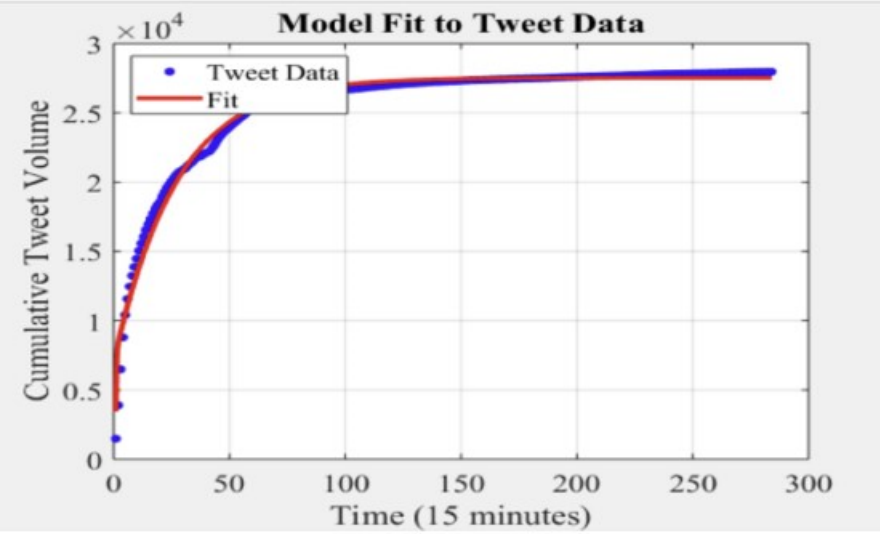


Figure 2.1: Discrete Twitter data and the fitted solution for the Infected parameter from [16].

Regarding data collection, the authors of [16] make use of a Twitter scraping tool named Twint [13]; this tool allows the user to grab large volumes of Twitter data. See Fig. 2.1 for a plot of the data collected in [16]. The blue points in Fig. 2.1 represent discrete values, with each point being the volume of tweets within a 15-minute interval. Every 4 points on the horizontal axis of the graph is the equivalent to 1 hour. We can see that most of the growth is the number of

tweets occurs during in the first 12.5 hours marked by 50 on the horizontal axis.

The red line in Fig. 2.1 represents a plot of Infected population over time, as obtained from the numerical solution of the ODE model, where the model parameters were fitted so that the Infected solution component agrees with the Twitter data optimally, in a least squares sense. That is, the authors determined the values for the parameters of the ODE model using a least squares algorithm to minimize this difference between the Twitter data and the Infected solution component of the ODE model. The software [19] used for the least squares computation was MATLAB's nonlinear least squares (lsqnonlin) function. Since the authors do not mention the specific algorithm they used, we assume they used the default algorithm which is an algorithm called TRF (trust region reflective) [10]. In [16], in order to solve the ODE model, the authors use the MATLAB ode45 function, which is based on a Runge-Kutta formula pair [11].

2.3 Our Investigation

Since the authors mentioned the tool they utilized to gather the information, we were also able to collect the data using the same tool. This approach presented a challenge as Twitter is an ever-changing platform and users can delete their tweets. The continuous updates on Twitter have resulted in the tool becoming

partially dysfunctional; we found that it does not collect all of the tweets and often would miss data. A temporary solution was implemented to allow us to collect the expected volume of tweets.

There are two differences between the data we collected and what was collected in [16]. The data we collected did not extend as far as the data collected in [16] because the data collection process started to become exponentially slower. We opted on stopping the collection because we felt we had captured the most important subset of the data, that being the data that exhibit extreme growth. Also the authors do not include the first 4 hours and 15 minutes of Tweets to the #dcblackout hashtag. We decided to include this data in our study since it represented significant new behavior compared to the original data set.

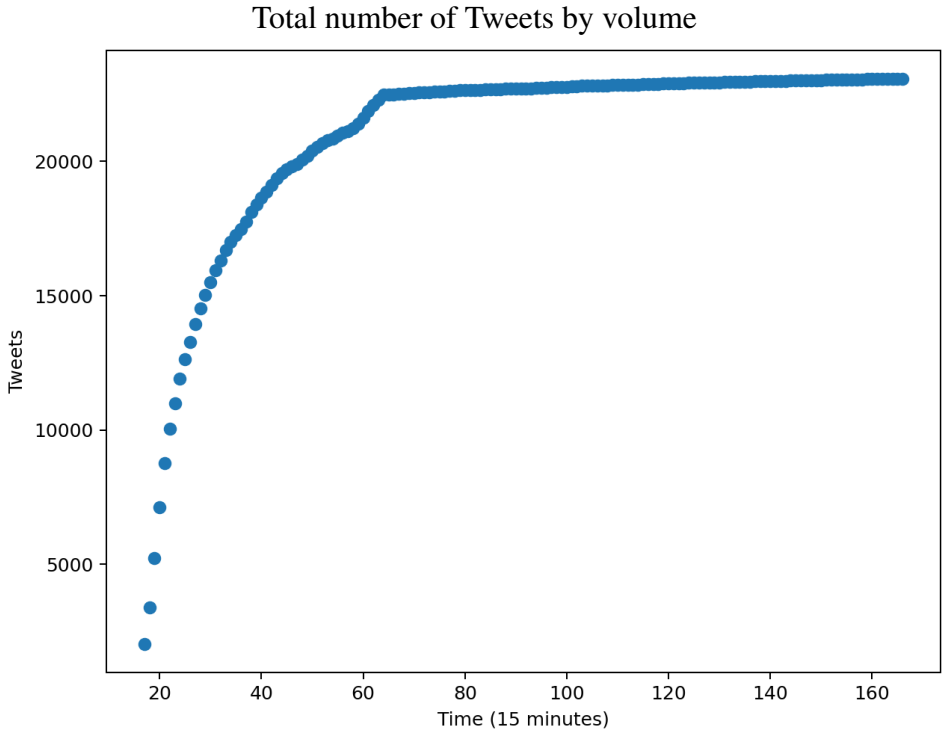


Figure 2.2: Tweets by volume as collected in our study (excluding the first 4 hours and 15 minutes.)

Recall that Fig. 2.1 (the blue dots) shows the Twitter data collected in [16]. Fig. 2.2 shows the data that we collected after the first 4 hours and 15 minutes. We note that, except for the fact that the data in Fig. 2.1 was collected for a larger period of time (almost 300 15-minute intervals) compared to our data which was collected to about 160 15-minute intervals, both figures are almost identical. To verify the results from [16], we employed the least-squares

function from the Python SciPy.optimise package. The algorithm used for the least-squares fitting was the Trust-Region-Reflective algorithm; this is the algorithm that is also available in Matlab. The initial values we used for input to the least squares function for the parameters were the values published in [16], namely $\beta = 4.3713, \rho = 1.3833 \cdot 10^{-6}, \epsilon = 0.0373, b = 8.1967, p = 0.7905$, and $\ell = 0.8161$. Since the initial values for the Susceptible, Exposed, Infected, and Skeptic (SEIZ) compartments were not explicitly stated in [16], we visually estimated these values based on the figures that appear in [16]. We choose the initial values to be S = 61000, E = 1000, I = 1000, and Z = 1000. Our N is smaller than what was used in [16] due to the fact that we did not include as much data. Our N value is 64,000 while the N values used in [16] is about 90,000.

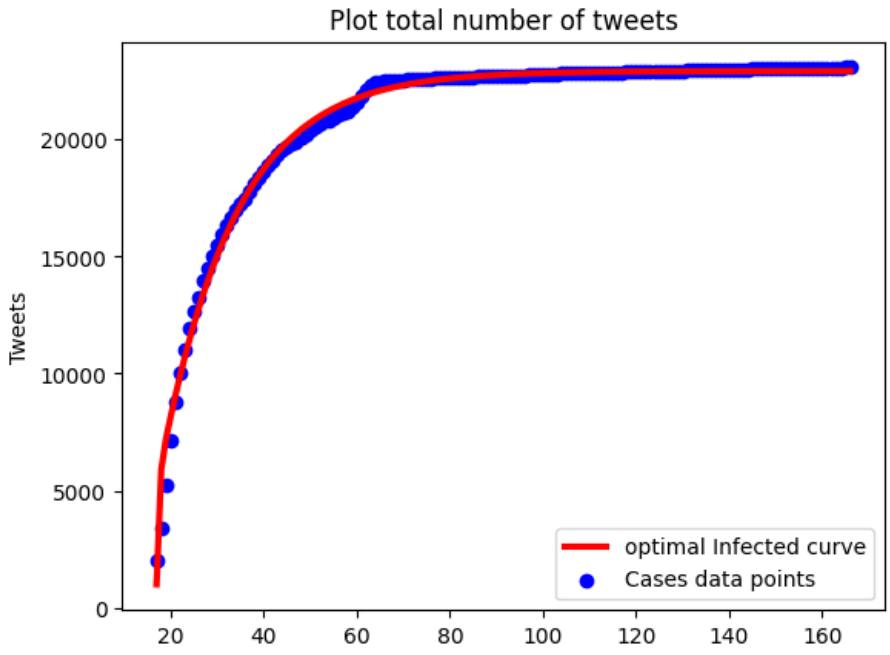


Figure 2.3: Twitter data and the corresponding Infected solution obtained from the parameter fitted ODE model (excluding the first 4 hours and 15 minutes).

For the solution of the ODE model, we choose to utilize the scipy.integrate package solve-ivp class, making use of its LSODA solver, and choosing the relative and absolute tolerances to be 10^{-12}. An examination of Fig. 2.3 reveals that our Infected solution is similar to what can be seen in Fig. 2.1. Comparing our fitted parameter values with those from [16], we note that we obtained parameter values that are somewhat different from those obtained in [16]. Our parameter values are $\beta = 4.1953, \rho = 0.0421, \epsilon = 0.0577, b = 8.1935, p = 0.6231, \ell = 0.7390$.

The difference in parameter values is to be anticipated given that the data sets differ. Notably, the greatest discrepancy is observed in parameter p, which reflects an increased contact between the Exposed and Infected populations. This discrepancy is reasonable, given that our data was obtained over a smaller time interval.

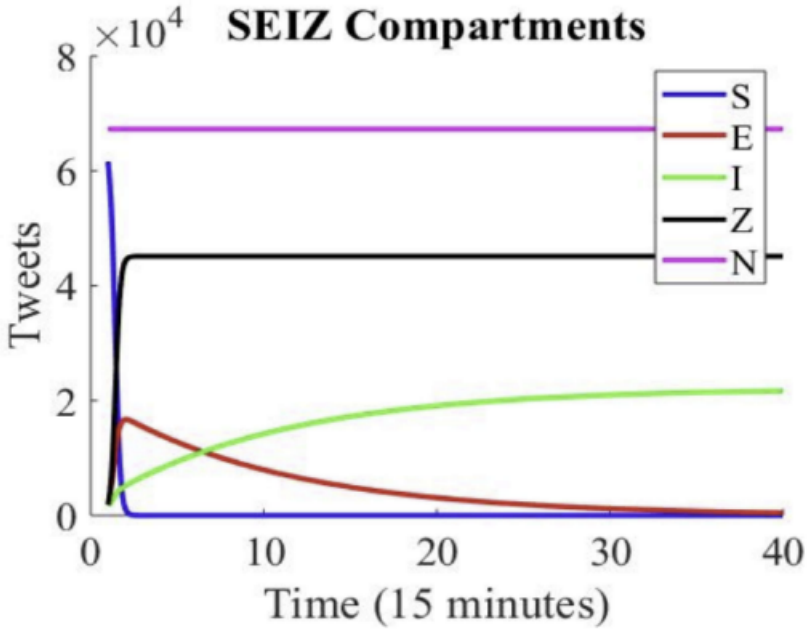


Figure 2.4: Calculated SEIZ values in [16]. This graph shows a rapid decrease in the Susceptible population and rapid increases in the Exposed and Skeptics populations. Afterwards, the Exposed population slowly decreases. The Infected population slowly increases over time.

