Threshold of Collapse

by

Samuel R. Baty and Peter J. Armijo
Team 62
Los Alamos High School, Los Alamos, NM 87544

Teacher: Mr. Lee Goodwin of Los Alamos High School.

Mentors: Dr. Don H. Tucker, Professor of Mathematics of the University of Utah, and Dr. Roy S. Baty of Los Alamos National Laboratory.

Abstract

This report details the results of numerical experiments involving models of threshold of collapse for economic systems and disease-host problems. An example is given showing the evolution of a disease for comparison of a healthy host with an unhealthy host. The Disease-Host model may be applied to simulate recovery, chronic illness and collapse (death) of the host.

1 Introduction

In the Threshold of Collapse project, a study of dynamical systems that approach stable equilibrium sets was performed. Example problems from economics and disease modeling were studied that have equilibrium points and lines to which the solutions tend. These problems either reach zero or non-zero equilibrium points. Collapse to zero represents the destruction of the system, or in biological models, the death of a patient or a population. The goal of the project was to model these circumstances and determine when subsistence or collapse results.

A seventh-order Runge-Kutta scheme was implemented to numerically integrate the model problems. The Economic/Population and Disease-Host problems are governed by non-linear ordinary differential equations (ODEs). A high-order numerical method was applied to handle the non-linear equations. The central theme in this project was to conduct numerical experiments to study the qualitative properties of ODEs. The qualitative properties were explored by plotting computed solutions of the dynamical systems in phase space to identify the equilibrium points.

The numerical scheme was coded using the language C. All plots for this project were generated using the software program Gnuplot. The computer that performed all calculations is a Mac with a 10.6.8 operating system. The C program developed for the project is listed in the Appendix.

2 Qualitative Theory of Ordinary Differential Equations

The qualitative theory of ordinary differential equations is the study of the global behavior of solutions to ODEs including their stability. Stability, in the context of this project, was defined as the long-time behavior of a solution of a dynamical system, Sanchez [1]. A stable system in a threshold of collapse

problem approaches an equilibrium set. The characteristics of a non-stable system would include the rate of change of the solution growing without bound, or the system suddenly exploding around a fixed value. In this study numerical experiments were used to determine whether or not the model problems are stable.

The analysis of all of the numerical experiments was done by plotting the solutions of the systems in phase space. Phase space is defined as a three-dimensional representation where the X and Y axes are solutions to a given ODE, while the Z axis, represents the evolution of the solution in time, Arnold [2]. The goal of the analysis was to discover the solutions that reach equilibrium points, and those systems that converge to zero, zero being one equilibrium point representing collapse.

A given system of ODEs represents a slope field in the region of phase space where the equations are defined. A solution of the system is a curve in phase space that is tangent to the slope field defined by the differential equations for each point on the curve.

3 Numerical Method

In this study a seventh-order Runge-Kutta-Fehlberg (RKF) scheme was implemented to integrate the threshold of collapse problems. This scheme employed variable stepsize control. High-order Runge-Kutta schemes were developed in astrodynamics, for example see the textbook of Battin [3].

The scheme applied in this study was developed by Fehlberg [4], and is defined by the following equations:

$$f_0 = f(x_0, y_0), \tag{1}$$

$$f_\kappa = f(x_0 + \alpha_\kappa h, y_0 + h\sum_{\lambda=0}^{\kappa-1} \beta_{\kappa\lambda} f_\lambda), \tag{2}$$

$$y = y_0 + h\sum_{\kappa=0}^{10} c_\kappa f_\kappa + 0(h^8), \tag{3}$$

and

$$\hat{y} = y_0 + h\sum_{\kappa=0}^{12} \hat{c}_\kappa f_\kappa + 0(h^9), \tag{4}$$

where h is the step size. In Equation (2), $\kappa = 1, 2, 3, ..., 12$. An ordinary differential equation is integrated numerically by applying Equations (1), (2), and (3) in an iterative fashion. Equation (1) represents the system of equations to be integrated at the initial data point, (x_0, y_0). Equations (3) and (4) are the seventh and eighth-order RKF schemes, respectively. Equation (3) was used to calculate the solution. Equation (4) was used to compute the stepsize update. In Equations (2), (3), and (4), there are quadrature constants, $\alpha_\kappa, \beta_{\kappa\lambda}, c_\kappa, \hat{c}_\kappa$, required by the numerical method, Fehlberg [4] (see page 65).

