Saint Mary's University
Department of Mathematics and Computing Science

Application of Error Control Software to ODE and PDE-based Epidemiological Models

Author:
Connor Tannahill

Supervisor:
Dr. Paul Muir

October 18, 2017

Abstract

Epidemiological models provide powerful tools for predicting the spread of a disease and understanding the dynamics which effect its transmission through a population. In this report, we describe time dependent and time-space dependent compartmental epidemic models and the application of adaptive error control numerical software to approximate their solutions.

Software that implements adaptive error control returns an approximate solution for which an associated error estimate satisfies a user-prescribed tolerance; this has two important advantages: (i) the user can have reasonable confidence that the numerical solution has an error that is consistent with the requested tolerance, and (ii) the cost of the computation is consistent with the requested accuracy.

Several examples of time dependent ODE based models are solved within the Scilab problem solving environment to compute error controlled solutions. A number of time-space PDE based epidemic models are solved using BACOLI, a recently developed FORTRAN B-spline collocation software package which provides adaptive spatial and temporal error control. We compare the numerical results from several previously published models with those obtained using our error control software based approach. We find that in some cases, the numerical results published are inconsistent with those we obtained, and that these descrepencies may be due uncontrolled error in the original numerical solutions.

Contents

List of Figures

List of Tables

1 Introduction

It is the goal of epidemiologists to understand how and why diseases are able to spread through populations, and through this understanding, be able to make predictions useful in managing current and future outbreaks [8]. Much of this research is done through the development of mathematical models to capture the behavior of a disease as it is transmitted through a host population over time, enabling us to analyze its future behavior. Dynamic, compartmental models of epidemiology attempt to accomplish this by dividing a host population into a number of compartments, each representing a stage of a disease, that are at any point in time occupied by a particular portion of the population [4, 19]. Individuals transfer between these compartments based on assumptions about how the transmission of a disease takes place within its host population, which are informed by the observed behavior of the disease being studied [8, 4].

A classic example of a compartmental epidemic model is the classic Kermack-McKendrick susceptible-infectious-recovered, or SIR model [14, 1, 30]. With the SIR model, a host population is divided into three compartments, those who are currently susceptible to infection (S), those who have been infected and are now able to propagate the infection (I), and the recovered class, who have endured the duration of the illness and are no longer infected (R) [14, 1, 30]. A visualization of these transmission dynamics

can be seen in Figure 1. As a disease progresses, the way the population transitions between compartments is modeled using the following system of time-dependent Ordinary Differential Equations (ODE's), with the parameters: β, the rate of transmission of the disease (i.e., the rate at which individuals transition from S to I), and v the recovery rate of the disease (i.e., the rate at which individuals transition from I to R) [14, 30, 19]:

$$\frac{dS}{dt} = -\beta IS, \tag{1}$$

$$\frac{dI}{dt} = \beta IS - vI, \tag{2}$$

$$\frac{dR}{dt} = vI. \tag{3}$$

This model, as in many of these models, is devised such that the total population relevant to the epidemic, N, can be expressed as $N = S+I+R$, or if we consider the proportion of the population present in each compartment, $1 = S + I + R$.

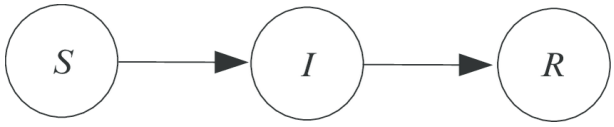


Figure 1: The SIR Model [1]

When one considers the dynamic, complex nature of a real world population, and the many factors which could effect the likelihood of an individual contracting a disease, it is clear that the SIR model is a simplification of the actual dynamics of an infectious disease. One major assumption made here is that the disease transmission follows the mass action law (expressed by the term βIS), meaning that each member of the host population is assumed to be equally likely to be in contact with an infected individual at any given time. In models making use of the mass action law, the rate of transmission is directly proportional to the population size rather than characteristics such as population movement or distribution which we would expect in reality to have a large effect on disease transmission; a disease population following these transmission dynamics is said to be homogeneously mixed [8].

These time dependent models have proven themselves useful in modeling real-world infections, but are not able to model structure in the host population, including considerations such as age and spatial distribution which can have an effect on disease transmission [4]. In order to more accurately represent these dynamics, a variety of models have been devised which take into account more complex behaviors of epidemic populations, including those which consider the spatial distribution of individuals and how they

move within space [4]. Although these models can give a more accurate approximation of disease behavior, they often require the costly collection of ecological data in order to approximate the model parameters. It is therefore the case that simpler models with fewer unknowns are sometimes preferred [37].

Many models of epidemiology use statistical approaches, which do tend to produce a more accurate representation of disease transmission [37]. However, the value mathematical models have is that they allow observation of the asymptotic behavior of an epidemic, and using this information, predict which model parameters have the most impact on disease transmission [37]. Ronald Ross, one of the early researchers in mathematical epidemiology, purposed the use of information from compartmental models in conjunction with statistical models, calling this the *a priori* method, which he described as: "we assume a knowledge of the causes, construct our differential equations on that supposition, follow up the logical consequences, and finally test the calculated results by comparing them with the observed statistics" [37]. Using this method, Ross was able to make predictions about which parameters most affected the spread of malaria and proposed methods for controlling and preventing outbreaks [37, 8].

In order to predict the behavior of an epidemic in the future, the study of two biologically significant equilibria are particularly useful, the disease-free equilibrium, where the portion of infected individuals approaches 0 over time, and the endemic equilibrium, a situation where a portion of the population will always consist of infectives [9, 12, 19]. The stability of these equilibria is determined by R_0, the basic reproduction number, which represents the average number of secondary infections generated by a single infectious case introduced into a completely susceptible population [1, 8, 14, 19, 12]. If $R_0 < 0$, then the disease free equilibrium will be stable and the epidemic will die out; if $R_0 > 0$ then a disease outbreak is certain, but if an endemic equilibrium can be proved to exist, then there will exist a threshold such that the disease will persist in the population [14, 9, 19, 12, 22]. Because of its important properties, the basic reproduction number is of interest when an epidemic model is proposed; in the case of the SIR model, $R_0 = \frac{N\beta}{v}$ [14], so we see that the parameters β and v are of particular importance for a disease following these transmission dynamics. In the case of a real world epidemic, taking measures to reduce the reproduction number below unity are vital in controlling any further spread of the disease, so understanding which values contribute to its growth is extremely valuable. Other important quantities in epidemic modeling include the contact number σ, representing the number of contacts resulting in successful disease transmission an average infective makes during the infectious period, and R, the average number of secondary infections produced by a typical infectious case over the entire period of infectiousness [19].

Once a model has been devised, it is often applied using parameters

gathered from actual disease outbreaks in order to test its validity. To solve these models, software is used to obtain numerical solutions which can be compared to actual recorded results [2, 16, 9]. Numerical solutions are preferred for solving these models since they are typically too complex to be solved analytically. Additionally, to aid in the mathematical analysis of these models, computer simulations of epidemic spread are used to confirm mathematical results and analyze the sensitivity of model parameters [19, 36, 9, 22].

The purpose of our research is to use error controlled software to find accurate numerical solutions to the differential equations arising in several compartmental epidemiological models. Software that implements adaptive error control returns an approximate solution which has an error consistent with user-prescribed error tolerances. From our observations, many previously published studies in this area of research do not employ error controlled software in their numerical simulations, inviting the possibility of erroneous results. It has become standard practice for ODE solvers to perform error control and as a result error control has become more common, but simulations using simple, non-error controlled methods can still be found. In the case of models based on PDE's, very few, if any, models are solved using error control software to control the spatial error; this is perhaps due to how few of these solvers exist. Numerical solutions in this field are often used to back up mathematical analysis of a disease model or in the analyses of properties of an epidemic, but using numerical methods that do not implement error control can lead to false positives or negatives.

In this report we apply error control software for time and time-space dependent epidemic models with the intent to show the advantage of error control software to solve problems in the domain of epidemiology. The *Scilab* problem solving environment *ode* function is applied in order to solve several standard time-dependent ODE-based models found in the literature. In order to solve several PDE reaction-diffusion systems we apply BACOLI [38], a recent B-spline collocation software package which provides adaptive spatial and temporal error control.

This report is organized as follows. We begin in Section 2 by introducing some necessary background material on mathematical epidemiological models of the kinds we are interested in as well as some background on the numerical techniques used to solve them. Information on ODE and PDE-based epidemiological models are provided in sections 2.1 and 2.2 respectively. Sections 2.3 and 2.4 then go on to describe the numerical methods we apply in solving these ODE and PDE-based models, as well as a description of the specific numerical software we apply. In Section 3, specific time-dependent ODE-based models of epidemic spread, solved using the *Scilab ode* function, are compared with those reported in the literature. Section 4 goes on to perform similar experimentation as in Section 3, in the context of PDE-based models dependent on both time and space. Section 5 gives our concluding

remarks and possible avenues of future research are discussed in Section 6.

2 Background Materials

2.1 ODE-based Epidemiological Models

Many mathematical epidemiological models, including the SIR model, are purely time dependent and do not take into account any population structure which may influence the transmission of an infectious disease. These ODE based epidemic models can be seen as modifications to the SIR model, developed in order to more closely represent the behavior of particular diseases [19, 4]. A simple extension is the SIRS model [14], which can be used in the modeling of diseases which confer temporary immunity to their hosts after they have been removed from the infectious class. Following this period of immunity, the individual then rejoins the susceptible class and may again become infected. Here we see the necessity of a transition from R to S, modeling the cyclic nature of diseases such as influenza where recurrent infections are common; this transition is controlled by a parameter γ, representing the duration of the immunity period. In addition, many models of infectious diseases add the consideration of certain vital dynamics of the host population; for example, birth and death rates may be represented, controlled by the parameters Π for birth rate, μ for the natural mortality rate (the rate of mortality in absence of disease) and δ for mortalities caused by the epidemic [28, 14, 24]. A similar kind of modified SIR model is used in [28]; this article considers the SZR model, a model suitable for representing an outbreak of zombies. A case study making use of this model is given in Section 3.1.

The susceptible, exposed, infected, recovered SEIR model expands on the Kermack-McKendrick model with the addition of a compartment containing members of the population who have been exposed to infection, but are currently in the incubation period of the disease [2, 19]. In this model, members of S transition to E rather than directly to I, and members of E transition to I at rate $1/\sigma$, the average incubation period of the disease. Adding this exposed class makes it possible to model diseases such as Ebola, Dengue fever, and Rabies, all diseases with incubation periods of varying length [2, 16, 29]. In Section 3.2, a case study on the 2014 Ebola outbreak in West Africa is discussed which makes use of the SEIR model.

Among the more complicated of these dynamic models are those which consider diseases spread by biting animals, which in epidemiology are called vectors. In these cases the models consider the dynamics of the host population, the vector population, and the interactions between the two. One classic example is the Ross-Macdonald model for the transmission of diseases between humans and mosquitoes. Here the human-mosquito transmission cycle works as follows: an uninfected human is bitten by an infected

mosquito and becomes infected, this infected human is then bitten by an uninfected mosquito which is subsequently infected [37]. These models are typically dependent on more variables since the population dynamics of both groups must be considered. In Section 3.2, a Ross-Macdonald model for Dengue fever is discussed which makes use of an SEIR model for the human population, and an SIR model for the vector population.

Within the realm of ODE-based epidemiological models, several other kinds have been devised in order to give a more accurate representation of the population dynamics of infectious diseases. Stochastic models introduce elements of randomness into the dynamics of disease transmission [24, 1]. Models can also consider diseases for which there is a treatment such as vaccination, allowing reduction in the size of the infected class faster than the natural rate [1, 28, 22]. Age structured epidemic models further subdivide the population based on age; the dynamics of each age group are then analyzed, which is useful for diseases such as measles, which has an infectivity largely dependent on age [1]. In many biological systems, including the population dynamics of epidemics, changes to the environment often have a delay before noticeable change occurs; to represent this, time delayed models of epidemiology have been devised which can be modeled using delay differential equations [5]. Network-based models consider the transmission of individuals between nodes representing specific areas members of the host population may move between (such as a hospital or place of work), with the movement between each node modeled using time dependent ODE's [35].

2.2 PDE-based Epidemiological Models

In order to better represent the dynamics of contact between the different populations relevant in an epidemic, it can be useful to develop models which can represent the spatial structure of a population. These models consider the heterogeneous mixing of a diseased population by making the number of individuals in each compartment dependent on both time and position in space. In making these models consistent with the biological phenomena, a logical requirement is that an infected individual is more likely to infect susceptibles who are close to them in space; to reflect this, spatial diffusion is allowed, thereby modeling the mobility of an actual population [1, 4]. One approach to represent these dynamics makes use of reaction-diffusion systems; models which implement systems of parabolic PDE's [4] and are generally significantly more complicated to solve than traditional ODE based models. Many of these models are Kermack-McKendrick type models which have been modified to include spatial dependence and diffusion; a simple example of an SIR model modified as such is given in [30]. These models have been used in studying the dynamics of multiple diseases including cholera [9], malaria [26, 4], and influenza [36]. The necessity of these models can be seen in situations where the places where infections occur are highly localized,

such as when a population center is geographically isolated, meaning that this area could have a much denser or less dense infectious population than surrounding areas; this has an impact on disease transmission that a non-spatial disease model could not capture.

In PDE Models, transmission between compartments is still largely controlled by a mass action transmission term for communicable diseases, but since the mixing here takes place over small regions of space it can account for differences in population density. This has an effect on the basic reproduction number R_0 due to the fact that an infected individual may produce more or less infections depending on their position in space. Spatial diffusion is associated with the second derivative of the solution component corresponding to each sub-population and a diffusion coefficient representing the rate at which members of each population diffuse through space. Many of the same modifications made to the non-spatial ODE compartmental models have also been applied to PDE-based models, including stochastic, age structured, and delayed models [25, 10, 40]. Another interesting modification here are mixed PDE and ODE models, which allow only certain components of a epidemic population to diffuse in space (for examples see [29, 7]). The models that we are interested in for this report are time dependent, deterministic models which consider one spatial dimension and permit spatial diffusion.

2.3 Numerical Solutions to ODE's

Differential equations are among the most important mathematical tools for describing any system which changes, and as such are applied in many fields such as science, engineering, and economics. ODE's are differential equations which involve functions of a single independent variable; in most applications this variable is time. The general form for an initial value ODE model is

$$\vec{y'} = \vec{f}(t, y), \quad \vec{y}(t_0) = \vec{y}_x. \tag{4}$$

for the purpose of our research, we consider systems of non-linear, coupled ODE's which arise in the study of compartmental epidemic models. Since most of these systems do not have explicit closed-form solutions, numerical software is employed to find approximate solutions for specified initial values.

When using numerical software to solve ODE's, there are four important factors to take into consideration regarding the numerical scheme implemented by the software; these are the local error, global error, stability and performance. Most numerical schemes for solving initial value problems (IVP's) rely on the concept of a step size h, the distance between discrete points $t_0, t_1, ..., t_n$ on the independent axis where the solution of the differential equation $y'(t, y)$ is approximated. For the i^{th} between t_i to t_{i+1}, we have a certain amount of error in our numerical solution; this is the local error and can be controlled by decreasing the size of h (that is, even if we were

to assume that the solution approximation at t_i was exact, there will still be an error associated with the solution approximation at t_{i+1}; this error is referred to as the local error) [6, 34]. Global error is the cumulative effect of the local error that has been generated on each step of the algorithm. At any point in time, it is simply the difference between the exact solution to the original ODE and the numerical solution at that point in time; therefore limiting the local error per step will also have an effect in reducing the global error [34].

A numerical algorithm for solving a differential equation is said to be unstable if small errors made while approximating the solution can result in "blow ups"; use of an unstable algorithm will lead to an approximate solution with a large amount of error disproportionate to the sum of the local errors [34]. Stability issues such as this can be avoided if appropriate measures are taken such as a reduction of step size, but this can come at a significant cost to performance; one such method which suffers significantly from this is Euler's method, one of the simplest and most common numerical methods [34, 6]. When Euler's method is used to solve the IVP $y' = \lambda y, \quad y(0) = y_0$ (where λ is a complex number with negative real part), its region of stability, the step sizes which will produce a stable solution, are limited by the requirement that $|1 + \lambda h| < 1$ [13]. This shows that depending on λ, the required step size may be forced to be extremely small, resulting in an expensive computation. In general, it is preferable to use numerical software which implements more sophisticated stable numerical schemes in order to limit the cost of computation while producing accurate output.

There are two main classes of numerical methods upon which most good quality ODE solvers are based, Runge-Kutta method [13, 6] and multistep methods [13, 6]. Runge-Kutta methods belong to the family of one-step methods which include Euler's method and are characterized by the fact that the numerical solution at t_n is solely determined from the previous approximate solution computed at t_{n-1}. In contrast, multistep methods make use of a number of previous numerical solution values $y_{n-k}, ..., y_{n-2}, y_{n-1}$ at times $t_{n-k}, ..., t_{n-2}, t_{n-1}$ respectively, in computing the numerical solution y_n.