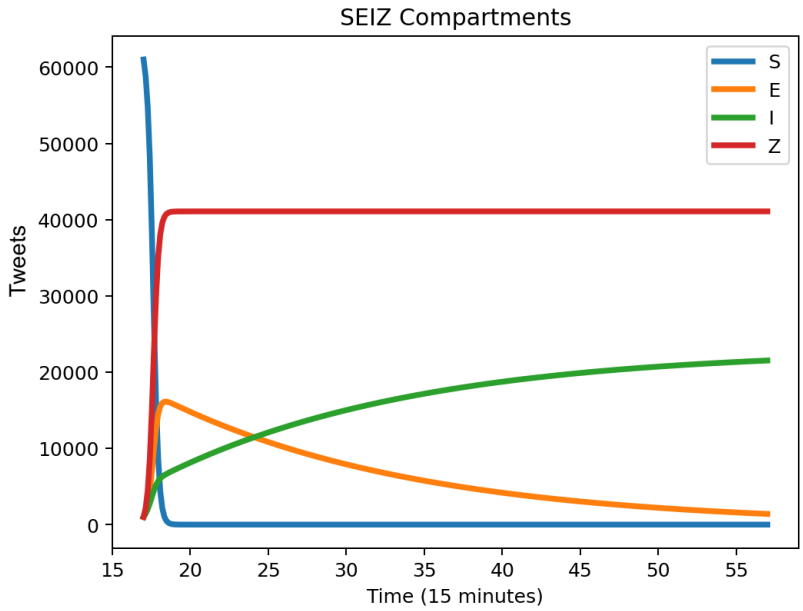


Figure 2.5: Our calculated SEIZ values. These agree well with what is seen in Figure 2.4. This graph shows a rapid decrease in the Susceptible population and rapid increases in the Exposed and Skeptics populations. Afterwards, the Exposed population slowly decreases. The Infected population slowly increases over time.

Fig. 2.4 and Fig. 2.5 show the plots for all solution components as presented in [16] and from our own computations; we obtained good agreement between the results from [16] and our results. The authors reported a relative error of 0.019 between their data and their fitted Infected population, which we were able to improve to 0.016, although this comparison is not entirely relevant as the data sets differ.

In Fig. 2.6, we show the full set of data we collected; what is significant about this data set is that it includes the initial 4 hours and 15 minutes of tweets sent to #DCBlackout.

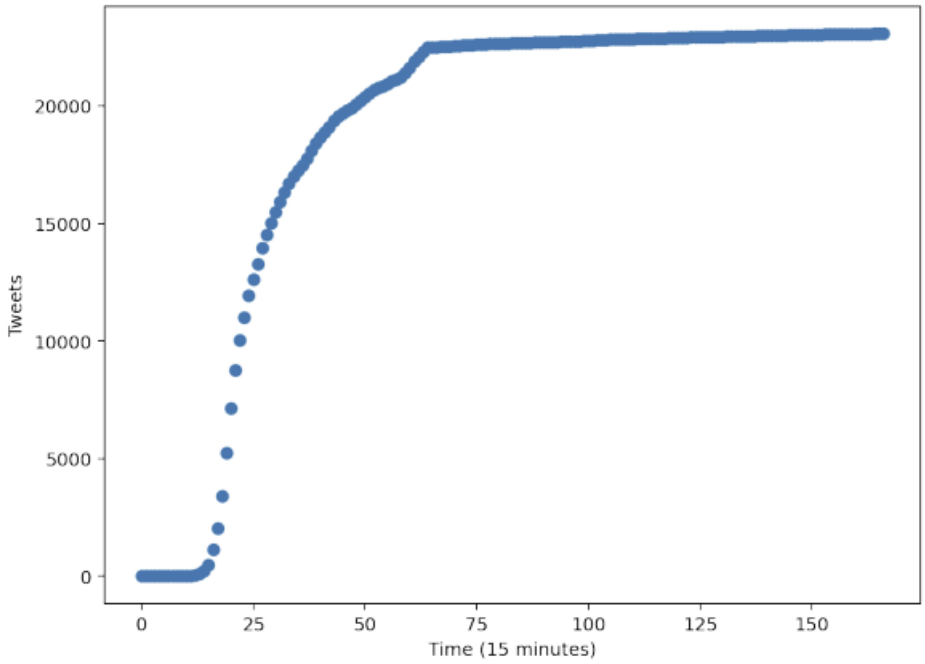


Figure 2.6: Total number of Tweets by volume including the first 4 hours and 15 minutes.

Fig. 2.7 shows the result of a computation we performed to attempt to fit the parameters of the Infected solution component of the ODE model using all the data from Fig. 2.6.

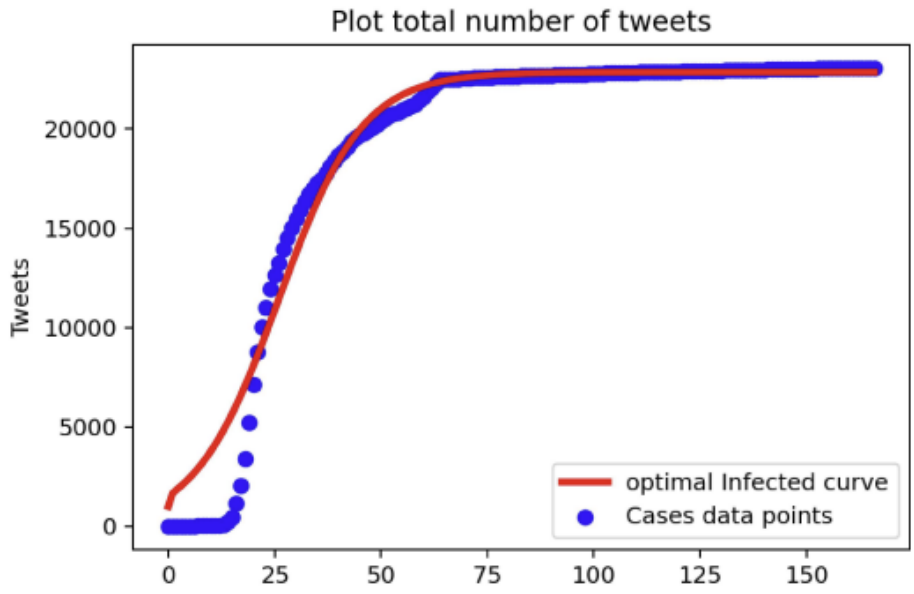


Figure 2.7: Curve fit for extended data set.

Analysis of Fig. 2.7 reveals that it is more difficult to fit the model to this data set; the error which is 0.065 is about four times greater than we previously achieved when we did not consider the initial interval.

Fig. 2.8 shows all four solution components of the ODE model that was fitted to the data from Fig. 2.6.

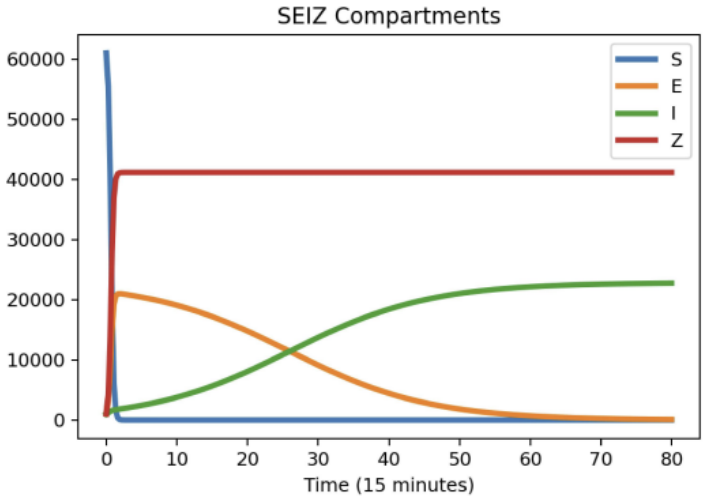


Figure 2.8: Solution for data from ODE System fitted to data shown in Fig. 2.6.

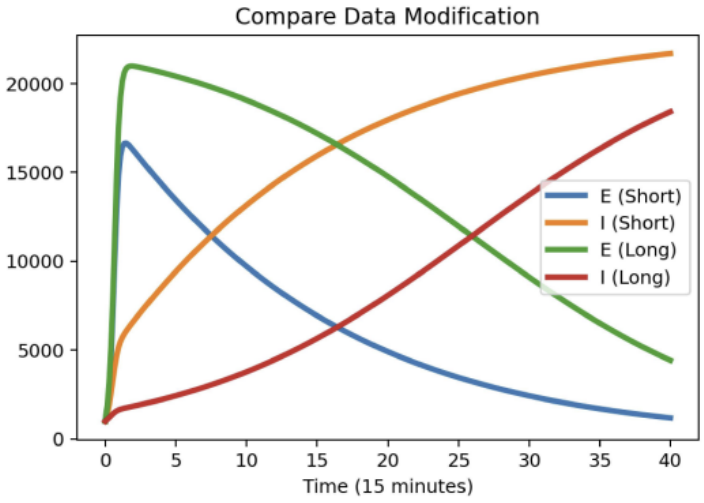


Figure 2.9: Exposed and Infected solution components corresponding to Extended (Long) vs Truncated data (Short).

By examining Fig. 2.8, and comparing it with Fig. 2.1, it is evident that the Exposed and Infected solution components are heavily impacted by the use of the data from the first 4 hours and 15 minutes, while the remaining compartments Susceptible and Skeptics are only marginally affected. A comparison between the Exposed and Infected solution components from our original data set and those associated with the extended data set that includes the data from the first 4 hours and 15 minutes is presented in Fig. 2.9. It shows that it is a much longer period of deliberation before users post information in the case where the data set is extended.

2.4 Summary

We found that when fitting parameter values to our ODE model based on data collected from a social media website, the point at which one starts the data collection can have a significant impact on the solutions that are obtained. We opted for including more data throughout our computations to show the impact that this can have. In our study, when the more limited data set is considered, our solutions align well with the results from [16]; however, our extended data set reveals discrepancies regarding the time at which the number of Exposed individuals falls below the number of Infected individuals. This then leads to the infection rate

growing slower than what was reported in [16]. Truncating initial data implies that people are exposed for less time, while in reality, their exposure time is somewhat larger.

2.5 Future Work

To extend our research, more work could be done conducting experiments using this model on dynamic data sets incorporating fluctuations in the spread of misinformation, rather than a single substantial increase. In other words, we feel that it would be interesting to investigate applications where there is greater variation within the data. This would allow us to better analyze waves of the spread of misinformation, rather than only an initial outbreak. Also, future work could include fixing a tool like Twint which would allow researchers more accessibility in being able to pull down Twitter data, which could improve the reliability of the results.

3 Coupling Dynamics with Information Diffusion in an Epidemiological Model

In this chapter, we examine the numerical results of a study integrating the coupling dynamics of an epidemiological ODE model with information diffusion.

The total amount of information is defined to be the total number of aware individuals in the population. To numerically analyze the four states of information S_+ (Susceptible aware), S_- (Susceptible unaware), I_+ (Infected aware), I_- (Infected unaware), with emphasis on how the information impacts the system. An interesting aspect of this mathematical model is that there is a discontinuity present in the ODEs, which arises due to the attempt to simulate behaviors such as the introduction of precautionary measures. This model is applied to two different diseases; the results show that the re-emergence of the diseases can be higher or lower than the original infection rate, depending on various factors associated with the model.

3.1 Introduction

The coupling dynamics of epidemic spreading and epidemic information diffusion on complex networks was studied in [19]. The authors used three methods throughout their paper. The Mean-Field method involves a set of ODEs and will be the main focus of this chapter. The second method is called Pairwise Analysis of the Mean-Field ODEs, which we will call the pairwise method; this method leads to a system of 14 ODEs. The authors also consider a simulation which they refer to as the Ground Truth Values method, which we will call the simulation

method.