The seventh-order Runge-Kutta-Fehlberg scheme was used in this study because the solutions may have highly non-linear characteristics. For example, problems similar to the Economics/Population model studied in the following section may have unstable solutions that become arbitrarily large in finite time. The RKF numerical method allows the accurate calculation of data when the rate of change of the solution is large.

The RKF code developed for this project was checked against known solutions to a linear vibration problem. For an outline of the validation problem and its solution, see Baty and Armijo [5].

4 Threshold of Collapse

In this section, two threshold of collapse problems are presented: an Economics/Population problem and a Disease-Host problem. Numerical experi-

ments are outlined showing the qualitative properties of the solutions of these dynamical systems.

For threshold of collapse problems empirical data is exceptionally hard to obtain. These systems can be so complex that whole studies have to be performed just to get approximate values for the rate constants in the governing differential equations and for the initial data points. For the Economics/Population problem studied here, example empirical data can be found in the article of Johansen and Sornette [6].

4.1 Economics/Population Modeling

The first numerical experiments studied the time-evolution of a series of Economics/Population simulations. The analysis of the simulations plotted solutions in phase space for variable initial data. The following system of ordinary differential equations (ODEs) was used as the model for the Economics/Population problem:

$$\frac{dy}{dt} = \beta y(t)[N(t) - y(t)], \tag{5}$$

and

$$\frac{dN}{dt} = \begin{cases} \alpha[N(t) - y(t)] & \text{if } y(t) \geq N(t) \\ 0 & \text{if } y(t) < N(t) \end{cases}, \tag{6}$$

where α and β are rate constants. In Equations (5) and (6), $y(t)$ is the value of whatever is is being modeled, i.e., individuals in the population or financial factors in an economy; $N(t)$ is defined as the carrying capacity for the model, i.e., the number of individuals that the system can support, or the ideal combination of economic factors for stability in the system. For Equations (5) and (6), either the solutions will intersect one of the equilibrium lines, or the system will collapse to zero. For the numerical experiments presented here, the rate constants were fixed as: $\alpha = \beta = 1$.

In Figure 1, the dotted-line (tan curve) of positive slope represents a set of non-collapse equilibrium points within the problem. Three situations can occur depending on initial data. If an initial-data point is below the equilibrium line, then the solutions (green and red curves) will jump up to a point at which the system has a constant population that can be maintained. If the initial-data point is above the line, two events can occur:

1. Either the system has a low enough population that the solution curve will intersect the equilibrium line (brown and cyan curves), or

2. The solutions (blue and purple curves) will be so skewed that the curve will intersect the $y(t)$ axis before it reaches the equilibrium line and collapse to zero.

Figure 2 shows the time-evolution in phase space of example solutions, which converge to constant population states of Figure 1.

The Economics/Population model studied here follows the development and notes of Tucker [7].