In our research, the numerical solutions of IVP's we consider are obtained using the *ode* function provided in the *Scilab* problem solving environment [15]. The function interfaces with the LSODE [20] solver from ODEPACK [21], which automatically detects for stiffness in the problem and uses Adams methods [13] to solve non-stiff ODE's and BDF methods [13, 6] to solve stiff problems [20]. This function also provides error control capabilities; the default tolerances used for the relative and absolute error tolerances are 1×10^{-5} and 1×10^{-7} respectively; alternative tolerances can be specified by providing the optional parameters *rtol* and *atol*. The *ode* function adjusts the time step it takes so that at the end of each step, the numerical solution is accepted only if the associated error estimate is less than the user-provided

tolerance. By applying advanced, error controlled numerical software to standard time dependent models used in epidemiology, we expect to produce results with a higher degree of accuracy than would be obtained from simple schemes such as Euler's method with a fixed stepsize.

2.4 Numerical Solutions to PDE's

PDE's are equations that involve partial derivatives of functions that are dependent on more than one independent variable. These equations are frequently applied in situations where a system changes with respect to both time and space. With these additional dependence relations comes increased complexity in the methods used in approximating their solutions. These methods typically involve a discretization process where the system of PDE's is replaced with an approximating system of time dependent ODE's, this is known as the method-of-lines [34]. There are three main methods for performing the spatial discretization process: finite differences, finite elements, and collocation methods [34, 17]; here we focus on the collocation methods, as it is these methods we will be using in our numerical simulations.

For the numerical investigations undertaken in this report, the BACOLI software package [33] for solving one dimensional parabolic PDE's is employed to find numerical solutions to systems arising in epidemiology of the form:

$$\vec{u}_t(x,t) = \vec{f}(t,x,\vec{u}(x,t),\vec{u}_x(x,t),\vec{u}_{xx}(x,t)), \quad x_a \leq x \leq x_b, \quad t \geq t_0, \tag{5}$$

with the initial conditions

$$\vec{u}(x,t_0) = \vec{u}_0(x), \quad x_a \leq x \leq x_b, \tag{6}$$

and separated boundary conditions

$$\vec{b}_L(t,\vec{u}(x_a,t),\vec{u}_x(x_a,t)) = \vec{0}, \quad \vec{b}_R(t,\vec{u}(x_b,t),\vec{u}_x(x_b,t)) = \vec{0}, \quad t \geq t_0. \tag{7}$$

BACOLI approximates the solution to these PDE's using a B-spline collocation scheme (as described in [33]) . The spatial domain is partitioned into N sub-intervals determined by the spatial mesh points $x_a = x_0 < x_1 < ... < x_N = x_b$; the solution is then approximated as a linear combination of B-spline basis functions, peicewise polynomials of chosen degree p. The collocation positions are the set of $p-1$ Gaussian points mapped onto each sub-interval; C^1 continuity is imposed at each mesh point, providing a continuous and smooth curve. The approximate solution is determined by requiring it to satisfy the boundary conditions, and the PDE at the collocation points on each sub-interval. From here a system of ODE's with time dependent coefficients is generated, which together with the boundary conditions form a system of differential algebraic equations (DAE's), which is then solved in an error controlled computation using the DASSL [32] DAE

solver. DASSL adjusts the time-step and the order of the time stepping method so that at the end of each step, the numerical solution of the DAE system is accepted only if the associated estimate of the error (in time) is less than the user-prescribed tolerance.

BACOLI has an advantage over other PDE solvers in that it also performs adaptive spatial error control to ensure that an approximate solution is obtained for which an associated spatial error estimate is within a user defined error tolerance [33]. This allows more confidence in the approximation than typical PDE solvers provide. Here we employ BACOLI in order to attempt to reproduce results in the domain of epidemiology for models with time and space dependence. We attempt to reproduce results which are as accurate or more accurate than those obtained with non-error controlled PDE software in order to show the strength of these error controlled methods.

3 Applications of Error Control ODE Software to Time Dependent Epidemiological Models

3.1 Zombie ODE Model

The basic time dependent epidemic models mentioned in Section 2.1 have been applied in modeling many different and diverse diseases, including some in the realm of the supernatural. In 2009, Munz et al. wrote an article [28] detailing several models for zombie outbreaks with the goal of seeing under which circumstances humanity is most likely to survive this apocalyptic event. The types of outbreaks considered varied in scope, some of which were very basic while others considered what would happen when various additions were made to the model, such as a cure for the zombie infection or an initial aggressive human response to the outbreak. Here we will attempt to reproduce the results given in [28] for the most basic of these models.

The model for this epidemic is given as the SZR model, a modification of the SIR model which takes into account the unique characteristics of a zombie outbreak. This is done by dividing the population into the Susceptible (S), Zombie (Z), and Removed (R) classes. The vital dynamics of this host population are considered in this model, meaning birth and death rates have been taken into account with birth rate Π and non-zombie related mortality rate δ. Members of the susceptible class have a chance to become infected when an encounter with a zombie occurs (parameter β) and transition into class Z. Members of S can also be removed through natural causes at rate δ and transition into the Removed class, after which they can transition to class Z at rate ζ. Members of Z can only transition to class R after an encounter with a human which leads to their destruction (parameter α). This relationship is modeled by the following set of ODE's:

$$\frac{dS}{dt} = \Pi - \beta SZ - \delta S, \tag{8}$$

$$\frac{dZ}{dt} = \beta SZ + \zeta R - \alpha SZ, \tag{9}$$

$$\frac{dR}{dt} = \delta S + \alpha SZ - \zeta R. \tag{10}$$

Parameter	Value Used
α	0.005
β	0.0095
ζ	0.0001
δ	0.0001

Table 1: Zombie Model Parameters [28].

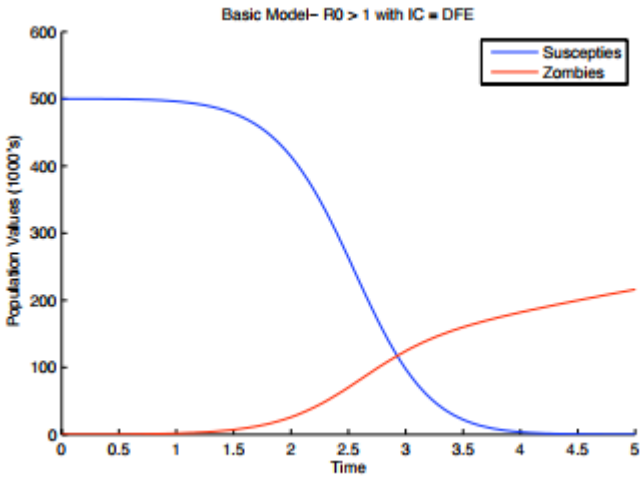


Figure 2: Zombie Model Original Numerical Results [28].

In [28], the authors used Euler's method with a fixed step size which was not identified, thus an exact reproduction of their numerical results could not be performed despite the fact that the source code was provided. For our purposes, we will use a more sophisticated method for solving this model, namely the *Scilab ode* function. Here we reproduce one of the results for this human-zombie population (shown in Figure 3 of [28]), which used initial conditions $(S, Z, R) = (500, 0, 0)$, and parameter values as given in Table 1. The original numerical result can be seen in Figure 2. In [28], it was also assumed that the birth rate was equal to the background death rate, therefore (7) can be rewritten as:

$$\frac{dS}{dt} = -\beta SZ. \tag{11}$$

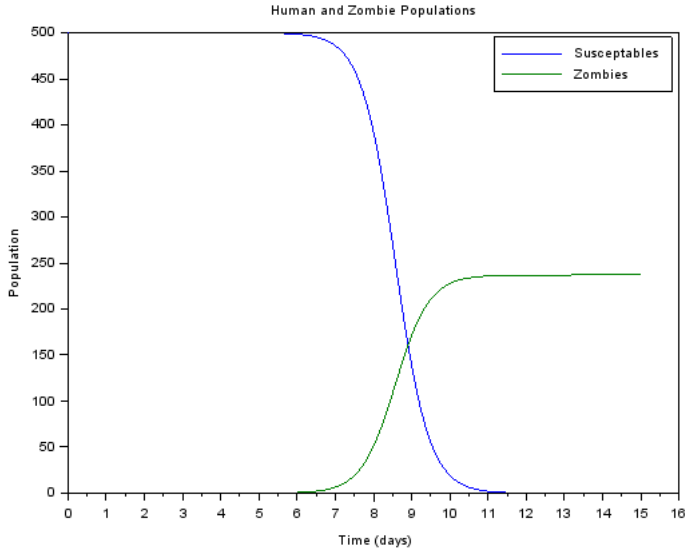


Figure 3: Plot of Human and Zombie Populations Over Time, as Computed Using the *Scilab ode* function.

The results for our simulation can be seen in Figure 3. Examining the result we obtain, we notice discrepancies between our results and those given in [28]. Most notably, the point at which the Susceptible and Zombie population curves intersect occurs when t is approximately equal to 9, which is outside the time domain given in [28].

In order to confirm this result, we also solved this problem in *Python* using the *scipy.integrate* module *ode* function which also implements a temporal error control scheme; the results (see Figure 4) were identical to those given by the *Scilab* script. The discrepancy between out numerical results and those reported in [28] may be due to the values for the parameters which are provided in [28] being different than those that were actually used in the original simulation, though in experimenting with changes to these parameters, we could not obtain the results given in [28]. The source code used to obtain our results, including both the *Sciliab* and *Python* implementations can be seen in Appendix A.

3.2 Ebola ODE Model

In 2014, an article was published with the purpose of using available data to gauge whether or not the control measures being employed in three African

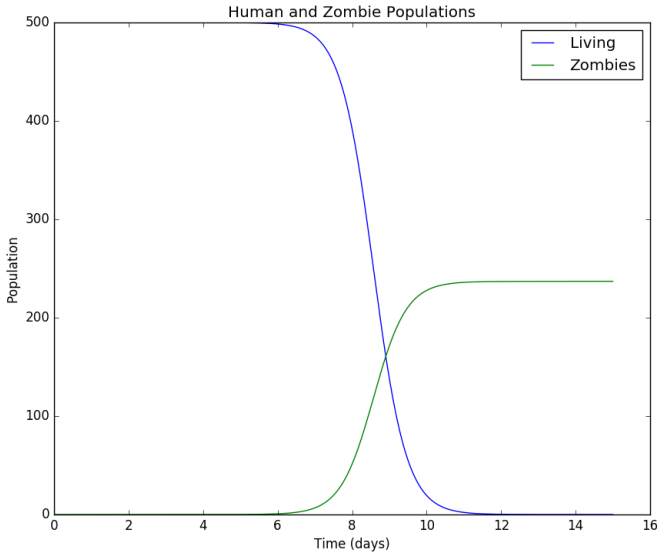


Figure 4: Plot of Human and Zombie Populations Over Time, as Computed Using the *scipy.integrate ode* function.

countries - Guinea, Seirra Leone, and Liberia - were effective in controlling the epidemics of Ebola during the 2014 outbreak in West Africa [2]. To model the transmission of Ebola, an SEIR model was used to account for Ebola's incubation period.

This model is represented by the following set of ODE's:

$$\frac{dS}{dt} = -\beta(t)SI/N, \tag{12}$$

$$\frac{dE}{dt} = \beta(t)SI/N - \sigma E, \tag{13}$$

$$\frac{dI}{dt} = \sigma E - \gamma I, \tag{14}$$

$$\frac{dR}{dt} = (1-f)\gamma I. \tag{15}$$

Additionally, the following equations were given, representing the rate of change of the number of the cumulative number of infected cases (C), as well as the total number of deaths (D):

$$\frac{dC}{dt} = \sigma E, \tag{16}$$

$$\frac{dD}{dt} = f\gamma I. \tag{17}$$

This system is dependent on several parameters: the population size (it was assumed that $N = 1,000,000$ for each country), the average incubation period of Ebola ($1/\sigma = 5.3$ days), its average duration of infection ($1/\gamma = 5.61$ days), the transmission rate at time t ($\beta(t)$), the rate at which control measures reduce transmission rate (k), and the fatality rate (f). The values of β, k, and f vary for each country and are given in Table 2. Also of note is that $\beta(t)$ is assumed to be constant before control measures are introduced, and after control measures are introduced, it is assumed that $\beta(t)$ decays exponentially. In [2], it is assumed that control measures are put into place immediately following the first infection; therefore we can express $\beta(t)$ as

$$\beta(t) = \beta e^{-kt}. \tag{18}$$

Using this information, we will attempt to reproduce the results given in [2] for the number of infections and number of deaths for each of the three African nations. In [2], the author states that this system of ODE's was solved numerically in the R software environment using the *deSolve* package *ode* function. However, the author did not give specific details regarding how this function was used; in particular the error tolerance which was employed for the computation is not indicated. For our attempted reproduction of these results we will use the default tolerance for the *ode* function. The initial conditions used in each case were $(S, E, I, R) = (N - 1, 0, 1, 0)$. The original results reported in the paper can be seen in Figure 5.

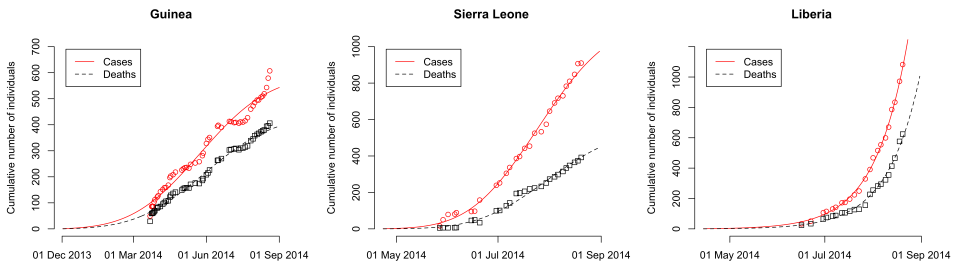


Figure 5: Ebola Model - Original Numerical Results [2].

Using the *Scilab's* built in *ode* function, we computed the solutions for the number of infected cases and number of deaths over time. The results for the Guinea, Sierra Leone, and Liberia Ebola outbreaks can be seen in Figures 6, 7, and 8 respectively.

Comparing these results to those plotted in the article, we see that they are either identical or close. We expect that any discrepancy in the result

Parameter	Guinea	Seirra Leone	Liberia
β	0.27	0.45	0.28
k	0.0023	0.0097	0.0
f	0.74	0.48	0.71

Table 2: Ebola Model Parameters For Each Country [2].

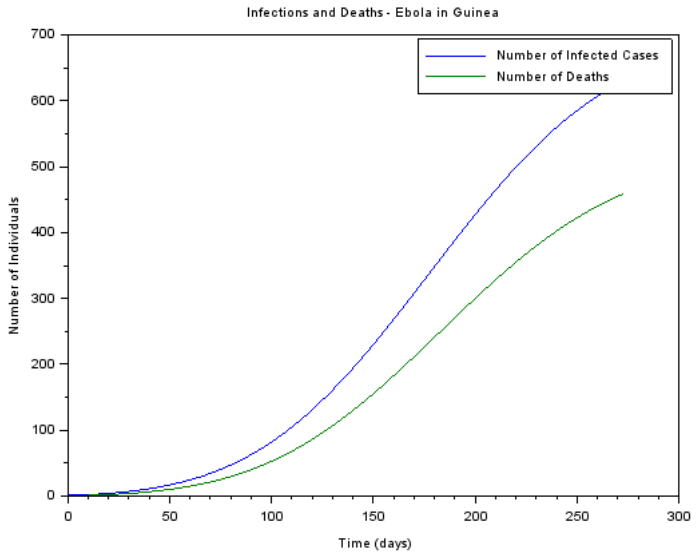


Figure 6: Guinea Infected Cases and Deaths.

may be due to differences in the algorithm or the error tolerance used in the computations associated with[2]. The *Scilab* source code used to obtain our results can be seen in Appendix B.

3.3 Dengue Fever ODE Model

Dengue fever is a vector transmitted disease which, with more than 50 million cases per year, is one of the most prominent mosquito borne illnesses [3]. In an article by Erickson et al. [16], a simple model for the possible transmission of Dengue was constructed, and verified using data from Lubbock, Texas. In this case, the vector population considered was the *Aedes albopictus* mosquito, the secondary vector of Dengue. We will attempt to reproduce these results here.

The model used is a Ross-MacDonald type SEIR-SEI model as described in section 2.1, which considers the dynamics of both human and vector populations. Additionally, the human birth rate, as well as the rate of human mortality from both dengue and other causes are considered.

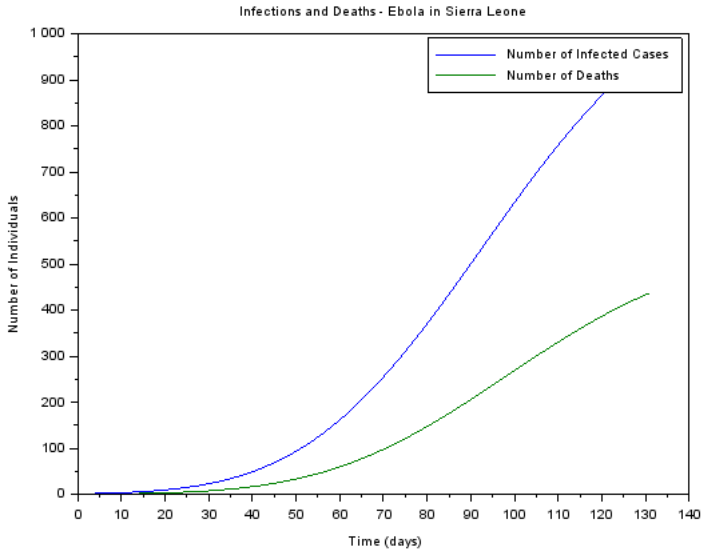


Figure 7: Seirra Leone Infected Cases and Deaths.