Regarding the Mean-Field method, the authors employ a set of ODEs to model four population states: Susceptible aware (S_+), Susceptible unaware (S_-), Infected aware (I_+), and Infected unaware (I_-), with emphasis placed on the amount of information present within the system. The model depends on a parameter, α, which represents the level of information diffusion. The authors propose two thresholds, I_{High}, (when this threshold is reached, α is set to 0.8) and I_{Low}, (when this threshold is reached, α is set to 0.3); these are defined as upper and lower percentages of information diffusion within the total population, respectively. When the information content reaches either of these bounds, the value of the parameter α is altered to be either the higher or lower value, depending on which threshold has been reached. This simulates behaviors where individuals periodically take precautionary measures, such as wearing face masks and practicing social distancing when information about an epidemic is high, and then when they relax these measures when information about the epidemic is low. The authors applied this model to two different diseases, H7N9 and Dengue fever, and demonstrated that depending on the values used for α and the upper and lower thresholds, the re-emergence of the diseases can be at either higher or lower levels than the original infection rate.

3.2 Model Analysis

The Mean-Field model that the authors consider is a system consisting of four ODEs, involving S_+, S_-, I_+, I_-; in these ODEs, the state variables are denoted with [*]; the ODE system has the following form.

$$\frac{d[S_-]}{dt} = -k\beta[I_-]\frac{[S_-]}{N} - k\sigma_I\beta[I_+]\frac{[S_-]}{N} - k\alpha([S_+]+[I_+])\frac{[S_-]}{N} + \lambda[S_+] + \gamma[I_-],$$
$$\frac{d[S_+]}{dt} = -k\sigma_s\beta[I_-]\frac{[S_+]}{N} - k\sigma_s\sigma_I\beta[I_+]\frac{[S_+]}{N} + k\alpha([S_+]+[I_+])\frac{[S_-]}{N} - \lambda[S_+] + \epsilon\gamma[I_+],$$
$$\frac{d[I_-]}{dt} = k\beta[I_-]\frac{[S_-]}{N} + k\sigma_I[I_+]\frac{[S_-]}{N} - k\alpha([S_+]+[I_+])\frac{[S_-]}{N} + \delta\lambda[I_+] - \gamma[I_-],$$
$$\frac{d[I_+]}{dt} = k\sigma_s\beta[I_-]\frac{[S_+]}{N} + k\sigma_s\sigma_I\beta[I_+]\frac{[S_+]}{N} + k\alpha([S_+]+[I_+])\frac{[S_-]}{N} - \delta\lambda[I_+] - \epsilon\gamma[I_+].$$

The parameters appearing in this system are

β	The probability that S_- is infected via the I_- neighbor $(S_-I_- + I_-I_-)$
$\sigma_s\beta$	The probability that S_+ is infected via the I_- neighbor $(S_+I_- + I_+I_-)$
$\sigma_I\beta$	The probability that S_- is infected via the I_+ neighbor $(S_-I_+ + I_-I_+)$
$\sigma_{SI}\beta$	The probability that S_+ is infected via the I_+ neighbor $(S_+I_+ + I_+I_+)$

γ The probability that I_- recover to S_-

$\epsilon\gamma$ The probability that I_+ recover to S_+

α Information transmission rate

λ Information fading rate $(S_+ \to S_-)$

$\delta\lambda$ Information fading rate $(I_+ \to I_-)$

k Average degree of the network

Fig. 3.1, from [19], illustrates how information flows among the state variables. From this figure, we can see, for example, that α controls the flow of information from the unaware (S_-) to aware (S_+). The information flow graph defined in [19] is as follows:

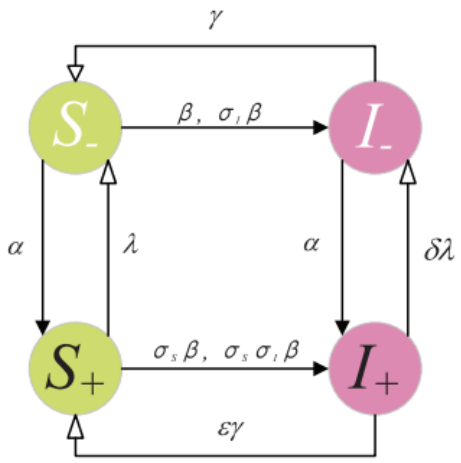


Figure 3.1: Information Flow Graph [19].

We now discuss the first ODE. This ODE describes the rate of change of S_- and has the form,

$$\begin{aligned}\frac{d[S_-]}{dt} =& - k\beta[I_-]\frac{[S_-]}{N} - k\sigma_I\beta[I_+]\frac{[S_-]}{N} \\ & - k\alpha([S_+] + [I_+])\frac{[S_-]}{N} + \lambda[S_+] + \gamma[I_-].\end{aligned} \tag{5}$$

Analyzing equation (5) term-by-term, we observe that the first term, $-k\beta[I_-]\frac{[S_-]}{N}$, where k is the average degree of the network and β is the contact rate for $[I_-]\frac{[S_-]}{N}$, represents the probability of an unaware infected individual coming into contact with an unaware susceptible individual and becoming infected. In the second term, $-k\sigma_I\beta[I_+]\frac{[S_-]}{N}$, we note two differences from the first term: the introduction of σ_I and the change from $[I_-]$ to $[I_+]$. Together, $\sigma_I\beta$ gives the contact rate for aware infected individuals coming into contact with unaware susceptible individuals. Consequently, both β and $\sigma_I\beta$ should flow into the infected population. From Fig. 3.1, we see that this is indeed the case. The third term $-k\alpha([S_+] + [I_+])\frac{[S_-]}{N}$ reflects the information diffusion process, where α captures the flow of information between the aware and unaware since $([S_+] + [I_+])$ represents all the information for these two states. $\frac{[S_-]}{N}$ makes this subtraction term proportional to the total population. Finally, the $\lambda[S_+]$ term can be seen to directly influence the number of aware individuals becoming unaware, while the $\gamma[I_-]$ term reflects the opposite, with individuals moving from unaware to aware.

The second ODE describes the rate of change of S_+ and has the form,

$$\begin{aligned}\frac{d[S_+]}{dt} =& - k\sigma_s\beta[I_-]\frac{[S_+]}{N} - k\sigma_s\sigma_I\beta[I_+]\frac{[S_+]}{N} \\ &+ k\alpha([S_+] + [I_+])\frac{[S_-]}{N} - \lambda[S_+] + \epsilon\gamma[I_+].\end{aligned} \tag{6}$$

The first term, $-k\sigma_s\beta[I_-]\frac{[S_+]}{N}$, represents a decrease in the S_+ population proportional to the fraction of infected unaware individuals relative to the number of the susceptible aware individuals; its influence is controlled by the factor $\sigma_s\beta$. The second term, $-k\sigma_s\sigma_I\beta[I_+]\frac{[S_+]}{N}$, is analogous, but with a substitution of $[I_-] \rightarrow [I_+]$ and the inclusion of an additional factor σ_S. This flow is consistent with what is shown in Fig. 3.1. The third term, $k\alpha([S_+] + [I_+])\frac{[S_-]}{N}$, which appears as a negative term in equation (5), is added to the right hand side of equation (6). The fourth term, $-\lambda[S_+]$, represents the flow of $[S_+]$ to $[S_-]$; this is the fading rate that represents, over time, aware people will become unaware. Finally, the term, $\epsilon\gamma[I_+]$, reflects the probability of infected aware individuals changing to become susceptible aware individuals, as shown in Fig. 3.1.

The third ODE describes the rate of change of I_- and has the form,

$$\begin{aligned}\frac{d[I_-]}{dt} =& k\beta[I_-]\frac{[S_-]}{N} + k\sigma_I[I_+]\frac{[S_-]}{N} \\ &- k\alpha([S_+] + [I_+])\frac{[S_-]}{N} + \delta\lambda[I_+] - \gamma[I_-].\end{aligned} \tag{7}$$

In equation (7), the first term, $k\beta[I_-]\frac{[S_-]}{N}$, characterizes the flow of individuals between the infected unaware and susceptible unaware populations. This is

followed by the term, $k\sigma_I[I_+]\frac{[S_-]}{N}$, which appears as a negative term the equation (5) and a positive term in equation (7). The information diffusion term, $-k\alpha([S_+]+[I_+])\frac{[S_-]}{N}$, is subtracted from the infected unaware population. Lastly, we consider the terms $\delta\lambda[I_+]$ and $-\gamma[I_-]$; δ is included along with λ to modify the fading rate from the infected aware to the infected unaware population. The latter term, $-\gamma[I_-]$, corresponds to the transition of individuals from the infected unaware to the susceptible unaware population.

The fourth ODE describes the rate of change of I_+ and has the form,

$$\begin{aligned}\frac{d[I_+]}{dt} =& k\sigma_s\beta[I_-]\frac{[S_+]}{N} + k\sigma_s\sigma_I\beta[I_+]\frac{[S_+]}{N} \\ &+ k\alpha([S_+]+[I_+])\frac{[S_-]}{N} - \delta\lambda[I_+] - \epsilon\gamma[I_+].\end{aligned} \tag{8}$$

The term, $k\sigma_s\beta[I_-]\frac{[S_+]}{N}$, corresponds to the flow between infected unaware individuals and aware susceptible individuals. The factor, $\sigma_s\sigma_I\beta$, is an additive flow factor that accounts for the flow of infected and relative susceptible aware individuals, $[I_+]\frac{[S_+]}{N}$. The term, $k\alpha([S_+]+[I_+])\frac{[S_-]}{N}$, represents the information diffusion term. The subtraction terms, $-\delta\lambda[I_+]-\epsilon\gamma[I_+]$, correspond to fading of individuals from the Infected aware population.

3.3 Numerical Solution of the Mean-Field ODE Model

We begin by defining two additional terms:

$$Information = \int_{t_0}^{t_f} (S_+(t) + I_+(t))dt, \tag{9}$$

$$Infected = \int_{t_0}^{t_f} (I_-(t) + I_+(t))dt, \tag{10}$$

where t_0 represents the starting time and t_f is the final time.

We next investigate the fixed α case that is introduced in [19] when they compare the three different methods that they consider in their paper. The following parameter values are used: $\alpha = 0.6, \beta = 0.3, \sigma_S = 0.3, \sigma_I = 0.6, \delta = 0.8, \epsilon = 1.5, \lambda = 0.15, k = 15$, and $\gamma = 0.1$.

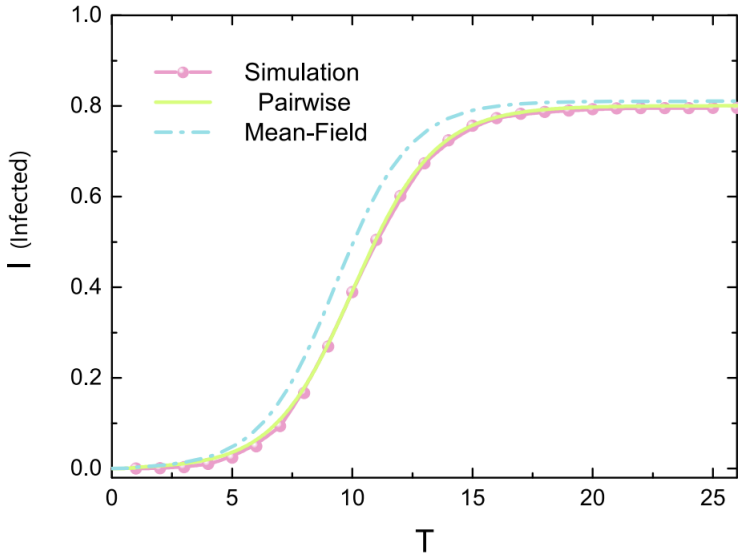


Figure 3.2: Infected component comparison from [19] where I = Infected, T = Time.