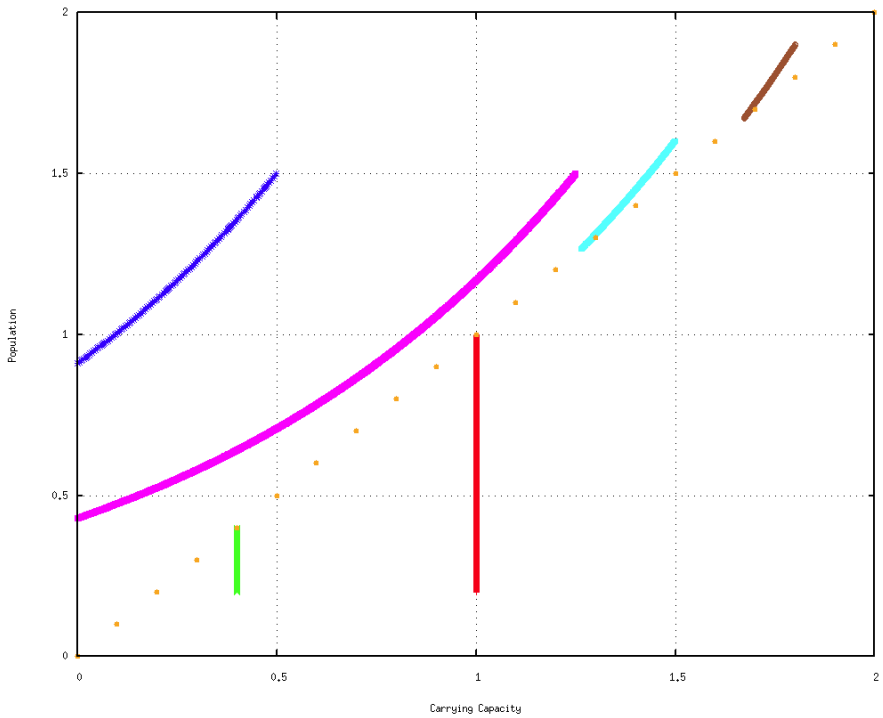

Figure 1: Typical solutions of the Economics/Population problem plotted in the $N(t)$-$y(t)$ plane showing the dotted, non-collapse, equilibrium line.

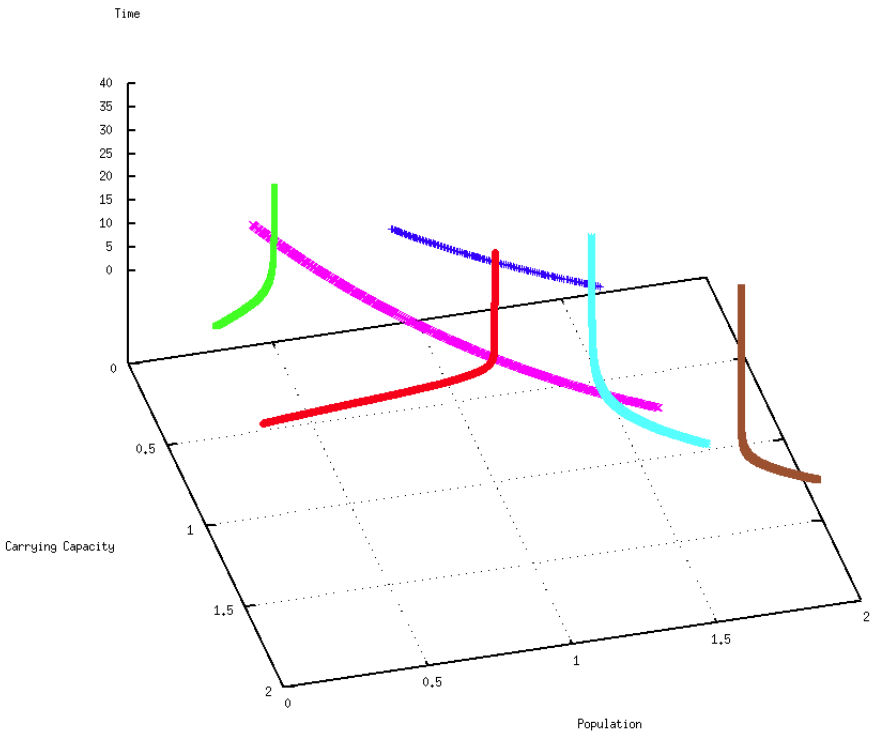

Figure 2: Typical solutions of the Economics/Population problem plotted in phase space. Here, the third axis is time.

4.2 Disease-Host Modeling

For the second problem in the threshold of collapse project, numerical experiments were run to study the time-evolution of a Disease-Host model. For the purposes of the simulations, idealized values were entered for the rate constants because of the lack of availability of measured data for disease-host interactions. The Disease-Host model follows the development and notes of Tucker [8].

The following system of ODEs was used to describe the evolution of the population of invaders and defenders for a disease-host model:

$$\frac{dx}{dt} = -kx^2(t) + k\mathbb{N}x(t) - lx(t)y(t), \tag{7}$$

and

$$\frac{dy}{dt} = m\mathbb{M}x(t) - (m + l_0)x(t)y(t), \tag{8}$$

where k, l, l_0, m are rate constants. In Equations (7) and (8), $x(t)$ is the population of invaders, and $y(t)$ is the population of defenders; moreover, $\mathbb{N}$ is the carrying capacity for the invaders, and $\mathbb{M}$ is the carrying capacity for the defenders. In the model, the invaders represent the population of a virus or a bacteria and the defenders represent the population of antibodies or T-cells. The carrying capacities $\mathbb{N}$ and $\mathbb{M}$ are specified as numbers near one for the simulations. The numerical experiments vary the relative magnitudes of the carrying capacities. In a real disease-host problem, the carrying capacity populations may be on the order of 10^{12}.