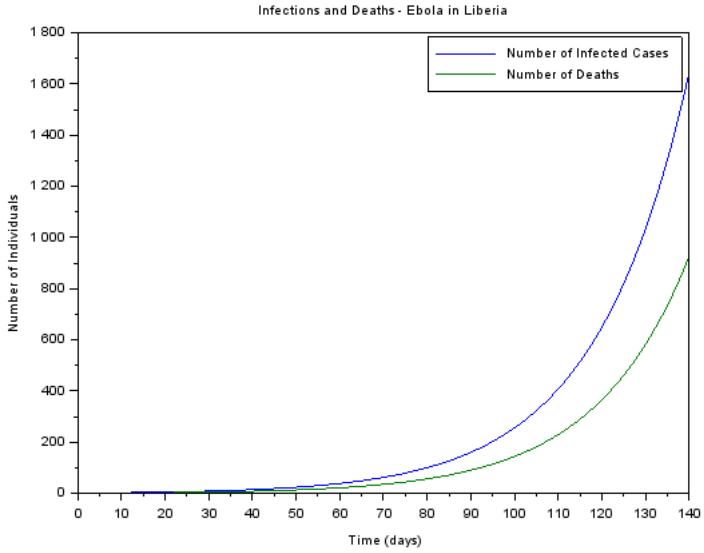


Figure 8: Liberia Infected Cases and Deaths.

The human SEIR population is divided into four components: H_s - the number of susceptible humans, H_e - the number of exposed, H_i - the number of infected, and H_r - the number of recovered. The dynamics for this human population are modeled using the following set of ODE's

$$\frac{dH_s}{dt} = \lambda H_t - H_s\left(\frac{cV_i}{H_t} + \mu_h\right), \tag{19}$$

$$\frac{dH_e}{dt} = H_s\frac{cV_i}{H_n} - H_e(\tau_{exh} + \mu_h), \tag{20}$$

$$\frac{dH_i}{dt} = H_e\tau_{exh} - H_i(\tau_{ih} + \alpha + \mu_h), \tag{21}$$

$$\frac{dH_r}{dt} = H_i\tau_{ih} - \mu_h H_r. \tag{22}$$

And the vector SEI population is divided into three components: V_s - the number of susceptible mosquitoes, V_e - the number of exposed mosquitoes, and V_i - the number of mosquitoes. The system modeling the vector population is:

$$\frac{dV_s}{dt} = \mu_a V_t - V_s\left(\frac{cH_i}{H_t} + \mu_a\right), \tag{23}$$

$$\frac{dV_e}{dt} = V_s\frac{cH_i}{H_t} - V_e(\tau_{exv} + \mu_a), \tag{24}$$

$$\frac{dV_i}{dt} = V_e\tau_e xv - \mu_a V_i. \tag{25}$$

Parameter	Meaning	Value Used
λ	Human population growth rate	2.244 x 10^{-5}
μ_h	Human mortality rate	1/28000
c	Contact probability	0.12
τ_{exh}	Inverse exposure time, host	1/10
α	Dengue host mortality rate	0.003
μ_a	Adult mortality	1/20
τ_{ih}	Inverse infective time, host	1/4
H_n	Initial number of humans	10000

Table 3: Dengue Model Parameters [16].

The parameters used in this model are explained and their values shown in Table 3. The initial conditions used for the human population were $(H_s, H_e, H_i, H_r) = (10000, 0, 0, 0)$, and for the vector population $(V_s, V_e.V_i) = (100000, 0, 10000)$. We will use this information in order to attempt to reproduce the results given in [16] for changes in the human and mosquito populations over time as determined by the model. In [16], the author states that the solution was computed using *Mathematica*, but did not specify how the software was used or give any indication that error control was used. The original numerical results reported in [16] can be seen in Figure 9. We used the *Scilab* function *ode* in our computation with its default error

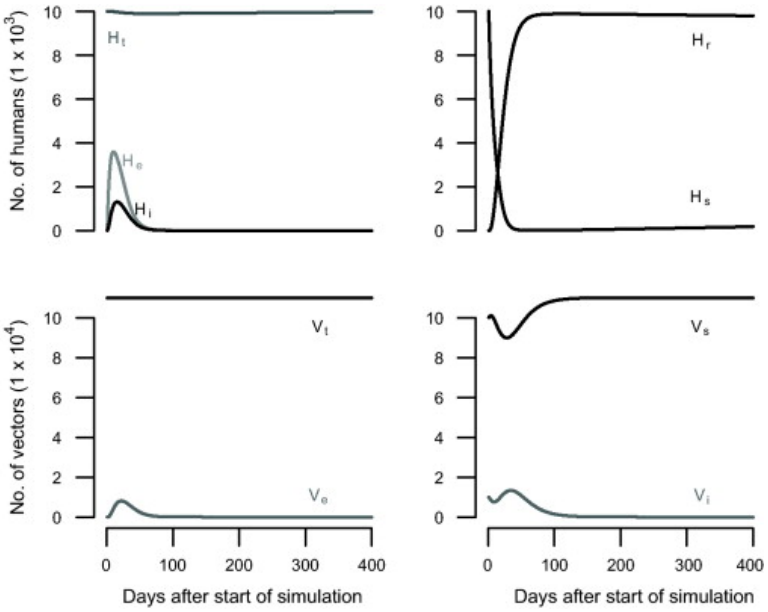


Figure 9: Dengue Model - Original Numerical Results [16].

tolerance; the script can be seen in Appendix C. The results we obtained are shown in Figure 10 for the human population, and Figure 11 for the vector population. Note that the original results for both the human and vector population were split into two graphs each, which have been combined into one in our results.

Comparing the plotted to results to those found in the article, we see that they are identical and therefore our reproduction of this model has been successful. Thus our results support the authors argument made in [16] that this model does give an accurate representation of Dengue fever spread through this host population.

4 Applications of Error Controlled PDE Software to Time-Space Epidemiological Models

4.1 Spatial SI Model

In 2014, Lofti et al. [27] published a mathematical analysis of a general reaction-diffusion SI epidemic model. The model purposed in [27] is relevant for communicable diseases, transmitted via direct contact by an infectious individual with a susceptible one, such as influenza, measles and rabies; in [27] the global dynamics of this model are analyzed. In particular, the global asymptotic stability of its disease free and endemic equilibria are analyzed both mathematically and numerically. Here we attempt to reproduce the

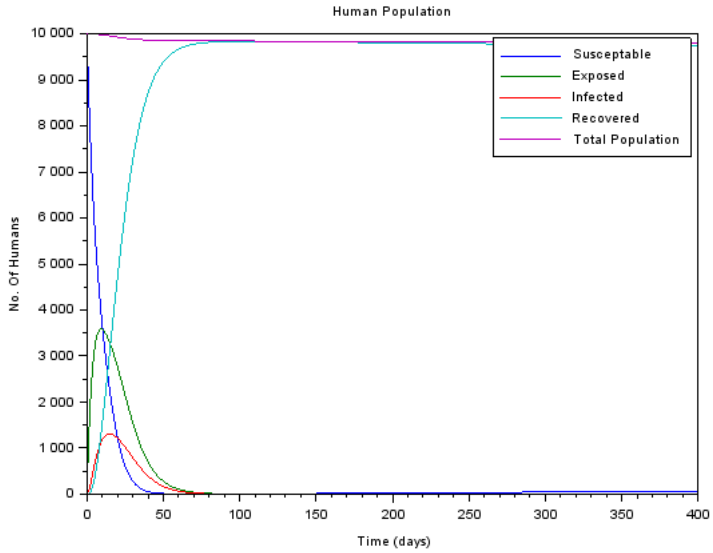


Figure 10: Plot of the human populations Susceptible (H_s), Exposed (H_e), Infected (H_i), Recovered (H_r), and Total (H_t) Populations.

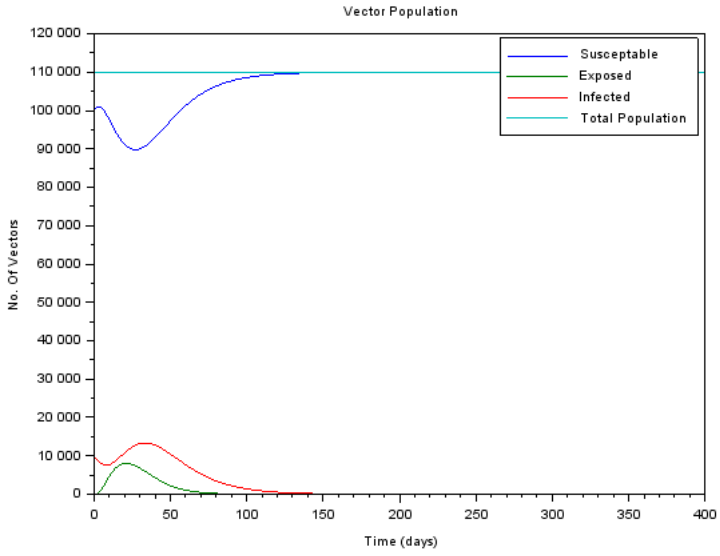


Figure 11: Plot of the *Aedes albopictus* Populations Susceptible (V_s), Exposed (V_e), Infected (V_i), and Total (V_t) Populations.

numerical simulations, which were used as confirmation for the mathematical results proven in [27]. The system is given as

$$\frac{\partial S}{\partial t} = d_S \Delta S + \Lambda - \mu S(x,t) - \frac{\beta S(x,t) I(x,t)}{1 + \alpha_1 S(x,t) + \alpha_2 I(x,t) + \alpha_3 S(x,t) I(x,t)}, \tag{26}$$

$$\frac{\partial I}{\partial t} = d_I \Delta I + \frac{\beta S(x,t) I(x,t)}{1 + \alpha_1 S(x,t) + \alpha_2 I(x,t) + \alpha_3 S(x,t) I(x,t)} - (\mu + \delta + r) I(x,t), \tag{27}$$

where Δ is the Laplacian operator. The boundary conditions are

$$\frac{\partial S}{\partial v} = \frac{\partial I}{\partial v} = 0, \quad t > 0, \quad x \epsilon [0,1], \tag{28}$$

and the initial conditions are

$$S(x,0) = \begin{cases} 1.1x, & 0 \leq \text{x} < 0.5, \\ 1.1(1-x), & 0.5 \leq \text{x} \leq 1, \end{cases}$$

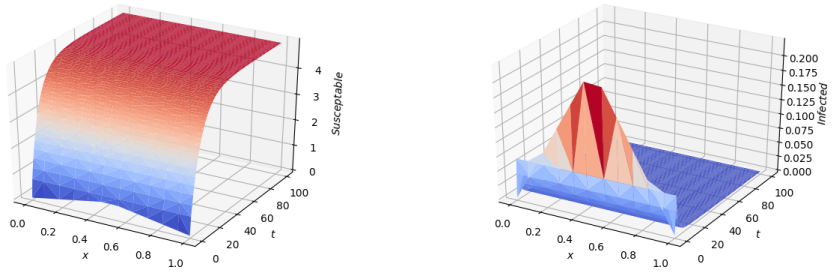
$$I(x,0) = \begin{cases} 0.5\text{x}, & 0 \leq \text{x} < 0.5, \\ 0.5(1-x), & 0.5 \leq \text{x} \leq 1. \end{cases} \tag{29}$$

The parameters used in this system and their explanations are given in Table 4. From the initial conditions given here, the highest concentration of the population is at the midpoint of the spatial domain, where $x = 0.5$; as time progresses we would expect the larger population at $x = 0.5$ to spread throughout the spatial domain.

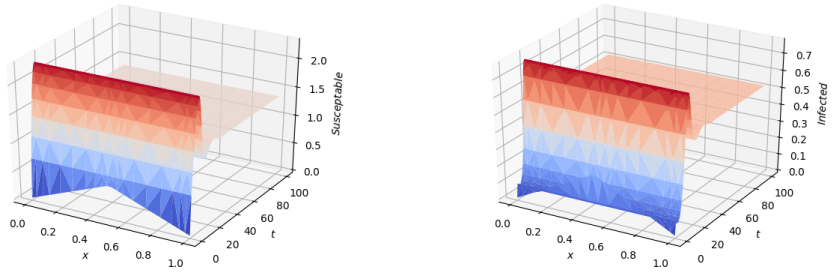
We now attempt to reproduce the numerical solutions found in [27]; no reference was made in [27] to the software used to produce their numerical results, thus the accuracy of the provided numerical results is unclear. A numerical approximation of the solution to this system was found with BACOLI using absolute and relative error tolerances of 10^{-5}. Note that the given initial conditions are not smooth (i.e., the first derivative is discontinuous) at the midpoint of the spatial domain; it is known that this can cause issues for BACOLI. From past experience, we know that this issue can be handled by placing a spatial mesh-point on this non-smooth point. The results from BACOLI are plotted in Figure 12. We see consistency with the results given in the paper, which provides additional confidence regarding the validity of the originally published numerical results. The scripts used for computing the results using BACOLI and plotting those results are given in Appendix D.

4.2 Spatial Influenza Model

Influenza is one of the most prevalent of the infectious diseases, with seasonal reoccurring outbreaks affecting large portions of their host population each year. Though it is seemingly harmless in most cases, influenza takes a



(a) Susceptable (left) and Infected (right) Populations when $\beta = 0.2$



(b) Susceptable (left) and Infected (right) Populations when $\beta = 0.6$

Figure 12: Numerical Results for the Reaction Diffusion SI Model (25) - (28).

large toll on healthcare systems and causes many deaths each year. In Samsuzzoha et. al [36] a spatial SEIR model is considered which modifies an existing ODE model to include spatial dependence and diffusion. This reaction-diffusion PDE model is as follows:

$$\frac{\partial S}{\partial t} = -\beta \frac{E+I}{N} S - \mu S + rN(1 - \frac{N}{K}) + d_1 \frac{\partial^2 S}{\partial x^2}, \tag{30}$$

$$\frac{\partial E}{\partial t} = \beta \frac{E+I}{N} S - (\mu + \sigma + \kappa) E + d_2 \frac{\partial^2 E}{\partial x^2}, \tag{31}$$

$$\frac{\partial I}{\partial t} = \sigma E - (\mu + \alpha + \gamma) I + d_3 \frac{\partial^2 I}{\partial x^2}, \tag{32}$$

$$\frac{\partial R}{\partial t} = \kappa E + \gamma I - \mu R + d_4 \frac{\partial^2 R}{\partial x^2}, \tag{33}$$

Parameter	Meaning	Case 1	Case 2
d_S	Diffusion rate for susceptible class	0.1	0.1
d_I	Diffusion rate for infected class	0.5	0.5
Λ	Recruitment rate of host population	0.5	0.5
μ	Natural death rate of the population.	0.1	0.1
d	Rate of death due to disease	0.1	0.1
r	Recovery rate of infected individuals	0.5	0.5
α_1	Incidence rate of S	0.1	0.1
α_2	Incidence rate of I	0.02	0.02
α_3	Incidence rate of R	0.03	0.03
β	Infection rate	0.6	0.2

Table 4: Reaction Diffusion SI Model Parameters [27].

on the spatial domain $x \epsilon [-2, 2]$ with boundary conditions:

$$\frac{\partial S(-2,t)}{\partial x} = \frac{\partial E(-2,t)}{\partial x} = \frac{\partial I(-2,t)}{\partial x} = \frac{\partial R(-2,t)}{\partial x} = 0, \tag{34}$$

$$\frac{\partial S(2,t)}{\partial x} = \frac{\partial E(2,t)}{\partial x} = \frac{\partial I(2,t)}{\partial x} = \frac{\partial R(2,t)}{\partial x} = 0. \tag{35}$$

The stability of this diffusion models equilibria are shown mathematically and the solution of this system is approximated using an operator splitting method, which splits the PDE system into two subsystems, a system of non-linear reaction equations applied to the first half of the time step and a system of linear diffusion equations for the second half. These subsystems were then solved numerically to simulate the spread of influenza in populations implementing control strategies with varying degrees of success; the control strategies are represented by varying the parameters β and v, the transmission and recovery rates respectively. In order to demonstrate how the addition of spatial diffusion affects the spread of an epidemic, the same problems were simulated with and without diffusion. For their numerical simulations, the authors used Euler's method to solve the derived systems with a fixed time step; there is, therefore, the potential for substantial error in these numerical results.

We will attempt to reproduce one of the results given in [36] for the diffusive model considered in their third case, which had parameters given as in Table 5, and the initial conditions:

$$S(x,0) = 0.96e^{-10x^2}, \tag{36}$$

$$E(x,0) = 0, \tag{37}$$

$$I(x,0) = 0.04e^{-100x^2}, \tag{38}$$

$$R(x,0) = 0. \tag{39}$$

From these initial conditions we can observe that we have a host population which is most dense at the center of the spatial domain, $x = 0$, and which gets progressively more diffused near the endpoints of the spatial domain. The original numerical results for this model can be seen in Figure 13.