In Fig. 3.2, (from [19]), we can see that the Mean-Field method gives a total number of Infected individuals that is slightly higher than the corresponding values obtained from the Simulation and Pairwise methods. N, the total number of individuals, has a value of 10,000 [19] for the results presented in Fig. 3.2. Also, the authors chose $t_0 = 0$ and $t_f = 25$.

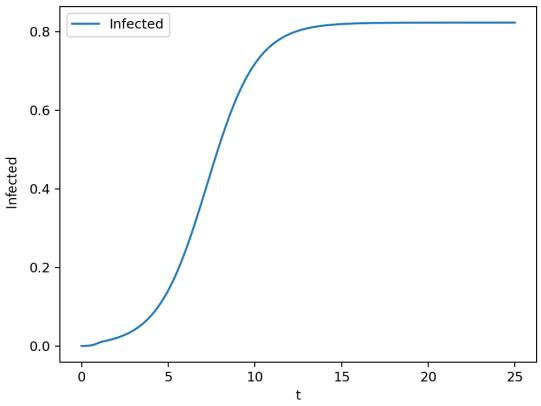


Figure 3.3: Our calculated Mean-Field Infected component over the interval [0, 25].

In Fig. 3.3, we present the results of our computations, where we use LSODA to solve the Mean-Field ODEs given earlier in this chapter. We see from Fig. 3.3 that we obtain a result that is similar to the corresponding result from [19], shown in Fig. 3.2.

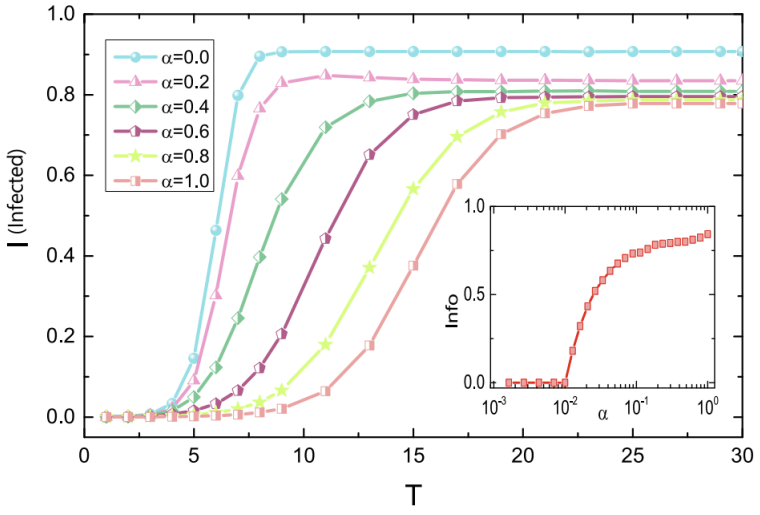


Figure 3.4: Simulation Method: Using Fixed α values I = Infected, T = Time [19].

We next consider the solution of the Mean-Field ODEs for several different values of α. Fig. 3.4 shows a comparison of how the value of α impacts the results of the total Infected population from [19]. The bottom right corner of Fig. 3.4 shows the total amount of information in the system as a function of α. As expected, when α decreases, we see a decrease in the total amount of information in the population.

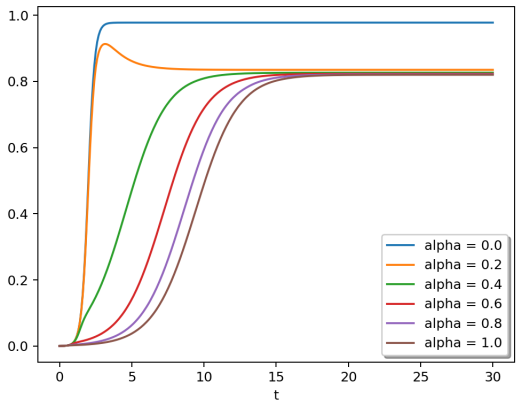


Figure 3.5: Comparison of Infected for population several α values.

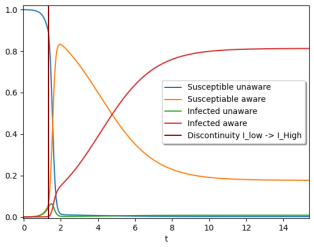
We attempt to verify these results with our calculation; our results are presented in Fig. 3.5. From Fig. 3.5, we see that, for most values of α, our results are similar values to those from [19]. The most significant difference is for the $\alpha = 0.2$ case. However, our results are associated with Mean-Field ODEs while the results shown in Fig. 3.4 are obtained using the Simulation Method, thus some differences between the solutions to the models are to be expected.

3.4 Discontinuities Analysis

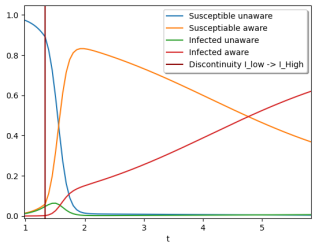
In this subsection, we consider the case where the α value is changed from time to time as t goes from t_0 to t_f. Later in this subsection, we will describe an approach that allows an initial value ODE solver to accurately and efficiently handle the

abrupt changes in α that arise. These abrupt changes in α lead to discontinuities in the right-hand sides of the ODEs in the model. The approach for dealing with these is called discontinuity handling.

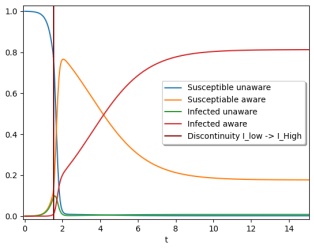
We first will show our results using discontinuity handling to solve the ODE system with parameter values $\beta = 0.3, \lambda = 0.15, \epsilon = 1.5, k = 15, \sigma_I = 0.6, \sigma_S = 0.3, \alpha = 0.6, \delta = 0.8, \gamma = 0.1$. Our initial values for a population of size 10,000, are $S_- = 9,997, S_+ = 1, I_- = 1, I_+ = 1$. This corresponds to the beginning of an infection where all components excluding susceptible unaware consist of a single individual. Unless explicitly stated these are the initial conditions for all of our computations.



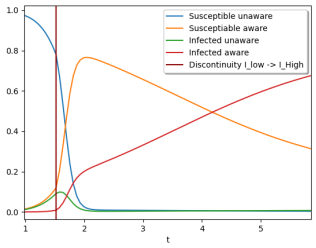
(a) $I_{High} = 0.05, I_{Low} = 0.0003$



(b) Fig. 3.6(a) Zoomed in



(c) $I_{High} = 0.1, I_{Low} = 0.001$



(d) Fig. 3.6(c) Zoomed in

Figure 3.6: Full Solution, $\alpha = \{0.3, 0.6, 0.8\}$, discontinuity points are t = 1.33 (a,b) , t = 1.52 (c,d).

The Infected thresholds, I_{High} and I_{Low}, are multiplied by the population size, as they represent a percentage of the infected. We consider three α values. We begin the computation with α = 0.6. When the total amount of Information reaches I_{low} we set $\alpha = 0.3$. When the total amount of Information reaches I_{high}, we set $\alpha = 0.8$.

When α changes abruptly, this causes a discontinuity in the system of ODEs,

as mentioned earlier. In turn, the presence of this discontinuity causes an initial value ODE solver to reduce its step size until either it fails, or it can step over the discontinuity. This directly translates into a decrease in efficiency and can often impact the reliability of the results.

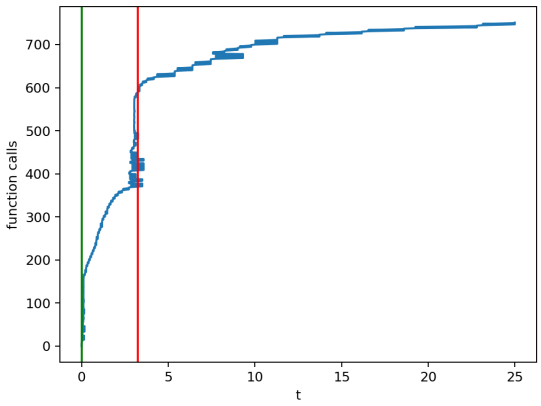


Figure 3.7: Total function evaluations over time, solver = Radau, (Green = I_{Low}, Red = I_{High}).

In order to implement the changes in the α value, we introduced "if" statements into the function that defines the right-hand side of the ODE system. In Fig. 3.7, we show the results of counting the cumulative number of evaluations of the right-hand side of the ODE system as time progresses. We also show the locations in time when the first two discontinuities are triggered, with green representing the I_{low} threshold and red representing the I_{High} threshold. We note that

there is a spike in the number of function calls wherever a discontinuity is introduced. The solver used to obtain these results is the Radau method from [12].

The reason for the spike in the number of function calls is that the solver is using adaptive step sizes in an attempt to get over the discontinuity, and in this particular situation, it does successfully step past the discontinuities with an acceptable error in the final result. However, this becomes an efficiency issue because the solver takes about 170 steps for the solver to step past the discontinuities. That means that the solver is taking about 340 additional steps to compute the solution compared to what is required when discontinuity handling is employed. The total number of function calls is about 730, which means the solver is doing 46% extra work due to the discontinuities. When discontinuity handling is employed, the solver requires about 390 function calls.

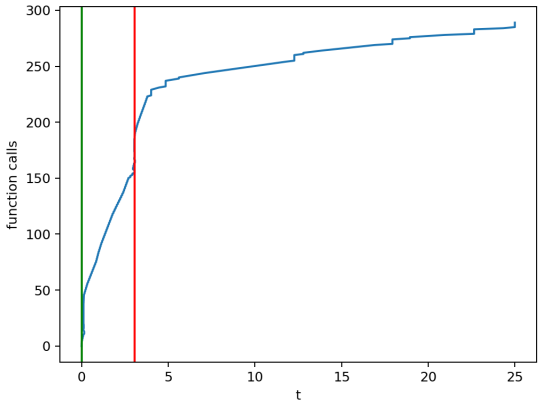


Figure 3.8: Total function evaluations over time, solver = LSODA, (Green = I_{Low}, Red = I_{High}).

We next conduct this same experiment using several other ODE solvers. Fig. 3.8 shows the corresponding results for the LSODA solver and Fig. 3.9 shows the results for the RK45 solver.

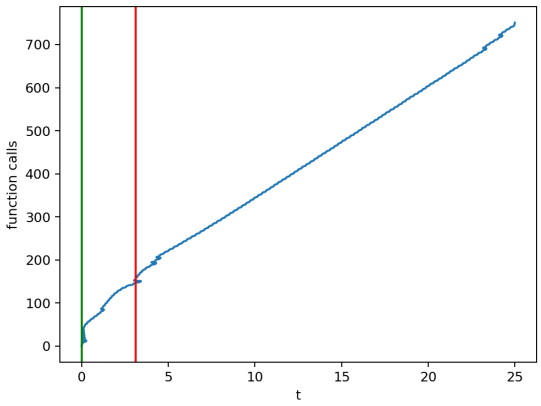


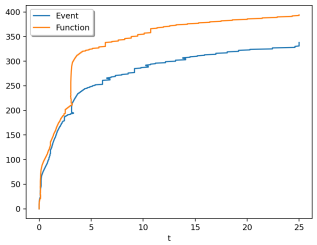
Figure 3.9: Total function evaluations over time, solver = RK45 (Green = I_{Low}, Red = I_{High}).

From Fig. 3.8, for the LSODA solver, we notice that about an additional 50 steps per discontinuity are required, resulting in roughly a 33% loss in efficiency. From Fig. 3.9, however, it appears RK45 has relatively better efficiency as it takes only a few extra steps to step past each discontinuity. The RK45 solver takes about an additional 25 more steps, which represents about a 3% loss in efficiency. However, comparing Fig. 3.9 with Fig. 3.8, we see that RK45 takes many more function evaluations overall than LSODA does.