The solutions to the Disease-Host simulations fall into three basic categories. The first category ends with the patient recovering from the disease. The solution curve in this situation would have an equilibrium point with a relatively high $y(t)$ value in comparison with the $x(t)$ value. The second set of solutions involve a patient surviving, but never fully recovering. The solutions for this case converge to an equilibrium point, where the invaders and defenders are in balance. The final category of solutions occur when the

$x(t)$ value greatly outweighs some threshold $y(t)$ value (death). This system would converge to a small value of $y(t)$, signifying that there are not enough defensive elements and the host has died.

Figure 3 show solutions of the Disease-Host model for carrying capacities fixed at $\mathbb{M} = \mathbb{N} = 1$ and rate constants of $k, l, l_0, m = 0.01$. For these values two equilibrium sets occur:

1. The solutions converge to the equilibrium point $(0.5, 0.5)$ for which the patient survives but never fully recovers, or

2. The solutions converge to an equilibrium point above $(0.0, 1.0)$ on the vertical axis for which the patient recovers and the disease dies off.

The phase space region where the patient recovers is above the tan curve in the top left-hand corner of the plot of phase space in Figure 3. Figure 4 shows the time-evolution in phase space of the simulations of Figure 3. The values of initial data in Figures 3 and 4 where chosen to make the simulations easier to plot. Initial values larger than $\mathbb{M} = \mathbb{N} = 1$ are not realistic in most cases.

An important application of a computational model like the Disease-Host model is being able to estimate the long time result of an illness. Figure 5 shows a comparison of a healthy host and an unhealthy host with a disease. The blue, red and green curves in the figure are for a healthy host with carrying capacities $\mathbb{M} = \mathbb{N} = 1$, rate constants $k, l, m = 0.01$, and $l_0 = 0.05$. Here l_0 is a rate constant reducing the number of defenders. The brown, cyan and purple curves all represent situations for an unhealthy host with carrying capacities of: $\mathbb{N} = 1$ and $\mathbb{M} = 0.7$. The box outlined by tan stars is an example of a region where the disease kills the host if $y(t) < 0.15$ for $x(t) > 0.5$. The death region was chosen arbitrarily for the given simulations and could be changed to fit specific circumstances. For these simulations, the host dies for the purple and cyan curves.