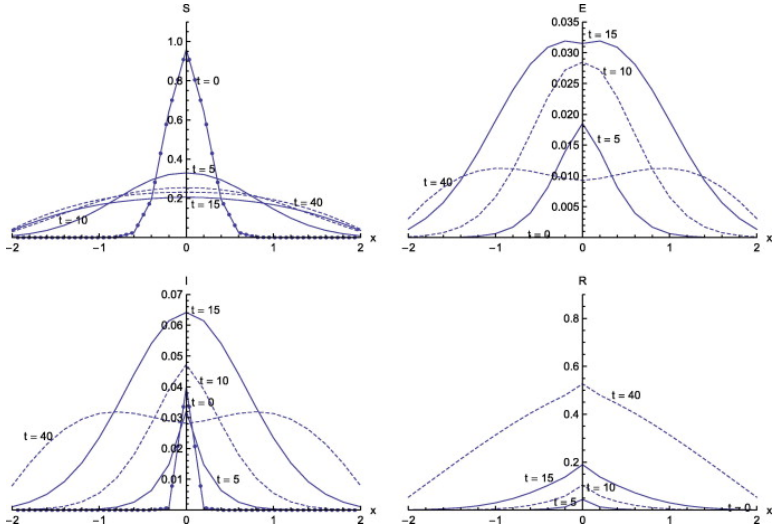


Figure 13: Influenza Model - Original Numerical Results [36].

Here this reaction-diffusion PDE system is solved using BACOLI, with relative and absolute error tolerances of 10^{-6}, giving us good confidence in the accuracy of our approximate solution. The result of our simulation can be seen in Figure 14. Comparing our solutions with those given in the [36], we see similar behavior for the times 5, 10, and 15, but our solution differs greatly when t is 40.

To further investigate these results, this system was then solved using the *Maple pdsolve* function with an absolute error tolerance of 10^{-5}. The *pdsolve* function provides error checking using estimation of the spatial and temporal errors. The spatial error estimate is obtained by computing the solution twice, first on the fixed initial mesh provided to the function and again on a courser mesh; the two solutions are then compared to obtain this estimate. Estimation of the temporal error is computed by comparing the solution computed using at the fixed time step with another computed with a slightly larger time step but with the same spatial mesh. If this error estimate is greater than the specified tolerance then the computation is terminated; in order for the error estimates to be within the user tolerance a dense spatial mesh and small step sizes must be used. Our computation was performed with spacestep = 0.001 and timestep = 0.001 passed as arguments to *pdsolve*. This computation produced numerical solutions similar to those we obtained with using BACOLI. We hypothesize that the reason for the

disagreement of our results with those of [36] is due to the authors use of a fixed step Euler's method. The scripts used in obtaining our results can be seen in Appendix E.

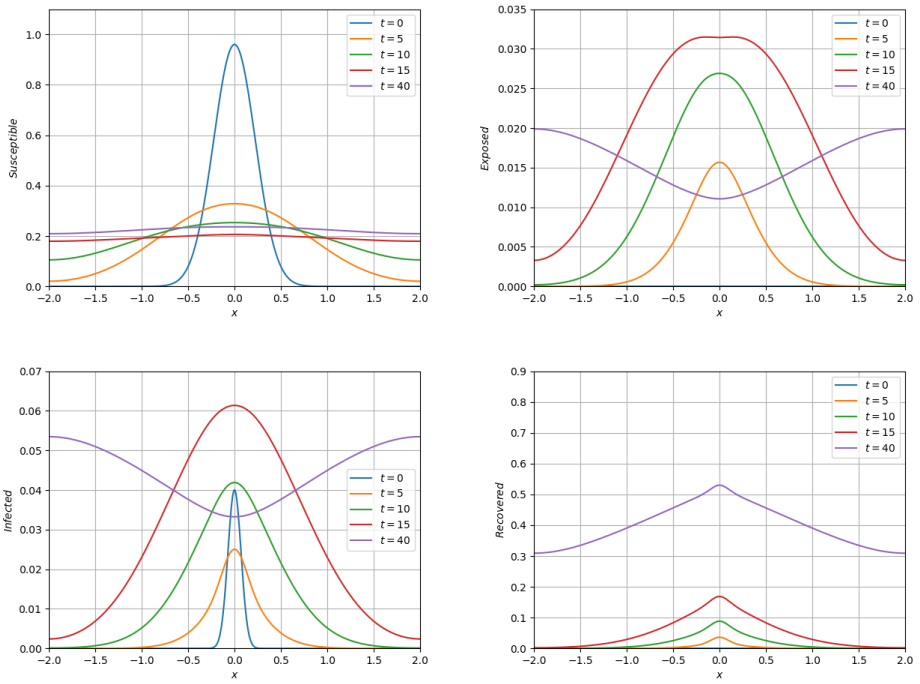


Figure 14: Numerical Results for the Reaction Diffusion Influenza Model

4.3 Spatial Cholera Model Applied to 2010-2012 Haiti Outbreak

Cholera is an infectious bacterial disease which is most commonly transmitted via food and water contaminated with the *Vibrio cholerae* bacteria, and much more rarely, direct human contact [9]. Many developing countries experience cholera epidemics due to insufficient access to clean water. When someone is infected with cholera, they suffer from diarrhea and vomiting which can cause dehydration, which can be fatal if the infected person cannot be re-hydrated with clean water. Due to the nature of the illness, a diseased patient can cause further contamination of the water source, which will in turn lead to more infections if proper precautions have not been put in place to prevent further consumption from the contaminated water source. Epidemic models representing cholera outbreaks often consider an SIR human population with the addition of a compartment representing the population of the bacteria, B [9, 39]. With these models it is useful to use the spatial domain to represent the contaminated water source, in this way

Parameter	Meaning	Value Used
β	Transmission coefficient.	0.514
μ	Natural mortality rate.	5.5×10^{-5}
r	Birth rate.	0.0714
K	Carrying capacity.	1.0
σ	Duration of latent period.	0.5
κ	Recovery rate of latents	0.1857
α	Flu induced mortality rate.	0.0093
γ	Recovery rate for clinically ill.	0.20
d_1	Diffusion coefficient for S.	0.5
d_2	Diffusion coefficient for E.	0.025
d_3	Diffusion coefficient for I.	0.001
d_4	Diffusion coefficient for R.	0.0

Table 5: Reaction Diffusion Influenza Model Parameters [36].

the infectiousness of the disease is related to the concentration of susceptible humans and bacteria at a given point along the water source.

A cholera model is given by Capone et. al [9] which represents the spread of Cholera over a spatial domain; the existence and stability of the disease free and endemic equilibria are shown and numerical simulations performed in order to support these results. The SIBR cholera model is

$$\frac{\partial S}{\partial t} = \mu(N_0 - S) + \gamma_1 \Delta S - \beta\lambda(B)S, \tag{40}$$

$$\frac{\partial I}{\partial t} = \beta\lambda(B)S - (\sigma + \mu)I + \gamma_2 \Delta I, \tag{41}$$

$$\frac{\partial B}{\partial t} = eI - (\mu_b - \pi_b)B + \gamma_3 \Delta B, \tag{42}$$

$$\frac{\partial R}{\partial t} = \sigma I - \mu R + \gamma_4 \Delta R, \tag{43}$$

where $\lambda(B)$,the probability of catching cholera, is

$$\lambda(B) = \frac{B}{K_B + B}, \tag{44}$$

and with the boundary conditions on domain $x = [0, 1]$:

$$\frac{\partial S}{\partial t} = \frac{\partial I}{\partial t} = \frac{\partial B}{\partial t} = \frac{\partial R}{\partial t} = 0. \tag{45}$$

In order to represent the effect of control strategies as well as demonstrate the asymptotic stability of the equilibria, various simulations were performed which vary the parameter β, the contact rate with contaminated water, the

diffusion terms for the populations and the initial conditions. Note that this model only considers transmission of Cholera from a water source to a human and considers no representation for human to human transmission. For these simulations, the authors state that the spatial domain was restricted to consider only one independent variable, x. The parameters chosen for these simulations came from the 2010-2012 cholera outbreak in Haiti, so that the numerical results could be compared to the true results recorded during that epidemic.

Here we attempt to reproduce two of the numerical results given in [9] using BACOLI, with the parameters used for each as given in Table 6. In simulation 1, the constant initial conditions used were given as $(S, I, R, B) = (2989.29, 9.17889, 278.148, 701.53)$; for simulation 2, the initial conditions are given as:

$$S(x,0) = 3500, \tag{46}$$

$$I(x,0) = \begin{cases} 100\pi cos(\pi x) & 0 \leq x \leq 0.5, \\ 0 & 0.5 < x \leq 1, \end{cases} \tag{47}$$

$$B(x,0) = \begin{cases} 10\pi cos(\pi x), & 0 \leq x \leq 0.5, \\ 0 & 0.5 < x \leq 1, \end{cases} \tag{48}$$

$$R(x,0) = 100. \tag{49}$$

The original numerical results provided in [9] for simulation 1 can be seen in Figure 15; the results for simulation 2 can be seen in Figure 16.

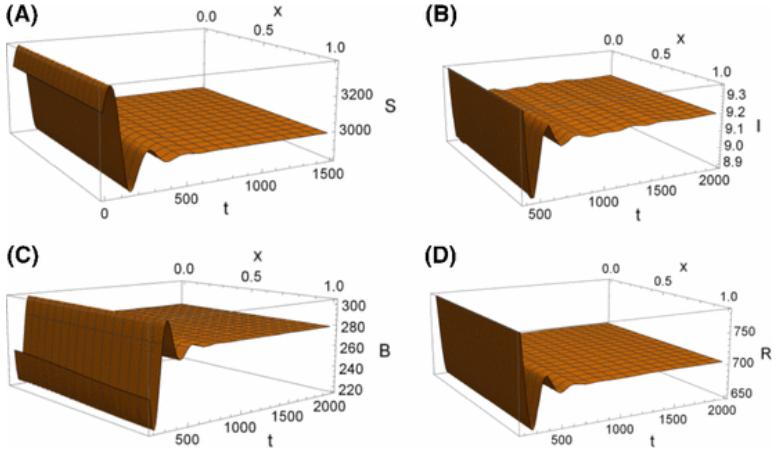


Figure 15: Cholera Model Simulation One - Original Numerical Results [9].

For simulation 1, we compute an approximate numerical solution using BACOLI with absolute and relative error tolerances of 10^{-7}. Plotting these results, we see good agreement with those given in [9], with the exception of the beginning of the time domain, where in the results obtained for each

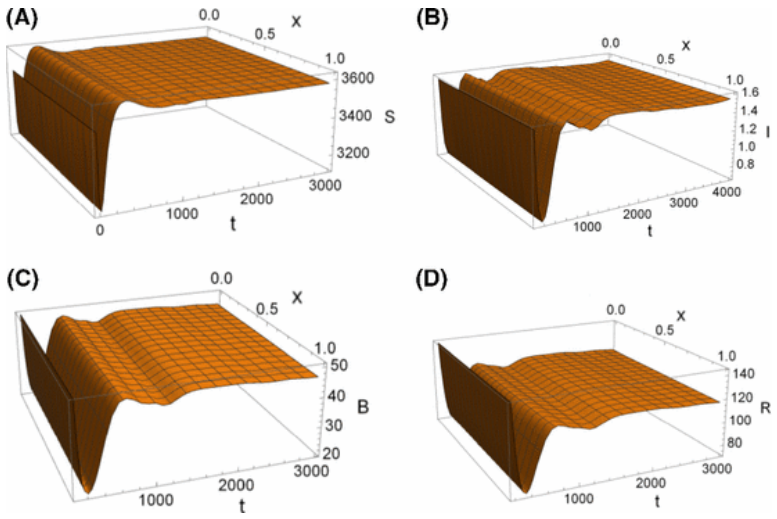


Figure 16: Cholera Model Simulation Two - Original Numerical Results [9].

portion of the disease population we see disagreement. The plot for simulation 1 can be seen in Figure 17; we also computed the solution using the *Maple* problem solving environment *pdsolve* function, which produces results consistent with those produced with BACOLI. This may indicate that some element of the PDE system used to produce the result as given in [9] for this case differs from what was described in [9].

In simulation 2, due to the non-smooth initial conditions, we choose the relatively large absolute and relative error tolerances of 10^{-4}; for sharp tolerance choices, the integration with BACOLI fails (that is, BACOLI was not able to obtain an error controlled numerical solution to within the specified tolerance). Additionally, the initial spatial mesh points have been chosen to include the point $x = 0.5$ to accommodate the non-smooth initial conditions. Examining the plot generated from our simulations in Figure 18, we see drastic differences between the results we obtain and those provided in [9]. The most notable differences which can be seen are the behaviors of the I and B populations at the beginning of the time domain, which rather than exhibiting a horizontal line as seen in [9], show a steep curve which changes across spatial domain from $x = 0$ to $x = 1$ from a high peak value down to zero. However, based on the initial values for this system, this behavior is unsurprising. This indicates that the initial conditions differ from those used in the simulation; from observation it seems likely that constant initial conditions where used rather than the spatially dependent ones presented in the paper [9].

In [9], we observe issues in the numerical results for two cases. While both of the results demonstrated asymptotic stability and therefore help to confirm the primary result of the paper, the simulations used to support

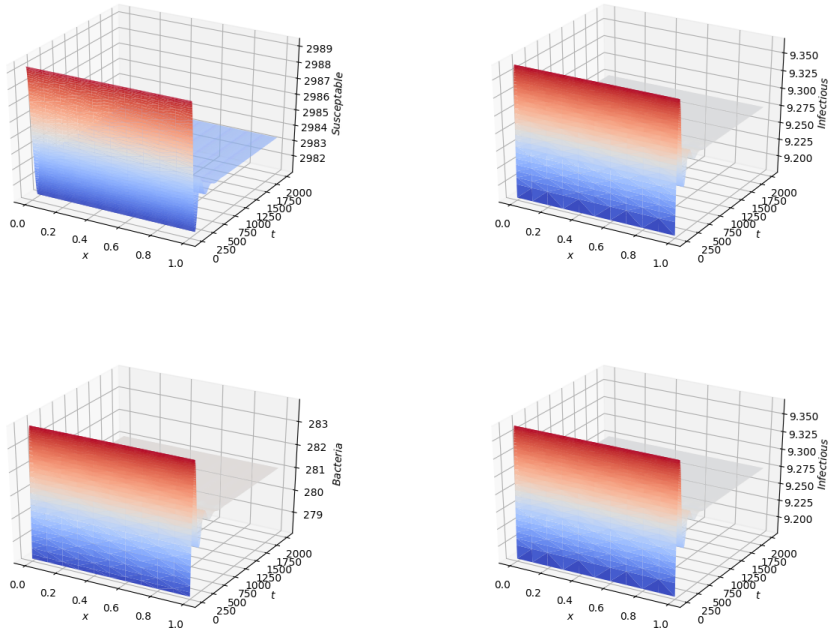


Figure 17: Numerical Results of Cholera Simulation 1

these results are problematic. All source code used in obtaining our results can be found in Appendix F.

5 Conclusions/Related Work/Future Work

5.1 Conclusion

In this report, we have applied error controlled numerical software to a variety of ODE and PDE based mathematical models from the domain of epidemiology. This was done in order to attempt to reproduce numerical results which have been presented in published literature in this field, and in doing so compare these results to those produced using error control ODE and PDE solvers. One pattern which emerged while performing this research is that many authors do not provide the source code they have used, specify which numerical methods have been used in solving these models, or provide complete information about important factors in how their simulation was performed, such as the prescribed error tolerance or the fixed step size they have used for Euler's method. As a consequence, it is often difficult or impossible to faithfully reproduce these numerical results, and when discrepancies in results do arise, it is difficult to pinpoint their cause.

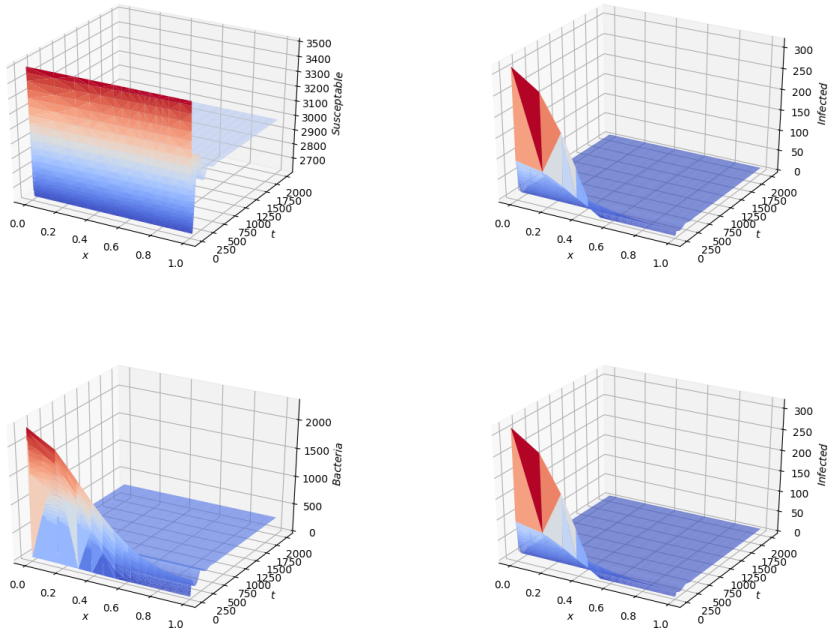


Figure 18: Numerical Results of Cholera Simulation 2

5.2 Related Work

Among the literature we have considered over the course of this research, a common trend we have observed among domain experts is the use of either high-level problem solving environments such as MATLAB, Maple, or Mathematica to produce numerical simulations. These environments give users the advantage of being able to obtain high-quality numerical results by writing quick scripts in a familiar, easy-to-use environment, which, for the convenience it provides, can, for many users be worth the performance cost associated with the use of these environments. To accommodate this usage case, we have recently developed a Python interface to BACOLI, packaged as a part of scikits, a distribution of scientific tools bundled as add-on packages for the SciPy scientific computing library [41]. This package is called scikits.bacoli_py; in its development, the f2py Python to Fortran interface generator [31] was used to create an interface module between the calling Python programs and a Fortran interface for the BACOLI software package. The version of BACOLI used in bacoli_py has undergone slight modifications, the most substantial change being the replacement of the COLROW [11] linear system solver with LAMPACK [23] in order to ensure BSD license compatibility.