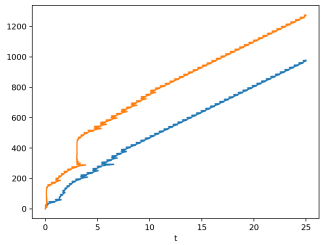
We next consider the question of what we can do to improvise the performance of the solvers when they encounter discontinuities. The key idea is to detect when there is a discontinuity. As mentioned earlier, this can be done by using what is

known as discontinuity or "event" handling; this is a capability that many state-of-the-art initial value ODE solvers have. An event is defined as a continuous function of time and state [12]. An event function has the form, $event(t, y(t))$. An event is defined to occur when the event function equals zero. The solver checks for a change in the sign of the event function over each time step. The sign change indicates that the event function has a root somewhere within the step; the solver uses a root-finding algorithm, e.g., bisection, to determine the time at which the root occurs. When an event is detected, we stop the solver and use the time of the event as the initial start time for a new call to the solver. Restarting the solver after each discontinuity is referred to as a "cold start" we repeat this process until the final time step is reached. We append all the numerical solutions computed between the discontinuities together in order to obtain the final solution.

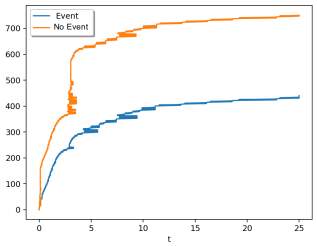
We next examine the impact of this discontinuity handling scheme across all methods available to us through Scipy.integrate.solve_ivp class.



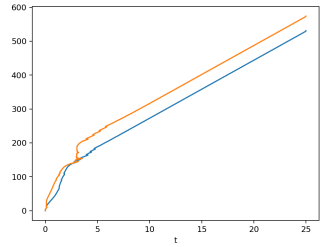
(a) BDF



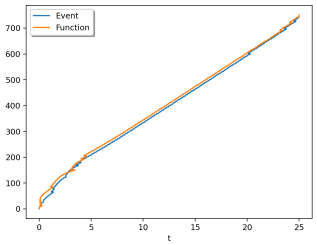
(b) DOP853



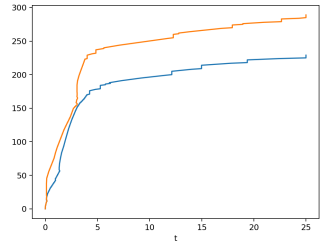
(c) Radau



(d) RK23



(e) RK45



(f) LSODA

Figure 3.10: Comparison of methods using discontinuity handling (Blue) with the same methods when they do not use discontinuity handling (Orange); relative tolerance = 10^{-3}, absolute tolerance = 10^{-6} (Default), y-axis = Number of Function Evaluations.

Figure 3.10 shows a comparison of six solvers applied to the version of the Mean-Field ODE model that involves discontinuities. In each case, the solvers are applied with no discontinuity handling and with discontinuity handling.

Fig. 3.10(c) (where the Radau solver is used) shows the biggest performance gain, while Fig. 3.10(e) (where the RK45 solver is used) shows very little performance gain. This may lead one to the conclusion that RK45 does not need to use discontinuity handling. However, if we sharpen the absolute and relative tolerances to 10^{-12}, we obtain the result shown in Fig. 3.11.

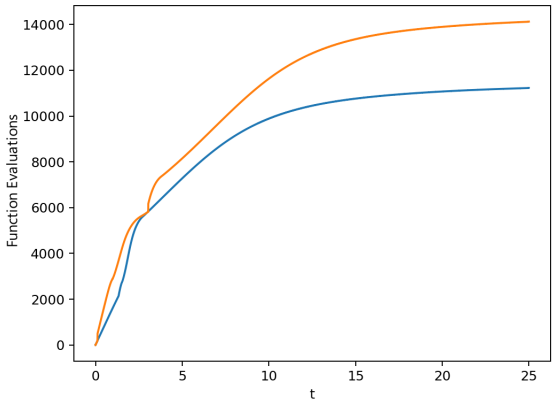


Figure 3.11: RK45, tolerances = 10^{-12}, y-axis = Number of Function Calls.

Fig. 3.11 shows that discontinuities have an impact even for the RK45 solver when the tolerance is sufficiently sharp. As seen in Fig. 3.11, about 4000 more function calls are required when discontinuity handling is not employed.

The benefits of using discontinuity handling are not limited to efficiency improvements. The use of discontinuity detection can also impact the accuracy of the results. From Fig. 3.12 we observe that the computed solutions are much different; the time at which the I_{high} threshold is reached is much different when discontinuity handling is employed compared to when it is reached when no discontinuity handling is employed. This means that solving ODE systems with embedded "if" statements and no discontinuity handling can also lead to inaccurate results. It can be seen in Figure 3.12 that solution obtain using event handling and the solution obtained using no event handling are much different.

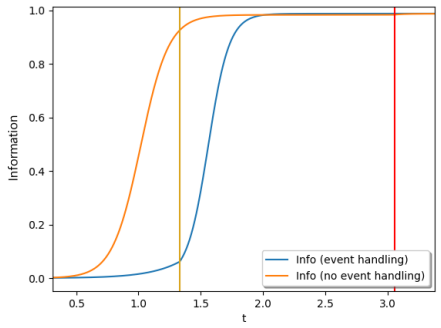


Figure 3.12: Event handling and no event handling for the LSODA solver; yellow vertical line shows time at which I_{high} is reached when event detection is employed; red vertical line shows time at which I_{high} is reached when no event detection is employed.

In [19], the authors do not specify what numerical methods they use. There is the possibility that the authors used discontinuity handling however there is no mention of it. If the authors did not use discontinuity handling then it is possible that their results are inaccurate and it is certain that they are computed inefficiently.

4 G-ODE-PDE Solver

In this chapter, we describe our new graphical ODE/PDE solver which we call G-ODE-PDE. To our knowledge there are currently three similar software graphical user interfaces (GUI) for solving ODEs, namely, Geode [6], Wolfram Alpha [15], and The Graphical ODE Solver [8]. The Graphical ODE Solver is primarily a teaching tool. All three have their benefits; Wolfram Alpha is the most advanced of the three. Wolfram Alpha is capable of also handling PDEs, but it can only handle one equation. Geode and the solver described in [8] can solve multiple ODEs, but neither has support for PDEs.

In this chapter we will describe the new software we have developed in order to support application experts who need to be able to numerically solve mathematical models involving systems of ODES and PDES, but who have a limited experience in programming. The point of this software is to allow the user to ac-

cess high quality ODE and PDE solvers without the user having to know how to program. It was important for us to allow access to this tool through a client/server model. This allows multiple researchers to run this software on a single machine. However, for the client, we opted for a software platform that will allow users to access our GUI using any modern browser.

4.1 Introduction

In order to use initial value solvers for ODEs, a user typically must have a reasonable grasp of programming; in addition, the user may need to know how to use certain software libraries with a programming environment. However, we understand that not every user who wants to solve initial value ODEs is comfortable with programming. This issue becomes amplified when we consider PDE solvers since these typically require a higher degree of programming experience relative to ODE solvers.

As mentioned earlier, our primary motivation for the development of this software is to enhance accessibility to several high-quality ODE and PDE solvers. It is important that our software builds upon already well-established ODE and PDE solvers. Our software package is split into two main components: the client side and the server side. The client-side is designed for the user; it is written

in JavaScript and HTML, making it accessible from any modern browser. The server side is written in Python and can be run on a server so that it can be available to multiple users. However, in order for our software to run there must be a FORTRAN compiler available due to the fact we employ a FORTRAN error control PDE solver, BACOLI, in order to compute numerical solutions to PDEs. The purpose of this chapter is to introduce our new ODE/PDE solver interface tool, G-ODE-PDE, and demonstrate how one might use it to solve both ODEs and PDEs. The solvers available within the ODE portion of our software are the suite of solvers available in scipy.integrate.solve_ivp [12] and for PDEs, the GUI employs bacoli_py [3], which is a Python interface to the BACOLI FORTRAN solver mentioned earlier.

4.2 Using the GUI to solve an initial value ODE problem

In this section we will review the user interface for the ODE portion of our software.

4.2.1 Example 1

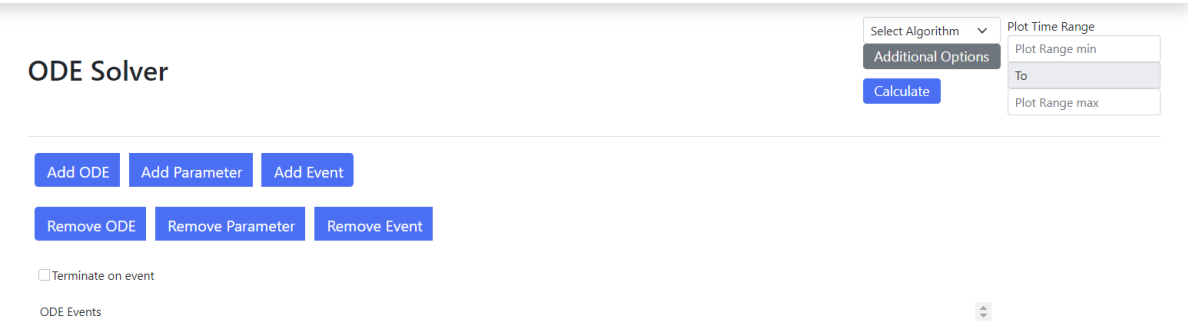


Figure 4.1: ODE full screen.

Fig. 4.1 presents the interface to the suite of ODE solvers. This interface presents the user with buttons for specifying ODES, parameters, and events.

We first discuss how the ODE and the initial value can be input. We consider an ODE system consisting of 1 differential equation. The ODE is y'(t) = sin(t), with an initial value of y(0) = 0; this problem has an exact solution of y(t) = 1 - cos(x). We want to compute the solution on the time interval [0,7]. Default values will be used for the absolute and relative tolerances.

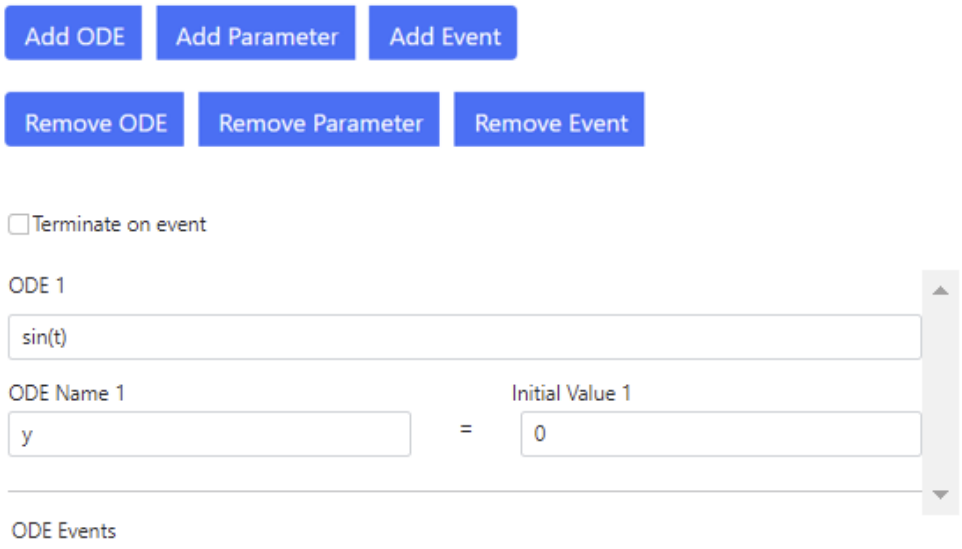


Figure 4.2: An example of how the ODE y'(t) = sin(t), y(0) = 0 is specified.