This Python wrapper was designed with ease of use in mind, with an in-

Parameter	Meaning	Case 1	Case 2
N_0	Total population size at t=0	3700	3700
γ_1	Diffusion coefficient for susceptible class	0.8	0.8
γ_2	Diffusion coefficient for infected class	0.003	0.1
γ_3	Diffusion coefficient for bacteria class	0.002	0.01
γ_4	Diffusion coefficient for recovered class	0.5	0.5
μ	Birth/Death rate	0.014	0.014
σ	Recovery rate	1.0678	1.0678
μ_B	Loss rate of bacteria	1.06	1.06
π_B	Growth rate of bacteria	0.73	0.73
e	*V. cholerae* produced by one infected	10	10
β	Contact rate with contaminated water	1.2	1.0
K_B	Half saturation rate	10^5	10^5

Table 6: Cholera Model Parameters [9].

terface which may be quickly grasped by those familiar with other high-level PDE solver implementations such as MATLAB's *pdepe* solver. However, interfacing with Python in this way has come at a fairly substantial cost to performance. From our tests it seems that when both the Python and pure Fortran implementations are used to solve the same problem, the Python implementation performs the computation in at best approximately twice the time of the Fortran implementation. To our current knowledge, there is no other PDE solver available for the Python language which implements the type of spatial error control available in this package.

5.3 Future Work

In order to accommodate the needs of domain experts, the BACOLI software package may be expanded in order to provide error controlled solutions to more classes of problems. Currently work is being done to allow it to handle mixed PDE-ODE models of the type we have seen in the literature [18]. Modifications may be also be made to accommodate time delayed and fractional PDE systems.

Appendices

Appendix A: Zombie Modeling Scripts

```
//Solving the system of ODE's representing the spread of zombification through
//a population. Uses modified SIR ODE model, the SZR model.
//Parameters Used:
//      a -- alpha in article -- zombie destruction rate
//      b -- beta in article -- new zombie rate
//      z -- zeta in article -- resurrection rate
//      d -- delta in article -- natural death rate
//      N -- initial human population

a = 0.005;
b = 0.0095;
z = 0.0001;
d = 0.0001;
N = 500;

function ydot = odes(t, y)
    ydot = zeros(y);
    //S
    ydot(1) = -b*y(1)*y(2);
    //Z
    ydot(2) = b*y(1)*y(2) - a*y(1)*y(2) + z*y(3);
    //R
    ydot(3) = a*y(1)*y(2) + d*y(1) - z*y(3);
endfunction

t = 0:0.005:15;
y0 = [N; 0; 0];
t0 = 0;
y = ode(y0, t0, t, odes);
y

plot(t, y(1,:), t, y(2,:));
xtitle('Human and Zombie Populations', 'Time (days)', 'Population');
legend('Susceptables', 'Zombies');
```

```
# Solving the set of ODE's representing the spread of zombification through
# a population. Uses modified SIR ODE model, the SZR model.
# Parameters Used:
#   a -- alpha in article -- zombie destruction rate
#   b -- beta in article -- new zombie rate
#   z -- zeta in article -- resurrection rate
#   d -- delta in article -- natural death rate
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
plt.ion()
plt.rcParams['figure.figsize'] = 10, 8

d = 0.0001  # natural death percent (per day)
b = 0.0095  # transmission percent  (per day)
z = 0.0001  # resurect percent (per day)
a = 0.005  # destroy percent  (per day)

# solve the system dy/dt = f(y, t)
def f(y, t):
     Si = y[0]
     Zi = y[1]
     Ri = y[2]
     # the model equations (see Munz et al. 2009)
     f0 = -1*b*Si*Zi
     f1 = b*Si*Zi + z*Ri - a*Si*Zi
     f2 = d*Si + a*Si*Zi - z*Ri
     return [f0, f1, f2]

# initial conditions
S0 = 500.                # initial population
Z0 = 0                   # initial zombie population
R0 = 0                   # initial death population
y0 = [S0, Z0, R0]      # initial condition vector
t  = np.linspace(0, 15., 3000)         # time grid

# solve the DEs
soln = odeint(f, y0, t)
S = soln[:, 0]
Z = soln[:, 1]
R = soln[:, 2]
```

```
# plot results
plt.figure()
plt.plot(t, S, label='Living')
plt.plot(t, Z, label='Zombies')
plt.xlabel('Time (days)')
plt.ylabel('Population')
plt.title('Zombie Apocalypse')
plt.legend(loc=0)

plt.savefig("temp.png")
```

Appendix B: Ebola Modeling Script

```
//Script to solve the system of ODE's for the model used
//for the spread of Ebola in three countries, in edition to ODE's
//representing the number of cases and the number of
//deaths over the course of the outbreak. The solution
//for the number of cases and deaths are then plotted.
//Parameters:
//      n - the population size.
//      inc - the average incubation period of Ebola.
//      inf - the average infection duration of Ebola.
//      b - the transmission rate prior to control measures.
//      k - the rate at which contol measures reduce transmission rate.
//      f - the fatility rate.
//      bt - the transmission rate at time t after control measures.

n=1000000;
inc = 5.3;
inf = 5.61

//k --
//For Guinea
k = 0.0023;
//For Sierra Leone:
//k = 0.0097;
//For Liberia:
//k = 0.0;

//f --
//For Guinea:
f = 0.74;
//For Seirra Leone:
//f = 0.48;
//For Liberia:
//f = 0.71;

//b --
//For Guinea:
b = 0.27;
//For Sierra Leone:
//b = 0.45;
//For Liberia:
//b = 0.28;

function ydot=odes(t, y)
    ydot=zeros(y);
    bt = b*\%e^(-k*t);
    ydot(1) = -1*bt*y(1)*y(3) / n;
    ydot(2) = bt*y(1)*y(3)/n - (1/inc * y(2));
    ydot(3) = (1/inc)*y(2) - (1/inf)*y(3);
    ydot(4) = (1 - f)*(1/inf)*y(3);
    ydot(5) = (1/inc) * y(2);
    ydot(6) = f * (1/inf) * y(3);
endfunction

//t's --
//For Guinea:
t = 0:1:273;
//For Seirra Leone:
//t = 0:1:131;
//For Liberia:
//t = 0:1:140;

y0 = [n-1; 0; 1; 0; 1; 0];
t0 = 0;
y = ode(y0, t0, t, odes);
y

plot(t,y(5,:),t,y(6,:));
xtitle('Infections and Deaths','Time (days)','Number of Individuals');
legend('Number of Infected Cases', 'Number of Deaths');
```

Appendix C: Dengue Modeling Script

```
//Solving the system of ODE's representing the spread of the Dengue Virus using
//data from the outbreak in Lubbock, Texas.
//Parameters used:
//       la -- lambda in article -- human population growth rate
//       uh -- mu sub h in article -- human mortality rate
//       c -- c in article -- contact probability
//       texh -- tau sub exh in article -- inverse exposure time, host
//       a -- alpha in article -- dengue host mortality rate
//       tih -- tau sub ih in article -- inverse infective time, host
//       ua -- mu sub a in article -- adult mortality
//       texv -- tau sub exv in article -- inverse exposure time, vector
//       Hn -- initial number of humans

la = 2.244D-5;
uh = 1/28000;
c = 0.12;
texh = 1/10;
texv = 1/9;
a = 0.003;
ua = 1/20;
tih = 1/4;
Hn = 10000;

function ydot = odes(t, y)
    Ht = y(1) + y(2) + y(3) + y(4);
    Vt = y(5) + y(6) + y(7);
    ydot = zeros(y);
    //Hs
    ydot(1) = la * Ht - y(1)*(c*y(7)/Ht + uh);
    //He
    ydot(2) = y(1)*c*y(7)/Hn - y(2)*(texh + uh);
    //Hi
    ydot(3) = y(2)*texh - y(3)*(tih + a + uh);
    //Hr
    ydot(4) = y(3)*tih - uh*y(4);
    //Vs
    ydot(5) = ua*Vt - y(5)*(c*y(3)/Ht + ua);
    //Ve
    ydot(6) = y(5)*c*y(3)/Ht - y(6)*(texv + ua);
    //Vi
    ydot(7) = y(6)*texv - ua*y(7);
endfunction

t = 0:1:400;

y0 = [10000; 0; 0; 0; 100000; 0; 10000];
t0 = 0;

y = ode(y0, t0, t, odes);

//For Humans:
ht = y(1,:) + y(2,:) + y(3,:) + y(4,:);
//For Vectors:
//vt = y(5,:) + y(6,:) +y(7,:);

//Human population information plot.
plot(t, y(1,:), t, y(2,:), t, y(3,:), t, y(4,:), t, ht);
xtitle('Human Population', 'Time (days)', 'No. Of Humans');
legend('Susceptable', 'Exposed', 'Infected', 'Recovered', 'Total Population');
//Vector population information plot.
//plot(t, y(5,:), t, y(6,:), t, y(7,:), t, vt);
//xtitle('Vector Population', 'Time (days)', 'No. Of Vectors');
//legend('Susceptable', 'Exposed', 'Infected', 'Total Population');
```

Appendix D: Spatial SI Modeling and Plotting Scripts

```
! Attempting to solve the spatial SI model given in Lofti et al. 2013.
! Problem will be solved using two values for parameter be, the transmission
! Rate of the disease.
! This system consists of two PDE's (npde=2).
! Problem Parameters:
! ds     -- Diffusion rate for susceptable class.
! di     -- Diffusion rate for infected class.
! la     -- Recruitment rate of the population.
! mu     -- Natural death rate of the population.
! d      -- Death rate due to disease.
! r      -- Recovery rate of infected individuals
! as     -- Incidence rate of S.
! ai     -- Incidence rate of I.
! ar     -- Incidence rate of R.
! be     -- Infection Rate.
```

```
subroutine f(t, x, u, ux, uxx, fval, npde)
    implicit none
    !Purpose: Subroutine defines the right hand side vector of the NPDE
    ! ut = f(t, x, u, ux, uxx)
    !PARAMETERS:
    !Input:
    !Number of PDE's in the system
    integer :: npde
    !The current time coordinate
    double precision :: t
    !The current spacial coordinate
    double precision :: x
    !u(!:npde) is the approximation of the solution at point (t,x)
    double precision :: u(npde)
    !ux(!:npde) is the approximation of the spatial derivative of the
    !solition at the point (t,x)
    double precision :: ux(npde)
    !uxx(1:npde) is the approximation of the second spatial derivate
    !of the solution at the pnt. (t,x)
    double precision :: uxx(npde)
    !fval(1:npde) is the right hand side of the vector in the PDE.
    double precision  :: fval(npde)
!-----------------------------------------------------------------------
!----PROBLEM CONSTANTS
    double precision :: ds, di, la, mu, d, r, as, ai, ar, be
    common /SI/ ds, di, la, mu, d, r, as, ai, ar, be

!-----------------------------------------------------------------------
! Assign values for our PDE's
!-----------------------------------------------------------------------
    !S
    fval(1) = ds*uxx(1) + la - mu*u(1) &
        & - (be*u(1)*u(2))/(1 + as*u(1) + ai*u(2) + ar*u(1)*u(2))
    !I
    fval(2) = di*uxx(2) - (mu + d + r)*u(2) &
        & + (be*u(1)*u(2))/(1 + as*u(1) + ai*u(2) + ar*u(1)*u(2))
!-----------------------------------------------------------------------
end subroutine f

!Note: Not using dirivf subroutine to make Jacobians for this problem.
!Note: difbxa and difbxb also not used.

! PURPOSE:
!       THE SUBROUTINE IS USED TO DEFINE THE BOUNDARY CONDITIONS AT THE
!       LEFT SPATIAL END POINT X = XA.
!                           B(T, U, UX) = 0
!
!-----------------------------------------------------------------------
subroutine bndxa(t, u, ux, bval, npde)
    implicit none
! SUBROUTINE PARAMETERS:
! INPUT:
    integer :: npde
!                                   THE NUMBER OF PDES IN THE SYSTEM.
!
    double precision :: t
!                                   THE CURRENT TIME COORDINATE.
!
    double precision :: u(npde)
!                                   U(1:NPDE) IS THE APPROXIMATION OF THE
!                                   SOLUTION AT THE POINT (T,XA).
!
    double precision :: ux(npde)
!                                   UX(1:NPDE) IS THE APPROXIMATION OF THE
!                                   SPATIAL DERIVATIVE OF THE SOLUTION AT
!                                   THE POINT (T,XA).
!
! OUTPUT:
    double precision :: bval(npde)
!                                   BVAL(1:NPDE) IS THE BOUNDARY CONDITION
!                                   AT THE LEFT BOUNDARY POINT.
!-----------------------------------------------------------------------
!       PROBLEM CONSTANTS
!
    double precision :: ds, di, la, mu, d, r, as, ai, ar, be
    common /SI/ ds, di, la, mu, d, r, as, ai, ar, be
!-----------------------------------------------------------------------
    bval(1) = ux(1)
    bval(2) = ux(2)
end subroutine bndxa

! PURPOSE:
!       THE SUBROUTINE IS USED TO DEFINE THE BOUNDARY CONDITIONS AT THE
```

```
!        RIGHT SPATIAL END POINT X = XB.
!                                B(T, U, UX) = 0
!
!-----------------------------------------------------------------------
subroutine bndxb(t, u, ux, bval, npde)
    implicit none
! SUBROUTINE PARAMETERS:
! INPUT:
    integer :: npde
!                                THE NUMBER OF PDES IN THE SYSTEM.
!
    double precision :: t
!                                THE CURRENT TIME COORDINATE.
!
    double precision :: u(npde)
!                                U(1:NPDE) IS THE APPROXIMATION OF THE
!                                SOLUTION AT THE POINT (T,XA).
!
    double precision :: ux(npde)
!                                UX(1:NPDE) IS THE APPROXIMATION OF THE
!                                SPATIAL DERIVATIVE OF THE SOLUTION AT
!                                THE POINT (T,XA).
!
! OUTPUT:
    double precision :: bval(npde)
!                                BVAL(1:NPDE) IS THE BOUNDARY CONTIDITION
!                                AT THE LEFT BOUNDARY POINT.
!-----------------------------------------------------------------------
!        PROBLEM CONSTANTS
!
    double precision :: ds, di, la, mu, d, r, as, ai, ar, be
    common /SI/ ds, di, la, mu, d, r, as, ai, ar, be
!-----------------------------------------------------------------------
    bval(1) = ux(1)
    bval(2) = ux(2)
end subroutine bndxb

!-----------------------------------------------------------------------
! PURPOSE:
!      THIS FUNCTION PROVIDES THE EXACT SOLUTION OF THE PDE.
!-----------------------------------------------------------------------
subroutine truu(t, x, u, npde)
    implicit none
! INPUT:
    integer :: npde
!                                THE NUMBER OF PDES IN THE SYSTEM.
!
    double precision :: t
!                                THE CURRENT TIME COORDINATE.
!
    double precision :: x
!                                THE CURRENT SPATIAL COORDINATE.
!
! OUTPUT:
    double precision :: u(npde)
!                                U(1:NPDE) IS THE EXACT SOLUTION AT THE
!                                POINT (T,X).
!-----------------------------------------------------------------------
! true solution is unknown
    u(1) = 0.d0
    u(2) = 0.d0
end subroutine truu

!-----------------------------------------------------------------------
! PURPOSE:
!        THIS SUBROUTINE IS USED TO RETURN THE NPDE-VECTOR OF INITIAL
!        CONDITIONS OF THE UNKNOWN FUNCTION AT THE INITIAL TIME T = T0
!        AT THE SPATIAL COORDINATE X.
!-----------------------------------------------------------------------
subroutine uinit(x, u, npde)
    implicit none
! SUBROUTINE PARAMETERS:
! INPUT:
    double precision :: x
!                                THE SPATIAL COORDINATE.
!
    integer :: npde
!                                THE NUMBER OF PDES IN THE SYSTEM.
!
! OUTPUT:
    double precision :: u(npde)
!                                U(1:NPDE) IS VECTOR OF INITIAL VALUES OF
!                                THE UNKNOWN FUNCTION AT T = T0 AND THE
!                                GIVEN VALUE OF X.
```

```
!-----------------------------------------------------------------------
!       PROBLEM CONSTANTS
!
    double precision :: ds, di, la, mu, d, r, as, ai, ar, be
    common /SI/ ds, di, la, mu, d, r, as, ai, ar, be
!-----------------------------------------------------------------------
    if (x >= 0.0d0 .and. x < 0.5d0) then
        u(1) = 1.1*x
        u(2) = 0.5*x
    else if (x >= 0.5d0 .and. x <= 1.0d0) then
        u(1) = 1.1*(1 - x)
        u(2) = 0.5*(1 - x)
    end if
end subroutine uinit

!-----------------------------------------------------------------------
! purpose:
!       this subroutine writes a header describing the npde dimensional
!       parabolic partial differential equation
!                         ut = f(t, x, u, ux, uxx).
!-----------------------------------------------------------------------
subroutine header(nout)
    implicit none
! subroutine parameters:
! input:
    integer :: nout
!                                   nout is the output unit number.
!-----------------------------------------------------------------------
! constants:
    double precision, parameter :: t0 = 0.0d0
!
    double precision, parameter :: xa = 1.0d0
!
    double precision, parameter :: xb = 2.0d0
!-----------------------------------------------------------------------
!
    write(nout,95) 'The SI Model'
    write(nout,95) 'domain:'
    write(nout,96) '    t0 =', t0, ' < t,'
    write(nout,96) '    xa =', xa, ' <= x <= xb =', xb, ','

    95 format(a)
    96 format(a,e13.5,a,e13.5,a,e13.5,a,e13.5,a)
end subroutine header
```

```
!-----------------------------------------------------------------------
! Driver program for the Spatial SI model from Samsuzzoha et. al 2010.

program spatial_si_driver

    use bacoli95_mod, only: bacoli95_init, bacoli95, bacoli95_vals
    use bacoli95_mod, only: bacoli95_sol, bacoli95_sol_teardown

    implicit none
    integer, parameter      :: dp = kind(0d0)
    type(bacoli95_sol)      :: sol

    integer,  parameter     :: npde = 2, nint_max=1000
    real(dp), parameter     :: xa = 0, xb = 1
    real(dp), allocatable   :: uout(:,:)
    real(dp)                :: tout, tstart, tstop, atol(npde), rtol(npde)

    integer                 :: i, j, k, ier, ntout
    character(len=32)       :: fname, npde_str

    external f, bndxa, bndxb, uinit

!-----------------------------------------------------------------------
    double precision :: ds, di, la, mu, d, r, as, ai, ar, be
    common /SI/ ds, di, la, mu, d, r, as, ai, ar, be

    ds = 0.1
    di = 0.5
    la = 0.5
    !be = 0.2 for figure 2, be = 0.6 for figure 3
    be = 0.2
    mu = 0.1
    as = 0.1
    ai = 0.02
    ar = 0.03
    d = 0.1
    r = 0.5
```

```
    !-----------------------------------------------------------------------
    ! Write out value of npde to allow user to confirm that its value
    ! is appropriate for the problem to be solved.

    write(6,*) 'The number of PDEs is assumed to be ', npde
    write(6,*)

    ! Get some user input
    !print*, "Enter tstop, the end of the temporal domain."
    !read(*,*,err=600) tstop
    tstop = 100.0d0

    !print*, "At how many equally-spaced points along the time domain" &
    !     // " is output desired?"
    !read(*,*,err=600) ntout
    ntout = 100

    !print*, "Please choose an error tolerance"
    !read(*,*,err=600) atol(1)
    atol(1) = 1d-5
    rtol(1) = atol(1)

    !-----------------------------------------------------------------------
    ! Initialization: Allocate storage and set problem parameters.
!     call bacoli95_init(sol, npde, (/xa,xb/), atol=atol, rtol=rtol, dirichlet=1)
    call bacoli95_init(sol, npde, (/xa,xb/), atol=atol, rtol=rtol, &
            nint_max=nint_max)

    allocate(uout(npde,sol\%nint_max+1), stat=ier)
    if (ier /= 0 .or. sol\%idid == -1000) goto 700
    tstart = sol\%t0

    !-----------------------------------------------------------------------
    ! Open files for output.
    do k = 1, npde
        write(npde_str,*) k
        fname = 'Points' // adjustl(trim(npde_str))
        open(unit=10+k, file=fname)
    end do

    !-----------------------------------------------------------------------
    ! Integrate solution from t=0 to t=tout.
    print '(/"THE INPUT IS")'
    print 900, sol\%kcol, sol\%nint, sol\%npde, tstop
    print 901, sol\%atol(1), sol\%rtol(1), "LOI"

    do j = 2, ntout

        tout = tstart + (j-1)*(tstop-tstart)/(ntout-1)

        call bacoli95(sol, tout, f, bndxa, bndxb, uinit)
        if (sol\%idid <= 0) goto 800
        !print 902, sol\%nint

        if (j == 2) then
            do i = 1, sol\%nint+1
                call uinit(sol\%x(i), uout(1,i), npde)
            end do
            do k = 1, npde
                do i = 1, sol\%nint+1
                    write(10+k,*) sol\%x(i), tstart, uout(k,i)
                end do
            end do
        end if

        call bacoli95_vals(sol, sol\%x(1:sol\%nint+1), uout)

        do k = 1, npde
            do i = 1, sol\%nint+1
                write(10+k,*) sol\%x(i), sol\%t0, uout(k,i)
            end do
        end do
    end do

    print '("IDID        = ",i10)', sol\%idid
    print '("nsteps      = ",i10)', sol\%num_accepted_time_steps

    !-----------------------------------------------------------------------

    call bacoli95_sol_teardown(sol) ; stop
600 print '("Error: Improperly formatted input")' ; stop
700 print '("Error: Could not allocate storage")' ; stop
800 print '("Error: Was not able to integrate to tsop")' ; stop
```

```
      !-----------------------------------------------------------------------
      ! Formats!
 900  format(" kcol = ",i2," nint0 = ",i4," npde = ",i3," tout = ",es7.1)
 901  format(" atol = ",es7.1," rtol = ",es7.1,"  ",17x,a3)
 902  format(" Number of subintervals in the current mesh:",i8)

      end program
```

```
import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt
mpl.use('AGG')  # for systems not running a GUI
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

styling = {
    'cmap': cm.coolwarm,
    'linewidth': 0,
    'antialiased': True
}

# For the susceptable population:
x, t, u = np.loadtxt('Points1', unpack=True)
# For the infected population:
# x, t, u, = np.loadtxt('Points2', unpack=True)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_trisurf(x, t, u, **styling)

ax.set_xlabel('$x$')
ax.set_ylabel('$t$')
ax.set_zlabel('$u(t,x)$')

plt.savefig('trimesh.png')
```

Appendix E: Spatial Influenza Modeling and Plotting Scripts

```
! Attempting to solve the Reaction-Diffusion Influenze model given in Samsuzzoha
! et al. 2013.
! Solution for system with initial condition iii (as given in the paper).
! This system consists of two PDE's (npde=4).
! Problem Parameters:
! be     -- Transmission coefficient.
! mu     -- Natural mortality rate.
! r      -- Birth rate.
! ka     -- Carrying capacity.
! sig    -- Duration of latency.
! kb     -- Recovery rate of latents.
! alph   -- Flu induced mortality rate.
! gam    -- Recovery rate for clinically ill.
! d1, d2, d3, d4  -- Diffusion coefficients.

subroutine f(t, x, u, ux, uxx, fval, npde)
    implicit none
    !Purpose: Subroutine defines the right hand side vector of the NPDE
    ! ut = f(t, x, u, ux, uxx)
    !PARAMETERS:
    !Input:
    !Number of PDE's in the system
    integer :: npde
    !The current time coordinate
    double precision :: t
    !The current spacial coordinate
    double precision :: x
    !u(!:npde) is the approximation of the solution at point (t,x)
    double precision :: u(npde)
    !ux(!:npde) is the approximation of the spatial derivative of the
    !solition at the point (t,x)
    double precision :: ux(npde)
    !uxx(1:npde) is the approximation of the second spatial derivate
    !of the solution at the pnt. (t,x)
    double precision :: uxx(npde)
    !fval(1:npde) is the right hand side of the vector in the PDE.
    double precision  :: fval(npde)
!-----------------------------------------------------------------------
!----PROBLEM CONSTANTS
    double precision :: be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4
    double precision :: N
    common /SEIR/ be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4
!-----------------------------------------------------------------------
! Assign values for our PDE's
!-----------------------------------------------------------------------
```

```
      N = u(1) + u(2) + u(3) + u(4)
      !S
      fval(1) = -be*u(1)*(u(2) + u(3))/N - mu*u(1) + r*N*(1 - N/ka) + d1*uxx(1)
      !E
      fval(2) = be*u(1)*(u(2) + u(3))/N - (mu + sig + kb)*u(2) + d2*uxx(2)
      !I
      fval(3) = sig*u(2) - (mu + alph + gam)*u(3) + d3*uxx(3)
      !R
      fval(4) = kb*u(2) + gam*u(3) - mu*u(4) + d4*uxx(4)
!-----------------------------------------------------------------------
end subroutine f

!Note: Not using dirivf subroutine to make Jacobians for this problem.
!Note: difbxa and difbxb also not used.

! PURPOSE:
!       THE SUBROUTINE IS USED TO DEFINE THE BOUNDARY CONDITIONS AT THE
!       LEFT SPATIAL END POINT X = XA.
!                              B(T, U, UX) = 0
!
!-----------------------------------------------------------------------
subroutine bndxa(t, u, ux, bval, npde)
    implicit none
! SUBROUTINE PARAMETERS:
! INPUT:
    integer :: npde
!                                  THE NUMBER OF PDES IN THE SYSTEM.
!
    double precision :: t
!                                  THE CURRENT TIME COORDINATE.
!
    double precision :: u(npde)
!                                  U(1:NPDE) IS THE APPROXIMATION OF THE
!                                  SOLUTION AT THE POINT (T,XA).
!
    double precision :: ux(npde)
!                                  UX(1:NPDE) IS THE APPROXIMATION OF THE
!                                  SPATIAL DERIVATIVE OF THE SOLUTION AT
!                                  THE POINT (T,XA).
!
! OUTPUT:
    double precision :: bval(npde)
!                                  BVAL(1:NPDE) IS THE BOUNDARY CONTIDITION
!                                  AT THE LEFT BOUNDARY POINT.
!-----------------------------------------------------------------------
!       PROBLEM CONSTANTS
!
    double precision :: be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4
    common /SEIR/ be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4
!-----------------------------------------------------------------------
    bval(1) = ux(1)
    bval(2) = ux(2)
    bval(3) = ux(3)
    bval(4) = ux(4)
end subroutine bndxa

! PURPOSE:
!       THE SUBROUTINE IS USED TO DEFINE THE BOUNDARY CONDITIONS AT THE
!       RIGHT SPATIAL END POINT X = XB.
!                              B(T, U, UX) = 0
!
!-----------------------------------------------------------------------
subroutine bndxb(t, u, ux, bval, npde)
    implicit none
! SUBROUTINE PARAMETERS:
! INPUT:
    integer :: npde
!                                  THE NUMBER OF PDES IN THE SYSTEM.
!
    double precision :: t
!                                  THE CURRENT TIME COORDINATE.
!
    double precision :: u(npde)
!                                  U(1:NPDE) IS THE APPROXIMATION OF THE
!                                  SOLUTION AT THE POINT (T,XA).
!
    double precision :: ux(npde)
!                                  UX(1:NPDE) IS THE APPROXIMATION OF THE
!                                  SPATIAL DERIVATIVE OF THE SOLUTION AT
!                                  THE POINT (T,XA).
!
! OUTPUT:
    double precision :: bval(npde)
!                                  BVAL(1:NPDE) IS THE BOUNDARY CONTIDITION
```

```
!                                   AT THE LEFT BOUNDARY POINT.
!-----------------------------------------------------------------------
!       PROBLEM CONSTANTS
!
      double precision :: be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4
      common /SEIR/ be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4
!-----------------------------------------------------------------------
      bval(1) = ux(1)
      bval(2) = ux(2)
      bval(3) = ux(3)
      bval(4) = ux(4)
end subroutine bndxb

!-----------------------------------------------------------------------
! PURPOSE:
!      THIS FUNCTION PROVIDES THE EXACT SOLUTION OF THE PDE.
!-----------------------------------------------------------------------
subroutine truu(t, x, u, npde)
      implicit none
! INPUT:
      integer :: npde
!                                   THE NUMBER OF PDES IN THE SYSTEM.
!
      double precision :: t
!                                   THE CURRENT TIME COORDINATE.
!
      double precision :: x
!                                   THE CURRENT SPATIAL COORDINATE.
!
! OUTPUT:
      double precision :: u(npde)
!                                   U(1:NPDE) IS THE EXACT SOLUTION AT THE
!                                   POINT (T,X).
!-----------------------------------------------------------------------
! true solution is unknown
      u(1) = 0.d0
      u(2) = 0.d0
      u(3) = 0.d0
      u(4) = 0.d0
end subroutine truu

!-----------------------------------------------------------------------
! PURPOSE:
!       THIS SUBROUTINE IS USED TO RETURN THE NPDE-VECTOR OF INITIAL
!       CONDITIONS OF THE UNKNOWN FUNCTION AT THE INITIAL TIME T = T0
!       AT THE SPATIAL COORDINATE X.
!-----------------------------------------------------------------------
subroutine uinit(x, u, npde)
      implicit none
! SUBROUTINE PARAMETERS:
! INPUT:
      double precision :: x
!                                   THE SPATIAL COORDINATE.
!
      integer :: npde
!                                   THE NUMBER OF PDES IN THE SYSTEM.
!
! OUTPUT:
      double precision :: u(npde)
!                                   U(1:NPDE) IS VECTOR OF INITIAL VALUES OF
!                                   THE UNKNOWN FUNCTION AT T = T0 AND THE
!                                   GIVEN VALUE OF X.
!-----------------------------------------------------------------------
!       PROBLEM CONSTANTS
!
      double precision :: be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4
      common /SEIR/ be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4
!-----------------------------------------------------------------------
      u(1) = 0.96*exp(-10*(x**2))
      u(2) = 0.0d0
      u(3) = 0.04*exp(-100*(x**2))
      u(4) = 0.0d0

end subroutine uinit

!-----------------------------------------------------------------------
! purpose:
!       this subroutine writes a header describing the npde dimensional
!       parabolic partial differential equation
!                         ut = f(t, x, u, ux, uxx).
!-----------------------------------------------------------------------
subroutine header(nout)
      implicit none
! subroutine parameters:
```

```
! input:
    integer :: nout
!                                   nout is the output unit number.
!-----------------------------------------------------------------------
! constants:
    double precision, parameter :: t0 = 0.0d0
!
    double precision, parameter :: xa = -2.0d0
!
    double precision, parameter :: xb = 2.0d0
!-----------------------------------------------------------------------
!
    write(nout,95) 'The SEIR Model'
    write(nout,95) 'domain:'
    write(nout,96) '    t0 =', t0, ' < t,'
    write(nout,96) '    xa =', xa, ' <= x <= xb =', xb, ','

    95 format(a)
    96 format(a,e13.5,a,e13.5,a,e13.5,a,e13.5,a)
end subroutine header
```

```
! Driver for Samsuzzoha et, al 2010.
! To be run multiple times with tout = 5, 10, 15, 40.

program spatial_influenza_driver

    use bacoli95_mod, only: bacoli95_init, bacoli95, bacoli95_vals
    use bacoli95_mod, only: bacoli95_sol, bacoli95_sol_teardown
    use bacoli95_mod, only: bacoli95_sol_to_splines
    use bacoli95_mod, only: bacoli95_splines, bacoli95_splines_teardown

    implicit none
    integer, parameter      :: dp = kind(0d0)
    type(bacoli95_sol)      :: sol
    type(bacoli95_splines) :: splines

    integer, parameter      :: npde = 4, nderiv = 1
    real(dp), parameter     :: xa = -2.0, xb = 2.0

    integer                 :: nout
    real(dp), allocatable   :: xout(:), uout(:,:,:)
    real(dp)                :: tout, atol(npde), rtol(npde)
    logical                 :: serialize, exist

    integer                 :: i, j, ier

    external f, bndxa, bndxb, uinit

    double precision :: be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4
    common /SEIR/ be, mu, r, ka, sig, kb, alph, gam, d1, d2, d3, d4

    ! Problem parameters.
    be = 0.514d0
    gam = 0.20d0
    mu = 5.5d-5d0
    r = 0.0714d0
    ka = 1.0d0
    sig = 0.50d0
    kb = 0.1857d0
    alph = 0.0093d0
    d1 = 0.05d0
    d2 = 0.025d0
    d3 = 0.001d0
    d4 = 0.0d0

    !-----------------------------------------------------------------------
    ! Write out value of npde to allow user to confirm that its value
    ! is appropriate for the problem to be solved.

    write(6,*) 'The number of PDEs is assumed to be ', npde
    write(6,*)

    ! Set output time
    tout = 40.0d0

    ! Set tolerance
    atol = 1d-6
    rtol = atol
    atol(4) = 1d-1

    ! Save the solution as a spline for later use with a python plotting script.
    serialize = .true.
```

```
    !-----------------------------------------------------------------------
    ! Initialization: Allocate storage and set problem parameters.
    call bacoli95_init(sol, npde, (/-2.0d0, -1.6d0, -1.2d0, -0.8d0, -0.4d0,   &
            0.0d0, 0.4d0, 0.8d0, 1.2d0, 1.6d0, 2.0d0/), atol=atol, rtol=rtol)

    ! Set ouput at 8 uniformly spaced points across the spatial domain
    nout = 8
    allocate(xout(nout), uout(npde,nout,0:nderiv), stat=ier)
    if (ier /= 0 .or. sol\%idid == -1000) goto 700

    !-----------------------------------------------------------------------
    ! Integrate solution from t=0 to t=tout.
    print '(/"THE INPUT IS")'
    print 900, sol\%kcol, sol\%nint, sol\%npde, tout
    print 901, sol\%atol(1), sol\%rtol(1), "LOI"

    ! Compute the solution at tout
    call bacoli95(sol, tout, f, bndxa, bndxb, uinit)

    ! Output idid to check for a successful computation
    print '("idid=",i5)',sol\%idid
    if (sol\%idid > 0) print '("idid > 0 => Successful computation")'

    !-----------------------------------------------------------------------
    ! Output results.
    if (sol\%idid > 0) then

            ! Set output points and evaluate solution at these points
            xout = (/xa,(xa+i*(xb-xa)/(nout-1), i=1,nout-1)/)
            call bacoli95_vals(sol, xout, uout, nderiv=nderiv)

            print '(/"THE OUTPUT IS")'
            print '(a13,a27)', "XOUT", "UOUT"
            do i = 1, nout
                print*, xout(i), uout(:,i,0)
            end do

            do j = 1, nderiv
                print '(/"SPATIAL PARTIAL DERIVATIVE",i2)', j
                print '(a13,a27)', "XOUT", "UOUT"
                do i = 1, nout
                    print*, xout(i), uout(:,i,j)
                end do
            end do

        ! Write B-spline information to file.
        if (serialize) then

            call bacoli95_sol_to_splines(sol, splines, ier)
            if (ier /= 0) goto 700


            ! Make it so running the program multiple times will append new
            ! solutions to the same text file.
            inquire(file="Bsplines", exist=exist)
            if (exist) then
                open(20, file="Bsplines", status="old", position="append", &
                    & action="write")
            else
                open(20, file="Bsplines", status="new", action="write")
            end if

            ! line 1: number of PDEs in the system
            write(20,*) splines\%npde
            ! line 2: breakpoint/knot sequence
            write(20,*) splines\%knots
            ! line 3: coefficient values at time t0
            write(20,*) splines\%y
            ! line 4: degree of the B-spline interpolant
            write(20,*) splines\%p
            close(20)

            call bacoli95_splines_teardown(splines)
        end if
    end if

    !-----------------------------------------------------------------------
    ! The end.
    call bacoli95_sol_teardown(sol) ; stop
600 print '("Error: Improperly formatted input")' ; stop
700 print '("Error: Could not allocate storage")' ; stop

    !-----------------------------------------------------------------------
    ! Formats
```

```
900 format(" kcol = ",i2," nint0 = ",i4," npde = ",i3," tout = ",es7.1)
901 format(" atol = ",es7.1," rtol = ",es7.1,",",17x,a3)
902 format("Number of subintervals in the current mesh:",i8)
end program
```

```
# Script which plots the solutions for the equations given in Sumsuzzoha et al.
# 2010.

import matplotlib as mpl
mpl.use('AGG')
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import splev

with open('Bsplines') as f: lines = f.readlines()

plt.axis([-2,2, 0.0, 0.9])

xbs, y, p, npde = [], [], [], []

# Read the required information for each of the 4 solutions.
for i in range(0, 5):
    npde.append(int(lines[0 + i*4]))
    xbs.append(np.fromstring(lines[1 + i*4], sep=' '))
    y.append(np.fromstring(lines[2 + i*4], sep=' ').reshape((npde[i], -1)))
    p.append(int(lines[3 + i*4]))

x = np.linspace(-2, 2, 1000)

# For each iteration of loop, plot the values for one of the times.
for i in range(0, 5):
    # y is a list of lists, make second index 0 for susceptable curve, 1 for
    # exposed curve, 2 for infected curve and 3 for recovered curve.
    u = splev(x, (xbs[i], y[i][3], p[i]))
    if i == 0:
        t = 0
    elif i == 1:
        t = 5
    elif i == 2:
        t = 10
    elif i == 3:
        t = 15
    else:
        t = 40

    plt.plot(x, u, label='$t = {%d}$'%(t))

plt.xlabel('$x$')
plt.ylabel('$Recovered$')
plt.legend()
plt.grid()
plt.savefig('curve.png')
```

```
> restart;
> be := .514;

> gam := .20;

> mu := 5.5*10^(-5);

> r := 0.714e-1;

> ka := 1.0;

> sig := .50;

> kb := .1857;

> alph := 0.93e-2;

> d1 := 0.5e-1;

> d2 := 0.25e-1;

> d3 := 0.1e-2;

> d4 := 0.1e-37;

> PDE1 := diff(U1(x, t), t) = -be*U1(x, t)*(U2(x, t)+U3(x, t))/N-mu*U1(x, t)
    +r*N*(1-N/ka)+d1*(diff(U1(x, t), x, x));

> PDE2 := diff(U2(x, t), t) = be*U1(x, t)*(U2(x, t)+U3(x, t))/N
```

```
    -(mu+sig+kb)*U2(x, t)+d2*(diff(U2(x, t), x, x));

> PDE3 := diff(U3(x, t), t) = sig*U2(x, t)-(mu+alph+gam)*U3(x, t)
    +d3*(diff(U3(x, t), x, x));

> PDE4 := diff(U4(x, t), t) = kb*U2(x, t)+gam*U3(x, t)-mu*U4(x, t)
    +d4*(diff(U4(x, t), x, x));

> IBC := {U1(x, 0) = .96*exp(-10*x^2), U2(x, 0) = 0.,
    U3(x, 0) = 0.4e-1*exp(-100*x^2), U4(x, 0) = 0.,
    (D[1](U1))(-2, t) = 0, (D[1](U1))(2, t) = 0,
    (D[1](U2))(-2, t) = 0, (D[1](U2))(2, t) = 0,
    (D[1](U3))(-2, t) = 0, (D[1](U3))(2, t) = 0,
    (D[1](U4))(-2, t) = 0, (D[1](U4))(2, t) = 0};

> PDE := {PDE1, PDE2, PDE3, PDE4};

> pds := pdsolve(PDE, IBC, numeric, abstol = 10^(-5), timestep = 1/1000,
    spacestep = 1/1000);

> p1 := pds:-plot(U1(x, t), t = 5); p2 := pds:-plot(U1(x, t), t = 10);
    p3 := pds:-plot(U1(x, t), t = 15); p4 := pds:-plot(U1(x, t), t = 40);
    plots[display]({p1, p2, p3, p4},
    title = `Susceptable Population at t=5,10,15,40`);

> p1 := pds:-plot(U2, t = 5); p2 := pds:-plot(U2, t = 10);
    p3 := pds:-plot(U2, t = 15);
    p4 := pds:-plot(U2, t = 40); plots[display]({p1, p2, p3, p4},
    title = `Exposed Population at t=5,10,15,40`);

> p1 := pds:-plot(U3, t = 5); p2 := pds:-plot(U3, t = 10);
    p3 := pds:-plot(U3, t = 15);
    p4 := pds:-plot(U3, t = 40); plots[display]({p1, p2, p3, p4},
    title = `Infected Population at t=5,10,15,40`);

> p1 := pds:-plot(U4, t = 5); p2 := pds:-plot(U4, t = 10);
    p3 := pds:-plot(U4, t = 15);
    p4 := pds:-plot(U4, t = 40); plots[display]({p1, p2, p3, p4},
    title = `Recovered Population at t=5,10,15,40`);
```