In Fig. 4.2 we give a screenshot of the part of the interface to our software that allows the user to specify the ODE and the initial condition. We have included buttons for ODEs, initial conditions, events, and parameters. Events and parameters will be discussed in the next example. The ODEs are labeled in ascending order starting from 1; should we want to remove an ODE from the system, the remove ODE button will remove the highest numbered ODE. Next, we have the right-hand side of the ODE which in this example is $\sin(t)$; t is a keyword in this software that refers to the independent variable, t. Underneath, we specify the initial value, $y(t_0) = 0$, where t_0 is defined in the initial time range (to be considered

later in this section). The “Terminate on event” checkbox is associated with event handling and will be explained later in this chapter.

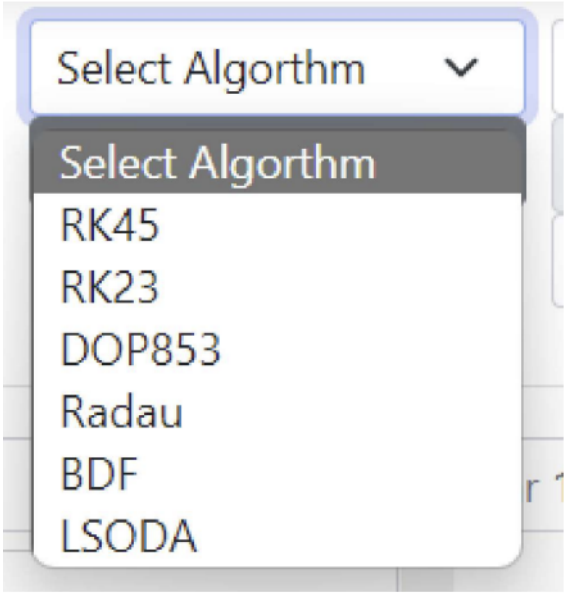


Figure 4.3: ODE solvers

After we have described the initial value ODE problem to the software we next need to select the solver that we want to use for the calculation. Fig. 4.3 shows another screenshot where the menu displays the available ODE solvers to the user. Selecting the “Select Algorithm” button shows the new screenshot given in Fig. 4.3; the user is given the choice between 6 solvers. We will select RK45 in this example.

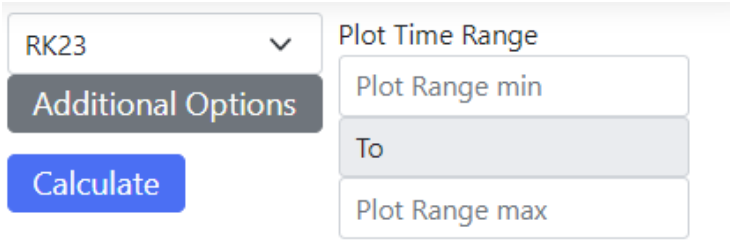


Figure 4.4: Options for time range, solver selection.

Next to the drop-down menu for the ODE solver selection, we can select the interval on which we want the solution to the ODE. In this case, the interval is from 0 to 7. There is also an "Additional Options" button; we will discuss its purpose in the next example. The user can initiate the numerical solution of the initial value ODE problem by clicking on the "Calculate" button. Once all required information is entered and the "Calculate" button has been clicked, the information will be sent to the server and the numerical solution will be calculated and returned to the GUI. See Fig. 4.5, where a screenshot showing the input fields and the computed solution to the initial value ODE problem is given. In Fig. 4.5, all the labels on the graph are editable; the user can simply click on a label in order to edit it.

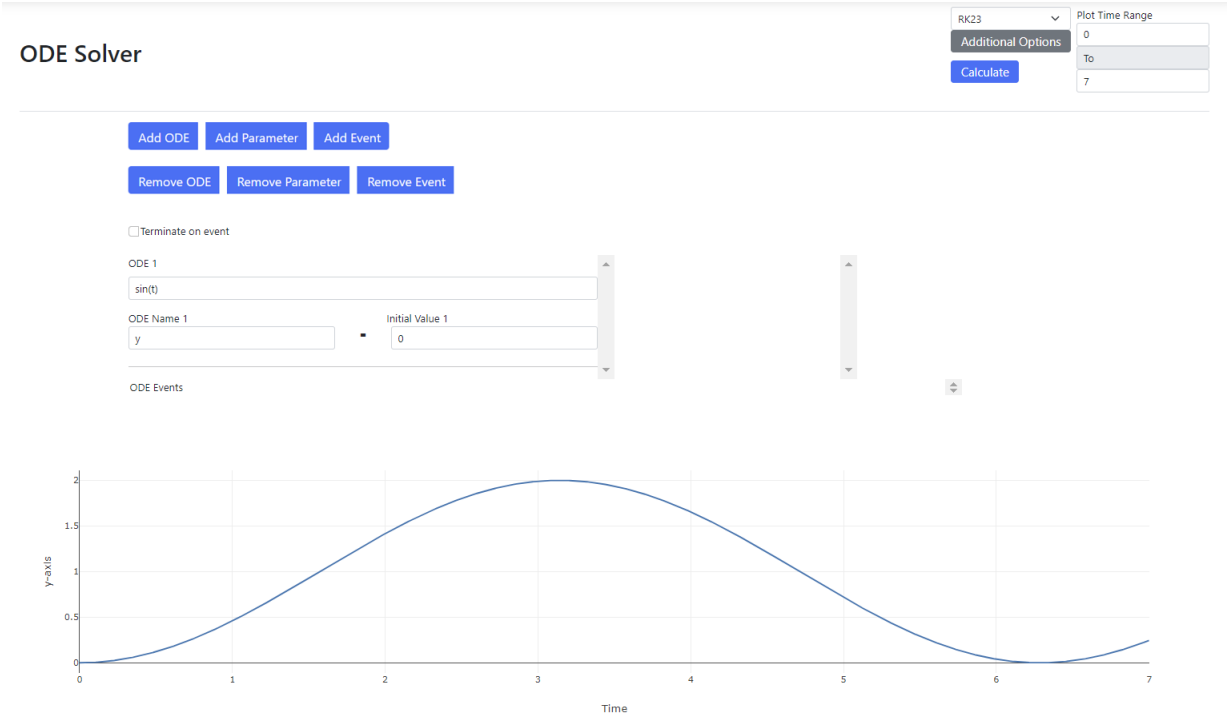


Figure 4.5: Numerical solution for y'(t) = sin(t), y(0) = 0. Solved with RK23 using absolute tolerance = 10^{-6}, relative tolerance = 10^{-4} (default tolerances), on interval $t \in [0, 7]$.

4.2.2 Example 2

Next, we consider a slightly more complicated problem, the Lotka-Volterra Predator Prey ODE system [9]. Equation (11) shows this system.

$$\begin{aligned} \frac{dx}{dt} &= \alpha x - \beta xy, \\ \frac{dy}{dt} &= \delta xy - \gamma y. \end{aligned} \tag{11}$$

As described by [9], we use the initial conditions of $x(0) = y(0) = 10$. The variable x is the number of prey, the variable y is the number of predators, t represents the time, and $\alpha = 1.1, \beta = 0.4, \delta = 0.1, \gamma = 0.4$, are positive real parameters that describe the interaction between the predators and preys; the derivatives, $\frac{dx}{dt}$ and $\frac{dy}{dt}$, represent the growth rates of the two populations over time. The first equation describes the rate of change in the prey population, while the second equation describes the rate of change in the predator population. This example will include an event; events are defined as a continuous function of time and state [12] where the event function, $event(t, y(t))$, is triggered when it equals zero. At the end of each time step the ODE solver checks for a change in the sign of the event function [12]; when a sign change in the event function is detected for a given time step, the solver uses a root-finding algorithm, e.g., bisection, to locate the time of the event. By selecting the "Terminate on event" button, the user can indicate to the solver that they want to exit the computation when the event is detected and return the result. If the "Terminate on event" checkbox is not selected, the solver will continue until t_f.

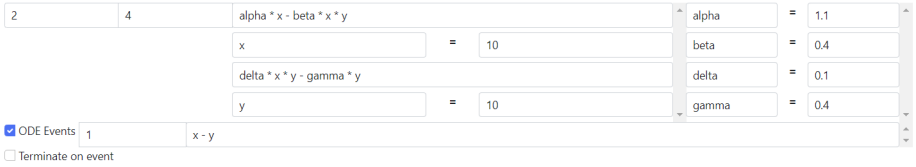


Figure 4.6: Input for the Lotka-Volterra equations.

Fig. 4.6 we have set the initial values of $x(0) = y(0) = 10$. In this example, we are using both variables, "x" and "y", in the specification of the ODEs. The ODE event here is x - y; this event corresponds to asking the software to look for the points in time at which x = y.

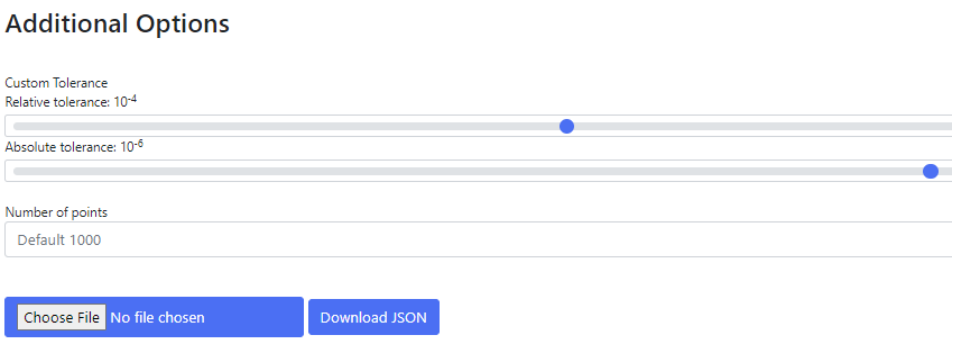


Figure 4.7: Additional settings for tolerance and number of output points.

By selecting, "Additional Options", as seen in Fig. 4.7, the user is able to set both an absolute and relative tolerance. Directly below the tolerance sliders is a field where one can set the number of points to be plotted; the default is 1000.

After that we have two buttons; one is to upload a JSON formatted file. JSON is defined as JavaScript Object Notation, and is used for transporting and storing objects [14]. When a JSON formatted file is uploaded, it will fill all the fields within the interface, including ODEs, parameters, tolerances, etc. The “Download JSON” button allows for the download of the data that has been entered into the system. The download button gives the user the option of downloading the plot data as well. This means that the next time the user uploads a JSON formatted file, they will be presented with their results without the need to recalculate the solution. For this example, we use the “Select Algorithm” button to select the LSODA solver.

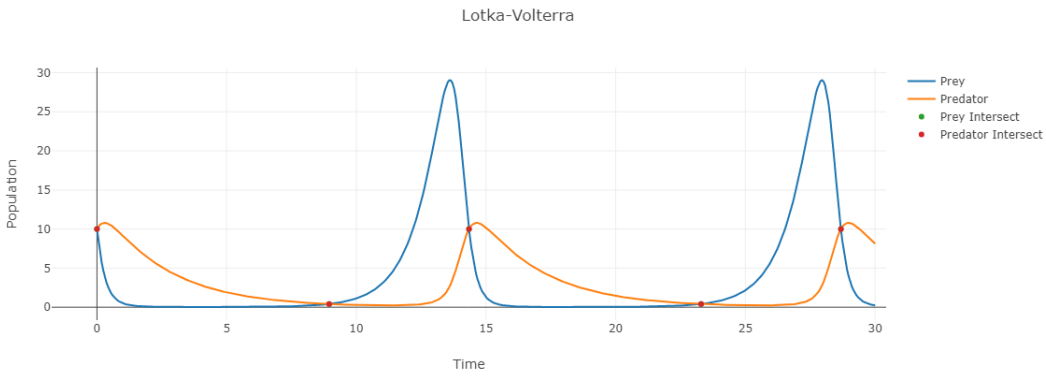


Figure 4.8: Numerical solution of Lotka-Volterra equations.

Clicking on the ”Calculate” button causes our software to send the problem

description to the server where the selected solver computes a numerical solution to the ODE system and then returns it to our interface software. The resultant plot is shown in Fig. 4.8. We note that the events have been identified; for each event, the time at which the event occurs is represented by the red and green points on the plot.