Appendix F: Cholera Modeling and Plotting Scripts

```
! Attempting to solve the spatial SIBR model given in Capone et al. 2014 to
! model the spread of cholera in Haiti from 2010 - 2012.
! Two simulations are performed:
! Simulation 1: Model (1) with initial conditions (48) and homogenous Neumann
!               boundary conditions applied to case (51).
! Simulation 2: Model (1) with initial conditions S(x,0) = 3500, (48)_2, (48)_3
!               and R(x,0) = 100.
!
! This system consists of four PDE's (npde=4).
! Problem Parameters:
! n0    - Total population size at time t=0.
! gam_i - Diffusion coefficients (i = 1,2,3,4).
! mu    - Birth/Death rate.
! sig   - Recovery rate.
! mu_b  - Loss rate of bacteria.
! pi_b  - Growth rate of bacteria.
! e     - Contribution of each infected to bacteria population.
! be    - Contact rate with contaminated water.
! k_b   - Concentration rate of cholera bacteria which leads to a 50\% chance
!         of catching cholera.

subroutine f(t, x, u, ux, uxx, fval, npde)
    implicit none
    !Purpose: Subroutine defines the right hand side vector of the NPDE
    ! ut = f(t, x, u, ux, uxx)
    !PARAMETERS:
    !Input:
    !Number of PDE's in the system
    integer :: npde
    !The current time coordinate
    double precision :: t
    !The current spacial coordinate
    double precision :: x
    !u(!:npde) is the approximation of the solution at point (t,x)
    double precision :: u(npde)
    !ux(!:npde) is the approximation of the spatial derivative of the
    !solition at the point (t,x)
    double precision :: ux(npde)
    !uxx(1:npde) is the approximation of the second spatial derivate
    !of the solution at the pnt. (t,x)
    double precision :: uxx(npde)
```

```
    !fval(1:npde) is the right hand side of the vector in the PDE.
    double precision  :: fval(npde)
!-----------------------------------------------------------------------
!----PROBLEM CONSTANTS
    double precision :: n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, &
            e, be, k_b
    common /SIBR/ n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, e, &
            be, k_b

!-----------------------------------------------------------------------
! Assign values for our PDE's
!-----------------------------------------------------------------------
    !S
    fval(1) = mu*(n0 - u(1)) + gam_1*uxx(1) - be*(u(3)/(k_b + u(3)))*u(1)
    !I
    fval(2) = be*(u(3)/(k_b + u(3)))*u(1) - (sig + mu)*u(2) + gam_2*uxx(2)
    !B
    fval(3) = e*u(2) - (mu_b - pi_b)*u(3) + gam_3*uxx(3)
    !R
    fval(4) = sig*u(2) - mu*u(4) + gam_4*uxx(4)
!-----------------------------------------------------------------------
end subroutine f

!Note: Not using dirivf subroutine to make Jacobians for this problem.
!Note: difbxa and difbxb also not used.

! PURPOSE:
!       THE SUBROUTINE IS USED TO DEFINE THE BOUNDARY CONDITIONS AT THE
!       LEFT SPATIAL END POINT X = XA.
!                               B(T, U, UX) = 0
!
!-----------------------------------------------------------------------
subroutine bndxa(t, u, ux, bval, npde)
! SUBROUTINE PARAMETERS:
! INPUT:
    integer :: npde
!                               THE NUMBER OF PDES IN THE SYSTEM.
!
    double precision :: t
!                               THE CURRENT TIME COORDINATE.
!
    double precision :: u(npde)
!                               U(1:NPDE) IS THE APPROXIMATION OF THE
!                               SOLUTION AT THE POINT (T,XA).
!
    double precision :: ux(npde)
!                               UX(1:NPDE) IS THE APPROXIMATION OF THE
!                               SPATIAL DERIVATIVE OF THE SOLUTION AT
!                               THE POINT (T,XA).
!
! OUTPUT:
    double precision :: bval(npde)
!                               BVAL(1:NPDE) IS THE BOUNDARY CONTIDITION
!                               AT THE LEFT BOUNDARY POINT.
!-----------------------------------------------------------------------
!       PROBLEM CONSTANTS
!
    double precision :: n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, &
            e, be, k_b
    common /SIBR/ n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, e, &
            be, k_b

!-----------------------------------------------------------------------
    bval(1) = ux(1)
    bval(2) = ux(2)
    bval(3) = ux(3)
    bval(4) = ux(4)
end subroutine bndxa

! PURPOSE:
!       THE SUBROUTINE IS USED TO DEFINE THE BOUNDARY CONDITIONS AT THE
!       RIGHT SPATIAL END POINT X = XB.
!                               B(T, U, UX) = 0
!
!-----------------------------------------------------------------------
subroutine bndxb(t, u, ux, bval, npde)
! SUBROUTINE PARAMETERS:
! INPUT:
    integer :: npde
!                               THE NUMBER OF PDES IN THE SYSTEM.
!
    double precision :: t
!                               THE CURRENT TIME COORDINATE.
!
```

```
    double precision :: u(npde)
!                                   U(1:NPDE) IS THE APPROXIMATION OF THE
!                                   SOLUTION AT THE POINT (T,XA).
!
    double precision :: ux(npde)
!                                   UX(1:NPDE) IS THE APPROXIMATION OF THE
!                                   SPATIAL DERIVATIVE OF THE SOLUTION AT
!                                   THE POINT (T,XA).
!
! OUTPUT:
    double precision :: bval(npde)
!                                   BVAL(1:NPDE) IS THE BOUNDARY CONTIDITION
!                                   AT THE LEFT BOUNDARY POINT.
!-----------------------------------------------------------------------
!       PROBLEM CONSTANTS
!
    double precision :: n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, &
          e, be, k_b
    common /SIBR/ n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, e, &
          be, k_b
!-----------------------------------------------------------------------
    bval(1) = ux(1)
    bval(2) = ux(2)
    bval(3) = ux(3)
    bval(4) = ux(4)
end subroutine bndxb

!-----------------------------------------------------------------------
! PURPOSE:
!     THIS FUNCTION PROVIDES THE EXACT SOLUTION OF THE PDE.
!-----------------------------------------------------------------------
subroutine truu(t, x, u, npde)
! INPUT:
    integer :: npde
!                                   THE NUMBER OF PDES IN THE SYSTEM.
!
    double precision :: t
!                                   THE CURRENT TIME COORDINATE.
!
    double precision :: x
!                                   THE CURRENT SPATIAL COORDINATE.
!
! OUTPUT:
    double precision :: u(npde)
!                                   U(1:NPDE) IS THE EXACT SOLUTION AT THE
!                                   POINT (T,X).
!-----------------------------------------------------------------------
! true solution is unknown
    u(1) = 0.0d0
    u(2) = 0.0d0
    u(3) = 0.0d0
    u(4) = 0.0d0
end subroutine truu

!-----------------------------------------------------------------------
! PURPOSE:
!       THIS SUBROUTINE IS USED TO RETURN THE NPDE-VECTOR OF INITIAL
!       CONDITIONS OF THE UNKNOWN FUNCTION AT THE INITIAL TIME T = T0
!       AT THE SPATIAL COORDINATE X.
!-----------------------------------------------------------------------
subroutine uinit(x, u, npde)
! SUBROUTINE PARAMETERS:
! INPUT:
    double precision :: x
!                                   THE SPATIAL COORDINATE.
!
    integer :: npde
!                                   THE NUMBER OF PDES IN THE SYSTEM.
!
! OUTPUT:
    double precision :: u(npde)
!                                   U(1:NPDE) IS VECTOR OF INITIAL VALUES OF
!                                   THE UNKNOWN FUNCTION AT T = T0 AND THE
!                                   GIVEN VALUE OF X.
!-----------------------------------------------------------------------
!       PROBLEM CONSTANTS
!
    double precision :: n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, &
          e, be, k_b
    common /SIBR/ n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, e, &
          be, k_b
    double precision :: pi
    ! Used in sumulation 2:
    pi = acos(-1.0d0)
```

```
!-----------------------------------------------------------------------
    ! For simulation 1:
    ! u(1) = 2989.29d0
    ! u(2) = 9.17889d0
    ! u(3) = 278.148d0
    ! u(4) = 701.53d0

    ! For simulation 2:
    u(1) = 3500.0d0

    if (x >= 0.0d0 .and. x <=0.5d0) then
        u(2) = 100*pi*cos(pi*x)
        u(3) = 10*pi*cos(pi*x)
    else
        u(2) = 0
        u(3) = 0
    end if

    u(4) = 100


end subroutine uinit

!-----------------------------------------------------------------------
! purpose:
!       this subroutine writes a header describing the npde dimensional
!       parabolic partial differential equation
!                         ut = f(t, x, u, ux, uxx).
!-----------------------------------------------------------------------
subroutine header(nout)
! subroutine parameters:
! input:
    integer :: nout
!                                 nout is the output unit number.
!-----------------------------------------------------------------------
! constants:
    double precision, parameter :: t0 = 0.0d0
!
    double precision, parameter :: xa = 0.0d0
!
    double precision, parameter :: xb = 1.0d0
!-----------------------------------------------------------------------
!
    write(nout,95) 'The SIBR Model'
    write(nout,95) 'domain:'
    write(nout,96) '    t0 =', t0, ' < t,'
    write(nout,96) '    xa =', xa, ' <= x <= xb =', xb, ','

    95 format(a)
    96 format(a,e13.5,a,e13.5,a,e13.5,a,e13.5,a)
end subroutine header
```

```
! Driver for modeling spatial SIBR model from Capone et. al 2014 to model
! the spread of cholera in Haiti.

program cholera_haiti_driver

    use bacoli95_mod, only: bacoli95_init, bacoli95, bacoli95_vals
    use bacoli95_mod, only: bacoli95_sol, bacoli95_sol_teardown

    implicit none
    integer, parameter      :: dp = kind(0d0)
    type(bacoli95_sol)      :: sol

    integer,  parameter     :: npde = 4, nint_max=2000
    real(dp), parameter     :: xa = 0, xb = 1
    real(dp), allocatable   :: uout(:,:)
    real(dp)                :: tout, tstart, tstop, atol(npde), rtol(npde)

    integer                 :: i, j, k, ier, ntout
    character(len=32)       :: fname, npde_str

    external f, bndxa, bndxb, uinit

!-----------------------------------------------------------------------
    double precision :: n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, &
            e, be, k_b
    common /SIBR/ n0, gam_1, gam_2, gam_3, gam_4, mu, sig, mu_b, pi_b, e, &
            be, k_b

    ! Assign values to problem constants.
    n0 = 3700
```

```
    gam_1 = 0.8

    ! For sumulation 1:
    gam_2 = 0.003
    ! For simulation 2:
    ! gam_2 = 0.1

    ! For simulation 1:
    gam_3 = 0.002
    ! For simulation 2:
    ! gam_3 = 0.01

    gam_4 = 0.5

    mu = 0.014
    sig = 1.0678
    k_b = 10**5
    mu_b = 1.06
    pi_b = 0.73
    e = 10
    ! For simulation 1:
    be = 1.2
    ! For simulation 2:
    ! be = 1.0

    !-----------------------------------------------------------------------
    ! Write out value of npde to allow user to confirm that its value
    ! is appropriate for the problem to be solved.

    write(6,*) 'The number of PDEs is assumed to be ', npde
    write(6,*)

    ! Get some user input
    !print*, "Enter tstop, the end of the temporal domain."
    !read(*,*,err=600) tstop
    tstop = 2000.0d0

    !print*, "At how many equally-spaced points along the time domain" &
    !     // " is output desired?"
    !read(*,*,err=600) ntout
    ntout = 2000

    !print*, "Please choose an error tolerance"
    !read(*,*,err=600) atol(1)
    ! For simuluation 1, rtol = atol = 1d-7
    ! For simulation 2, rtol = atol = 1d-4
    atol(1) = 1d-4
    rtol(1) = atol(1)

    !-----------------------------------------------------------------------
    ! Initialization: Allocate storage and set problem parameters.
!     call bacoli95_init(sol, npde, (/xa,xb/), atol=atol, rtol=rtol, dirichlet=1)
    call bacoli95_init(sol, npde, (/xa,xb/), atol=atol, rtol=rtol, &
            nint_max=nint_max)

    allocate(uout(npde,sol\%nint_max+1), stat=ier)
    if (ier /= 0 .or. sol\%idid == -1000) goto 700
    tstart = sol\%t0

    !-----------------------------------------------------------------------
    ! Open files for output.
    do k = 1, npde
        write(npde_str,*) k
        fname = 'Points' // adjustl(trim(npde_str))
        open(unit=10+k, file=fname)
    end do

    !-----------------------------------------------------------------------
    ! Integrate solution from t=0 to t=tout.
    print '(/"THE INPUT IS")'
    print 900, sol\%kcol, sol\%nint, sol\%npde, tstop
    print 901, sol\%atol(1), sol\%rtol(1), "LOI"

    do j = 2, ntout

        tout = tstart + (j-1)*(tstop-tstart)/(ntout-1)

        call bacoli95(sol, tout, f, bndxa, bndxb, uinit)
        if (sol\%idid <= 0) goto 800
        !print 902, sol\%nint

        if (j == 2) then
            do i = 1, sol\%nint+1
                call uinit(sol\%x(i), uout(1,i), npde)
```

```
            end do
            do k = 1, npde
                do i = 1, sol\%nint+1
                    write(10+k,*) sol\%x(i), tstart, uout(k,i)
                end do
            end do
        end if

        call bacoli95_vals(sol, sol\%x(1:sol\%nint+1), uout)

        do k = 1, npde
            do i = 1, sol\%nint+1
                write(10+k,*) sol\%x(i), sol\%t0, uout(k,i)
            end do
        end do
    end do

    print '("IDID        = ",i10)', sol\%idid
    print '("nsteps      = ",i10)', sol\%num_accepted_time_steps

    !-------------------------------------------------------------------------

    call bacoli95_sol_teardown(sol) ; stop
600 print '("Error: Improperly formatted input")' ; stop
700 print '("Error: Could not allocate storage")' ; stop
800 print '("Error: Was not able to integrate to tsop")' ; stop

    !-------------------------------------------------------------------------
    ! Formats!
900 format("kcol = ",i2,", nint0 = ",i4,", npde = ",i3,", tout = ",es7.1)
901 format("atol = ",es7.1,", rtol = ",es7.1,",",17x,a3)
902 format("Number of subintervals in the current mesh:",i8)

end program
```

```
# Plot for cholera epidemic models from Capone et. al 2014.

import matplotlib as mpl
from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

styling = {
    'cmap': cm.coolwarm,
    'linewidth': 0,
    'antialiased': True
}

# For the susceptable population:
# To get graphs for the other compartments:
# --'Points1' for susceptable.
# --'Points2' for infected.
# --'Points3' for bacteria.
# --'Points4' for recovered.
x, t, u = np.loadtxt('Points1', unpack=True)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_trisurf(x, t, u, **styling)

ax.set_xlabel('$x$')
ax.set_ylabel('$t$')
ax.set_zlabel('$Recovered$')

plt.savefig('trimesh.png')
```

```
> restart;
> n0 := 3700;

> gam_1 := .8;

> gam_2 := 0.3e-2;

> gam_3 := 0.2e-2;

> gam_4 := .5;

> mu := 0.14e-1;

> sig := 1.0678;
```

```
> k_b := 10^5;

> mu_b := 1.06;

> pi_b := .73;

> e := 10;

> be := 1.2;

> PDE1 := diff(U1(x, t), t) = mu*(n0-U1(x, t))
+gam_1*(diff(U1(x, t), x, x))-be*U3(x, t)*U1(x, t)/(k_b+U3(x, t));

> PDE2 := diff(U2(x, t), t) = be*U3(x, t)*U1(x, t)/(k_b+U3(x, t))
-(sig+mu)*U2(x, t)+gam_2*(diff(U2(x, t), x, x));

> PDE3 := diff(U3(x, t), t) = e*U2(x, t)-(mu_b-pi_b)*U3(x, t)
+gam_3*(diff(U3(x, t), x, x));

> PDE4 := diff(U4(x, t), t) = sig*U2(x, t)-mu*U4(x, t)
+gam_4*(diff(U4(x, t), x, x));

> PDE := {PDE1, PDE2, PDE3, PDE4};

> IBC := {U1(x, 0) = 2989.29, U2(x, 0) = 9.17889, U3(x, 0) = 278.148,
U4(x, 0) = 701.53,
(D[1](U1))(0, t) = 0, (D[1](U1))(1, t) = 0, (D[1](U2))(0, t) = 0,
(D[1](U2))(1, t) = 0, (D[1](U3))(0, t) = 0,
(D[1](U3))(1, t) = 0, (D[1](U4))(0, t) = 0,
(D[1](U4))(1, t) = 0};

> pds := pdsolve(PDE, IBC, numeric, abstol=10^(-5), spacestep=1/1000,
    timestep=1/1000);

> plt1 := pds:-plot3d(U1(x, t), x = 0 .. 1, t = 0 .. 2000,
   axes = boxed, orientation = [-120, 40],
   color = [0, 0, U1],
   title = "Simulation 1 Susceptable Population");

> plt2 := pds:-plot3d(U2(x, t), x = 0 .. 1, t = 0 .. 2000,
   axes = boxed, orientation = [-120, 40],
   color = [0, 0, U2],
   title = "Simulation 1 Infected Population");

> plt3 := pds:-plot3d(U3(x, t), x = 0 .. 1, t = 0 .. 2000,
   axes = boxed, orientation = [-120, 40],
   color = [0, 0, U3],
   title = "Simulation 1 Bacteria Population");

> plt4 := pds:-plot3d(U4(x, t), x = 0 .. 1, t = 0 .. 2000,
    axes = boxed, orientation = [-120, 40],
    color = [0, 0, U4],
    title = "Simulation 1 Recovered Population");
```

```
> restart;
> n0 := 3700;

> gam_1 := .8;

> gam_2 := .1;

> gam_3 := 0.1e-1;

> gam_4 := .5;

> mu := 0.14e-1;

> sig := 1.0678;

> k_b := 10^5;

> mu_b := 1.06;

> pi_b := .73;

> e := 10;

> be := 1.0;

> PDE1 := diff(U1(x, t), t) = mu*(n0-U1(x, t))+gam_1*(diff(U1(x, t), x, x))
                  -be*U3(x, t)*U1(x, t)/(k_b+U3(x, t));

> PDE2 := diff(U2(x, t), t) = be*U3(x, t)*U1(x, t)/(k_b+U3(x, t))
```

```
                    -(sig+mu)*U2(x, t)+gam_2*(diff(U2(x, t), x, x));

> PDE3 := diff(U3(x, t), t) = e*U2(x, t)-(mu_b-pi_b)*U3(x, t)
                   +gam_3*(diff(U3(x, t), x, x));

> PDE4 := diff(U4(x, t), t) = sig*U2(x, t)-mu*U4(x, t)
                   +gam_4*(diff(U4(x, t), x, x));

> PDE := {PDE1, PDE2, PDE3, PDE4};

> UINIT1 := U1(x, 0) = 3500;

> UINIT2 := U2(x, 0) = piecewise(0 <= x and x <= .5, 100*Pi*cos(Pi*x),
.5 < x and x <= 1, 0);

> UINIT3 := U3(x, 0) = piecewise(0 <= x and x <= .5, 10*Pi*cos(Pi*x),
.5 < x and x <= 1, 0);

> UINIT4 := U4(x, 0) = 100;

> IBC := {UINIT1, UINIT2, UINIT3, UINIT4, (D[1](U1))(0, t) = 0,
                 (D[1](U1))(1, t) = 0, (D[1](U2))(0, t) = 0,
                 (D[1](U2))(1, t) = 0, (D[1](U3))(0, t) = 0, (D[1](U3))(1, t) = 0,
                 (D[1](U4))(0, t) = 0, (D[1](U4))(1, t) = 0};

> pds := pdsolve(PDE, IBC, numeric, timestep = 1/20, spacestep = 1/20);

> plt1 := pds:-plot3d(U1(x, t), x = 0 .. 1, t = 0 .. 2000,
axes = boxed, orientation = [-120, 40], color = [0, 0, U1],
title = "Simulation 2 Susceptable Population");

> plt2 := pds:-plot3d(U2(x, t), x = 0 .. 1, t = 0 .. 2000,
axes = boxed, orientation = [-120, 40], color = [0, 0, U2],
title = "Simulation 2 Infected Population");

> plt3 := pds:-plot3d(U3(x, t), x = 0 .. 1, t = 0 .. 2000,
 axes = boxed, orientation = [-120, 40], color = [0, 0, U3],
title = "Simulation 2 Bacteria Population");

> plt4 := pds:-plot3d(U4(x, t), x = 0 .. 1, t = 0 .. 2000,
axes = boxed, orientation = [-120, 40], color = [0, 0, U4],
title = "Simulation 2 Recovered Population");
```