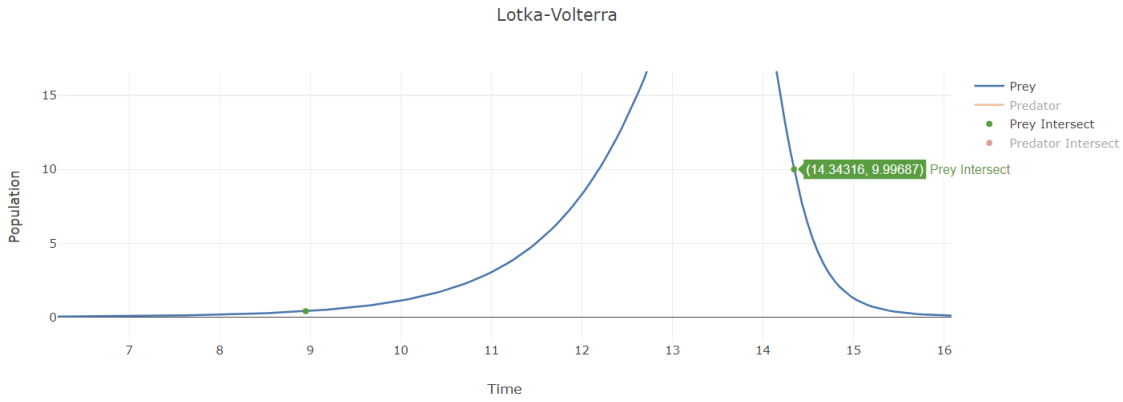


Figure 4.9: Lotka-Volterra solution event.

However, the green points are not visible as they share the same value with red points, because the event function is x - y. The user can hide and show the event values by selecting them on the legend to the right of the plot. In Fig. 4.9, we have zoomed in on the part of the plot that corresponds to the time interval [7,14]. Our software allows the user to zoom into a section of the plot by holding down a mouse button and dragging over the section. The user can also hover over an event to see what the $(t, x(t))$ and $(t, y(t))$ values are. The user can also hide

solution components; in Fig. 4.9 we do not show the predator solution component.

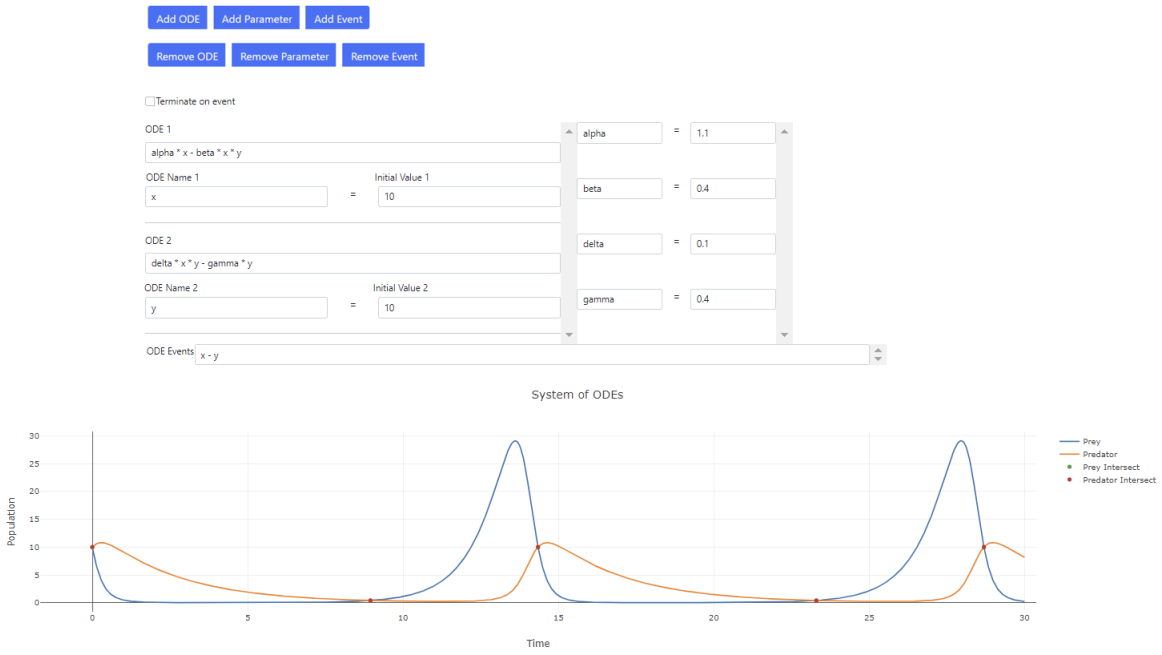


Figure 4.10: Full-page screen for Example 2.

Reviewing the whole screen in Fig. 4.10, we can see how the screen would look with all the information presented in order to set up this example; the resultant computed solutions are also shown. The user may also want to save their results for later use. This can be done by selecting the "Additional Options" button (see Fig. 4.4) in order to download a JSON file. The JSON file for this example is shown in Fig. 4.11

```
{
    "Equations": { "set1": {
            "equation": "alpha * x - beta * x * y",  "initialVarableName": "x", "initVariableValue": "10"},
                       "set2": {
            "equation": "delta * x * y - gamma * y", "initialVarableName": "y", "initVariableValue": "10"}
     },
    "Variables": { "set1": { "VariableName": "alpha", "Value": "1.1" }, "set2": {"VariableName": "beta", "
         Value": "0.4" },
                       "set3": { "VariableName": "delta", "Value": "0.1" }, "set4": {"VariableName": "gamma"
                            ,"Value": "0.4"}
    },
    "Events": {
        "event1": "x - y"
    },
    "EventValues": [
        [ [ 0, ..., 28.68231856533941] ],
          [ [ [10, ...,  9.993756434440286],
              [10, ..., 9.993756434440268]
            ]
          ]
    ],
    "Algorithm": "LSODA",
    "TimeRange": {"timeStart": "0","timeFinish": "30", "points": "500"},
    "Tolerance": {"relativeTolerance": "4","absoluteTolerance": "6"},
    "Data": {"t": [0, ..., 30], "y": [10, 10, ..., 0.23689628593177658, 8.119505654684382]
    }
}
```

Figure 4.11: JSON file for the Lotka-Volterra example. Note: "..." will be replaced with data points.

Once the user has downloaded the JSON file, they can later upload this file to an empty page within the GUI. Doing this will auto fill the necessary fields and

present the data shown in Fig. 4.10. One important detail regarding the JSON file should be noted here. Referring to line 23 in Fig. 4.11. the y data structure specifies the two solution components for each output time.

4.3 Using the GUI to solve a PDE

This section describes how the GUI can be used to solve a PDE. We consider the One Layer Burgers Equation; this equation is defined as follows.

$$\begin{aligned} u_t(t,x) = &\quad \epsilon u_{xx}(t,x) - u(t,x)u_x(t,x), \epsilon \in \mathbb{R}, \\ u(t_0,x) = &\quad \tfrac{1}{2} - \tfrac{\tanh(\frac{x-\frac{1}{4}}{4\epsilon})}{2}, \quad x \in [0,1], \\ u(t,x_a) = &\quad \tfrac{1}{2} - \tfrac{\tanh(\frac{-\frac{1}{2}t-\frac{1}{4}}{4\epsilon})}{2}, \quad t \in [0,1], \\ u(t,x_b) = &\quad \tfrac{1}{2} - \tfrac{\tanh(\frac{\frac{3}{4}-\frac{1}{2}t}{4\epsilon})}{2}, \quad t \in [0,1]. \end{aligned} \tag{12}$$

In Equation (12), $u(t,x)$ represents the unknown solution that we want to compute, the parameter ϵ is chosen to have the value 10^{-3}, the initial time is $t_0 = 0$, and $x_a = 0$ and $x_b = 1$. The second equation is the initial condition and the left and right Dirichlet boundary conditions are given in the third and fourth equations. The time domain is $t \in [0,1]$. We choose default tolerances; this means that the absolute tolerance will be 10^{-4} and that the relative tolerance will be 10^{-4}. Now that we have defined the model, we describe how to enter the model

information into our interface.

Figure 4.12: One layer Burgers Equation; specification of the PDE, the initial and boundary conditions, and the parameter ϵ.

In Fig. 4.12, we can see the PDE, initial condition, and boundary conditions entered into the user interface. First we specify the PDE and the parameter ϵ. For PDEs, we have five reserved variables: x, t, u, ux, uxx; x is the spatial variable, t is the time variable, and u, ux, and uxx represent u and its first and second spatial derivatives. The first field allows us to specify the PDE. The next three fields allow us to specify the initial condition, followed by the left and right boundary conditions. The field on the right allows us to set the value for the parameter, ϵ.

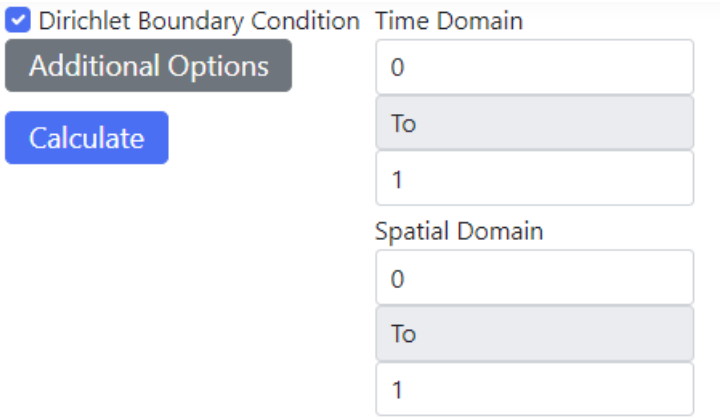


Figure 4.13: Specification of the spatial domain and the time domain

Fig. 4.13 shows how the user can specify the time range and the spatial domain, which for this example is [0,1]. The Dirichlet boundary condition checkbox defines the type of boundary condition. The choice is between Neumann and Dirichlet boundary conditions. A Neumann boundary condition means that the boundary condition involves the derivative of the solution. A Dirichlet boundary condition means that the boundary condition depends upon the solution itself. Since, for this example, we have Dirichlet boundary conditions, the "Dirichlet Boundary Condition" box should be checked. We use the default tolerances, which in our case are an absolute tolerance of 10^{-4} and a relative tolerance of 10^{-4}. After all the required information has been input to the appropriate fields, the user should select the "Calculate" button. This will send the information to the

server, where a solution to the PDE will be calculated and returned to the interface software. For this example the computed solution is shown in Fig. 4.14.

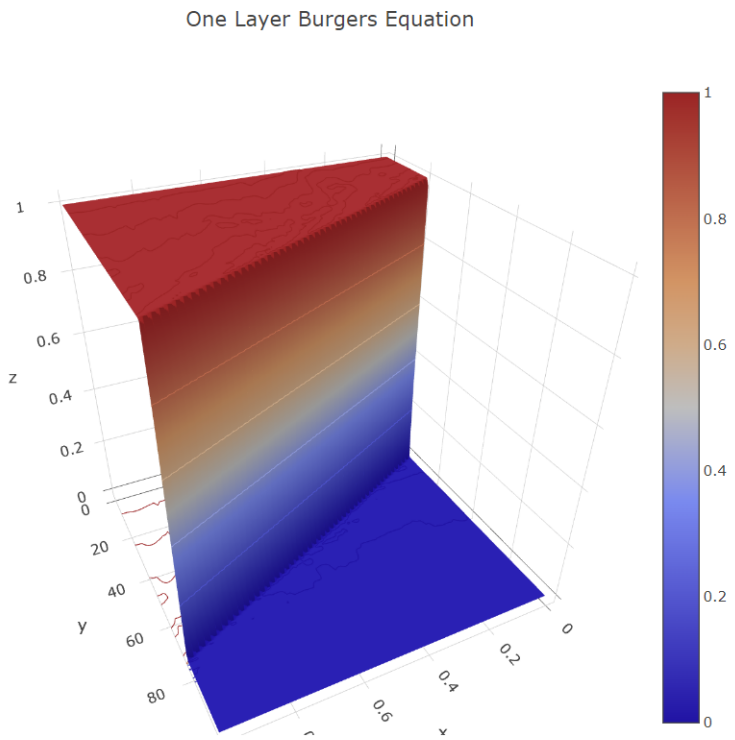


Figure 4.14: Calculated solution for Burgers Equation.

Fig. 4.14 shows a 3D plot which the user can rotate and move around by holding down the mouse button and moving the mouse. The user can also zoom in and out of the plot by using the mouse wheel. Similar to the ODE section, the user is able to download the JSON file for post processing.

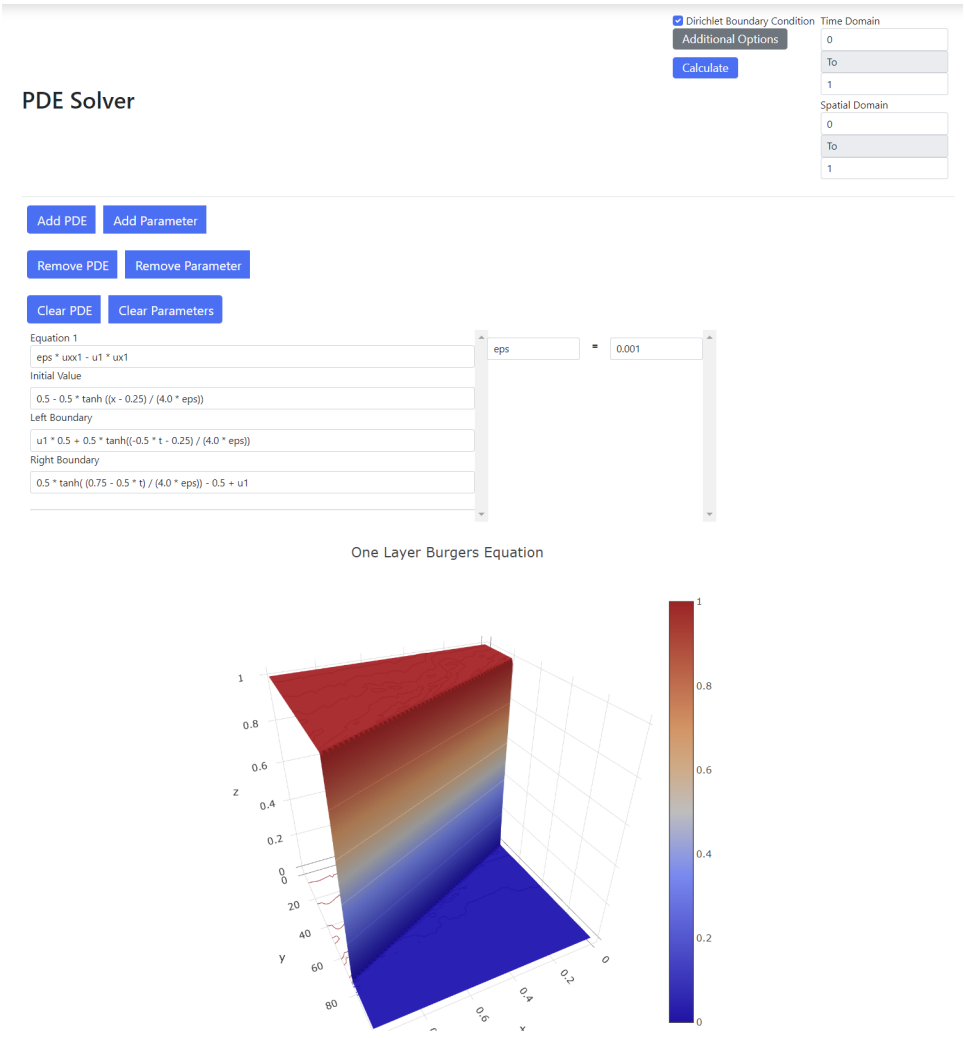


Figure 4.15: Full-page screen for PDE example.

Fig. 4.15 shows a full screen of the GUI once all data is entered and the solution is calculated.

4.4 Summary

This software allows the user to compute numerical solutions to ODEs and PDEs without the user needing to be able to program. Ultimately, the goal is to allow researchers more time to concentrate on the solving the mathematical model rather than having to have experience in programming. This software utilizes Scipy.integrate.Solve_ivp for the ODE solvers; it makes use of the Bacoli_py software for solving PDEs.

4.5 Future Work

Further work needs to be done to improve error reporting; for example, currently if BACOLI fails, it will kill the process resulting in the main thread of our software also being killed. A future timeout feature is currently also needed to eliminate undesirably long computations. We would like to implement an ability to allow the user to change parameter values while the software is computing the numerical solution from t_0 to t_f. Optimal parameter selection of ODE and PDE parameters, as seen earlier in this thesis, is another important capability, and should be near the top of the list of things to be added to the GUI.

5 Summary, Conclusions and Future work

5.1 Summary

In this thesis, we considered three scientific computing projects. The first project investigated a mathematical model [16] associated with the study of misinformation from the Twitter feed associated with the death of George Floyd. This investigation included using an error control initial value ODE solver from Python to verify the results from [16]. The second project investigated mathematical models [19] associated with information diffusion during epidemics. This investigation involved a careful study of how well several of the initial value ODE solvers from Python are able to solve one of the models given the presence of a discontinuity in the definition of the model. Finally, we described some new work involving the development of a GUI, called G-ODE-PDE, that allows a user to access the suite of error control initial value ODE solvers available in scipy.integrate.solve_ivp and the FORTRAN error control PDE solver, BACOLI, available through the Bacoli_py Python interface.

5.2 Conclusions

Regarding the mathematical models that are based on the Twitter feed associated with the riots following the murder of George Floyd, we found that the numerical solutions that we computed agree well with those that were reported in [16]. We accessed the original Twitter feed to collect similar data to what was collected in [16] and found that the authors did not include the first 4 hours and 15 minutes of data. Therefore, in our work, we extended the investigation to include fitting the parameters of the ODE model to this new data set. We presented a numerical solution of this model for this extended data set. We observed that the model does not fit this extended data set as well as it did the original data set.

Regarding the investigation that involved a mathematical model for information diffusion associated with an epidemic, we first solved one of the models from [19] to experimentally verify the results from [19]. We also applied the six initial value problem solvers available in [12], and found that the presence of a discontinuity in one of the parameters of the model led to some noticeable challenges for some of the solvers. Even though these are high quality solvers, care must be taken when these solvers are applied to problems with discontinuities. We showed that by employing discontinuity detection based on the event detection capability of the solvers and then using cold starts at the times of the discontinuities, we can

get more accurate solutions from the solvers and also improve the efficiency of the computations.

Regarding the G-ODE-PDE GUI, we demonstrated that this new tool allows the user access to the error control ODE solvers in [12], and the error control PDE solver, BACOLI, accessible through the Python interface, bacoli_py. In particular, the user does not need to know how to program; this will allow a wider class of users to have access to these solvers; they simply need to enter the information required to describe the ODE or PDE system, and then the GUI can compute numerical solutions and provide plots of the solution which can be then downloaded by the user.

5.3 Future work

Regarding the work considered in Chapter 2, by including the data from the first 4 hours and 15 minutes, we observe that this data exhibits substantially different behavior than the data that was considered in the original paper [16]. In order to accommodate this different behavior, it would be interesting to introduce one or more terms into the ODE model to attempt to represent the kind of behavior we see in the data when we include the additional data from first 4 hours and 15 minutes. Therefore, future work on this project will involve investigating mod-

ifications of the ODE system that will allow the solution of the ODE model to better reflect what the data shows us when the first 4 hours and 15 minutes of data is included. Another idea for future work would be to modify the ODE system to allow those in the skeptic population to possibly become part of the exposed population. We believe this would better model the real world, as Skeptics would be able to see new misinformation over time that may lead them to engage with the misinformation.

Regarding the investigation of Chapter 3, the original paper also considers a more complex ODE system that involves 14 ODEs. Thus future work in this area would be to generalize our investigation to consider this more complicated model. As well, we would be interested in examining how the presence of a discontinuity in this model impacts the performance of the initial value ODE solvers in terms of efficiency and accuracy.

Regarding the G-ODE-PDE GUI software that we have developed, there needs to be more work done with exception handling in order to be able to inform the user of issues that may arise during the computation. There is a new version of BACOLI, called BACOLIKR, that offers a generalization, to the PDE setting, of the event detection capability that is available in many high quality initial value ODE solvers. Therefore, future work could also include generalizing the Ba-

coli_py interface to use BACOLIKR [17] rather than BACOLI and then generalizing the GUI so that it could provide access to an event detection capability for PDEs.

References

[1] B-splines. https://en.wikipedia.org/wiki/B-spline. Accessed: 2023-04-23.

[2] BACOLI. https://cs.smu.ca/~muir/BACOLI-3_Webpage.htm. Accessed: 2022-06-25.

[3] Bacoli_py. https://github.com/connortannahill/bacoli_py. Accessed: 2022-06-25.

[4] Differential-algebraic systems of equations. https://en.wikipedia.org/wiki/Differential-algebraic_system_of_equations. Accessed: 2023-04-23.

[5] Gaussian quadrature. https://en.wikipedia.org/wiki/Gaussian_quadrature. Accessed: 2023-04-23.

[6] Geode. http://graphics.stanford.edu/~klingner/geode/. Accessed: 2023-04-21.

[7] George Floyd protests in Washington, D.C. https://en.wikipedia.org/wiki/George_Floyd_protests_in_Washington,_D.C. Accessed: 2022-06-25.

[8] Graphical ODE solver for one or two ordinary differential equations. https://www.mathworks.com/matlabcentral/fileexchange/26146-graphical-ode-solver-for-one-or-two-ordinary-differential-equations. Accessed: 2023-04-21.

[9] Lotka-Volterra equations. https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations#:~:text=The%20Lotka%E2%80%93Volterra%20equations%2C%20also,and%20the%20other%20as%20prey. Accessed: 2023-04-20.

[10] Mathworks - least-squares (model fitting) algorithms. https://www.mathworks.com/help/optim/ug/

least-squares-model-fitting-algorithms.html. Accessed: 2023-04-18.

[11] Mathworks - ode45. https://www.mathworks.com/help/matlab/ref/ode45.html. Accessed: 2023-04-18.

[12] Scipy python package solve_ivp. https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html. Accessed: 2022-06-25.

[13] Twint - twitter intelligence tool. https://github.com/twintproject/twint. Accessed: 2022-06-25.

[14] What is JSON. https://www.oracle.com/ca-en/database/what-is-json/. Accessed: 2023-04-21.

[15] Wolfram Alpha numerical differential equation solving. https://www.wolframalpha.com/examples/mathematics/differential-equations/numerical-differential-equation-solving. Accessed: 2023-04-21.

[16] M. Maleki, E. Mead, M. Arani, and N. Agarwal. *Using an epidemiological model to study the spread of misinformation during the black lives matter movement*. *arXiv preprint arXiv:2103.12191*, 2021.

[17] P. Muir, J. Pew, and C. Tannahill. *Error control B-spline Gaussian collocation PDE software with time-space event detection*. Technical report, *Saint Mary's University, Dept. of Mathematics and Computing Science Technical Report*, 2020.

[18] L. R. Petzold. Description of DASSL: a differential/algebraic system solver. Technical report, Sandia National Labs., Livermore, CA (USA), 1982.

[19] X. Zhan, C. Liu, G. Zhou, Z.-K. Zhang, G.-Q. Sun, J. J. Zhu, and Z. Jin. Coupling dynamics of epidemic spreading and information diffusion on complex networks. *Applied Mathematics and Computation*, 332:437–448, 2018.