References

[1] L. Allen et al. *Mathematical Epidemiology*. Vol. 1945. Springer, 2008.

[2] C. Althaus. *Estimating the Reproduction Number of Ebola Virus (EBOV) During the 2014 Outbreak in West Africa*. PLoS Currents 6 (2014).

[3] M. Andraud et al. *Dynamic Epidemiological Models for Dengue Transmission: A Systematic Review of Structural Approaches*. PloS ONE 7.11 (2012), pp. 1–14.

[4] S. Anita and V. Capasso. *Reaction-Diffusion Systems in Epidemiology*. ArXiv e-prints (2017).

[5] J. Arino and P. Van Den Driessche. *Time Delays in Epidemic Models. Delay Differential Equations and Applications*. Springer, 2006, pp. 539–578.

[6] K. Atkinson, W. Han, and D.E. Stewart. *Numerical Solution of Ordinary Differential Equations*. Vol. 108. John Wiley & Sons, 2011.

[7] E. Bertuzzo et al. *On Spatially Explicit Models of Cholera Epidemics*. Journal of the Royal Society Interface (2009), pp. 321–333.

[8] F. Brauer. *Mathematical Epidemiology: Past, Present, and Future.* Infectious Disease Modelling 2.2 (2017), pp. 113 –127.

[9] F. Capone, V. De Cataldis, and R. De Luca. *Influence of Diffusion on the Stability of Equilibria in a Reaction–Diffusion System Modeling Cholera Dynamic.* Journal of Mathematical Biology 71.5 (2015), pp. 1107–1131.

[10] F. Chalub and M. Souza. *Discrete and Continuous SIS Epidemic Models: A Unifying Approach.* Ecological Complexity 18 (2014), pp. 83–95.

[11] J. C. Díaz, G. Fairweather, and P. Keast. *Algorithm 603: COLROW and ARCECO: FORTRAN Packages for Solving Certain Almost Block Diagonal Linear Systems by Modified Alternate Row and Column Elimination.* ACM Transactions on Mathematical Software (TOMS) 9 (1983), pp. 376–380.

[12] P. Van den Driessche and J. Watmough. *Reproduction Numbers and Sub-Threshold Endemic Equilibria for Compartmental Models of Disease Transmission.* Mathematical Biosciences 180.1 (2002), pp. 29–48.

[13] Hairer E. and Lubich C. *Numerical Solution of Ordinary Differential Equations.* The Princeton Companion to Applied Mathematics (2012).

[14] L. Elelstein-Keshet. *Applications of Continuous Models to Population Dynamics. Mathematical Models in Biology.* Society for Industrial and Applied Mathematics, 2004. Chap. 6.

[15] Scilab Enterprises. *Scilab: Free and Open Source Software for Numerical Computation.* Scilab Enterprises. Orsay, France, 2012.

[16] R.A. Erickson et al. *A Dengue Model with a Dynamic Aedes Albopictus Vector Population.* Ecological Modelling 221.24 (2010), pp. 1–14.

[17] G. Everstine. *Numerical Solution of Partial Differential Equations.* 2010.

[18] K.R. Green and R.J. Spiteri. *Extended BACOLI: Solving One-Dimensional Multiscale Parabolic PDE Systems with Error Control.* Submitted to ACM Transactions on Mathematical Software (TOMS) (2017).

[19] H. Hethcote. *The Mathematics of Infectious Diseases.* SIAM review 42.4 (2000), pp. 599–653.

[20] A. Hindmarsh. *LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers.* ACM Signum Newsletter 15.4 (1980), pp. 10–11.

[21] A. Hindmarsh. *ODEPACK - A Systematized Collection of ODE Solvers.* IMACS Transactions on Scientific Computation 1 (1983), pp. 55–64.

[22] A. Huppert and G. Katriel. *Mathematical Modelling and Prediction In Infectious Disease Epidemiology*. Clinical Microbiology and Infection 19.11 (2013), pp. 999–1005.

[23] P. Keast. *LAMPACK - Software for the Factorization of Almost Block Diagonal Systems*. Unpublished Software.

[24] M. Keeling and P. Rohani. *Modeling Infectious Diseases In Humans and Animals*. Princeton University Press, 2008.

[25] B.L. Keyfitz and N. Keyfitz. *The McKendrick Partial Differential Equation and Its Uses In Epidemiology and Population Study*. Mathematical and Computer Modelling 26.6 (1997), pp. 1–9.

[26] C.M.L. Kon, J.S. Labadin, and J. Tiga. *Generic Reaction-Diffusion Model For Transmission Of Mosquito-Borne Diseases: Results Of Simulation With Actual Cases. European Conference on Modelling and Simulation*. 2016, pp. 1–4.

[27] E.M. Lotfi et al. *Partial Differential Equations of an Epidemic Model with Spatial Diffusion*. International Journal of Partial Differential Equations 2014 (2014).

[28] P. Munz et al. *When Zombies Attack!: Mathematical Modelling of an Outbreak of Zombie Infection*. Infectious Disease Modelling Research Progress 4 (2009), pp. 133–150.

[29] J.D. Murray. *Mathematical Biology II: Spatial Models and Biomedical Applications*. Interdisciplinary Applied Mathematics (2013).

[30] D.P. O'Leary. *Scientific Computing with Case Studies*. Society for Industrial and Applied Mathematics, 2009.

[31] P. Peterson. *F2PY: a Tool For Connecting Fortran and Python Programs*. International Journal of Computational Science and Engineering 4 (2009).

[32] L. Petzold. *A Description of DASSL: A Differential/Algebraic System Solver*. Scientific Computing 1 (1982), pp. 65–68.

[33] J. Pew, Z. Li, and P. Muir. *Algorithm 962: BACOLI: B-spline Adaptive Collocation Software for PDEs with Interpolation-Based Spatial Error Control*. ACM Transactions on Mathematical Software (TOMS) 42.3 (2016), pp. 1–17.

[34] J.R. Rice. *Numerical Methods in Software and Analysis*. Elsevier, 2014.

[35] E. Rusu. *Network Models in Epidemiology: Considering Discrete and Continuous Dynamics*. arXiv preprint arXiv:1511.01062 (2015).

[36] M. Samsuzzoha, M. Singh, and D. Lucy. *Numerical Study of an Influenza Epidemic Model with Diffusion*. Applied Mathematics and Computation 217.7 (2010), pp. 3461–3479.

[37] D.L. Smith et al. *Ross, Macdonald, and a Theory for the Dynamics and Control of Mosquito-Transmitted Pathogens.* PLoS Pathogens 8.4 (2012), pp. 1–13.

[38] R. Wang, P. Keast, and P. Muir. *BACOL: B-spline Adaptive Collocation Software for 1-D Parabolic PDEs.* ACM Transactions on Mathematical Software (TOMS) 30.4 (2004), pp. 454–470.

[39] K. Yamazaki and X. Wang. *Global Stability and Uniform Persistence of the Reaction-Convection-Diffusion Cholera Epidemic Model.* arXiv Preprint (2017).

[40] J. Yang, S. Liang, and Y. Zhang. *Travelling Waves of a Delayed SIR Epidemic Model with Nonlinear Incidence Rate and Spatial Diffusion.* PLOS ONE 6 (2011), pp. 1–14.

[41] Eric J. et. al. *SciPy: Open Source Scientific Tools for Python.* 2001.