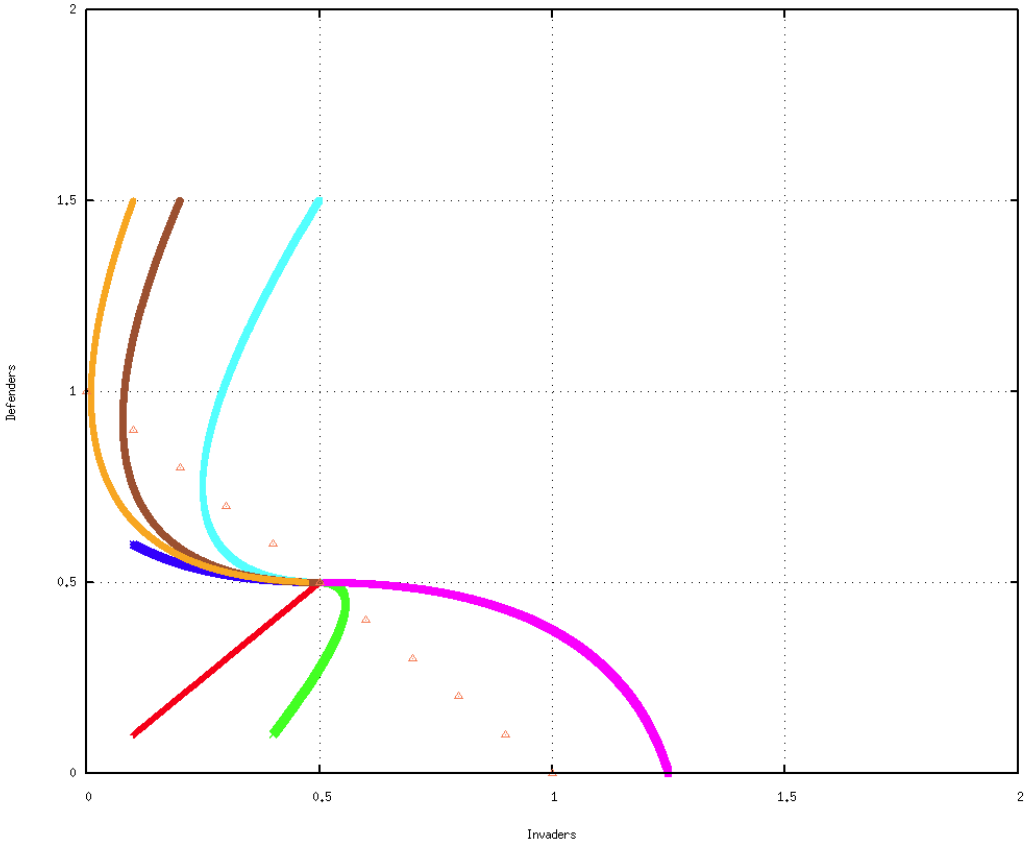

Figure 3: Typical solutions of the Disease-Host problem plotted in phase space showing a chronic-illness, equilibrium point at (0.5,0.5).

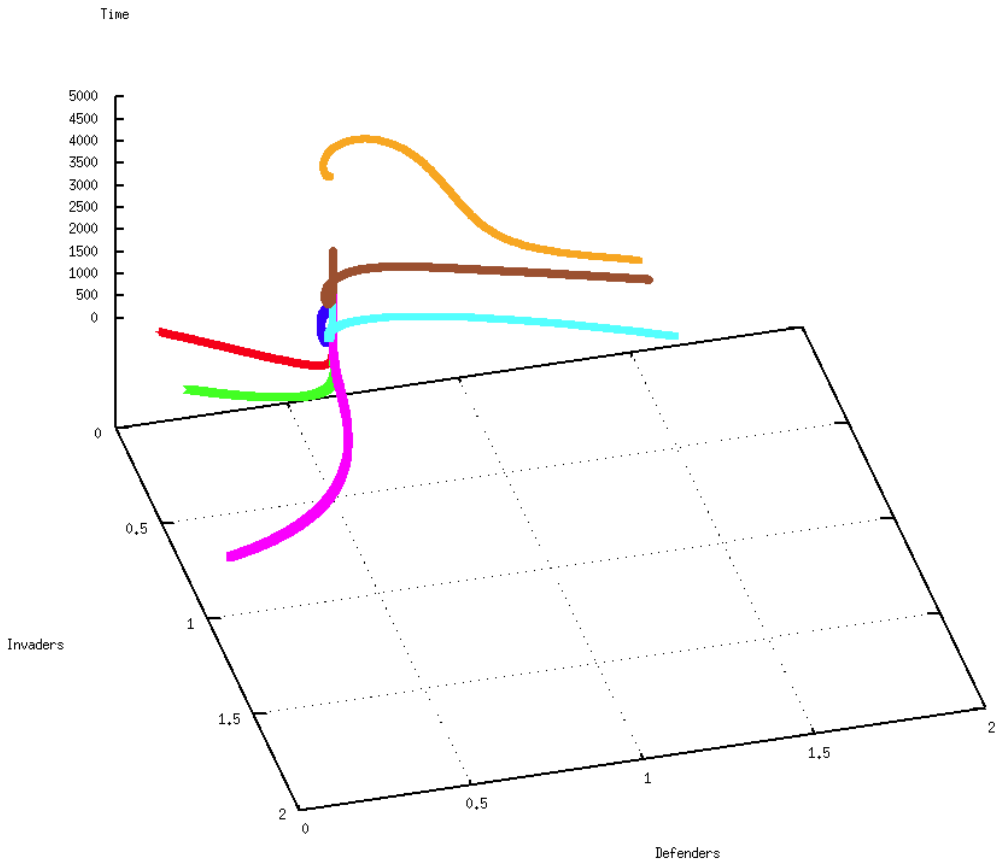

Figure 4: Typical solutions of the Disease-Host problem plotted in phase space showing the evolution of the solutions in time.

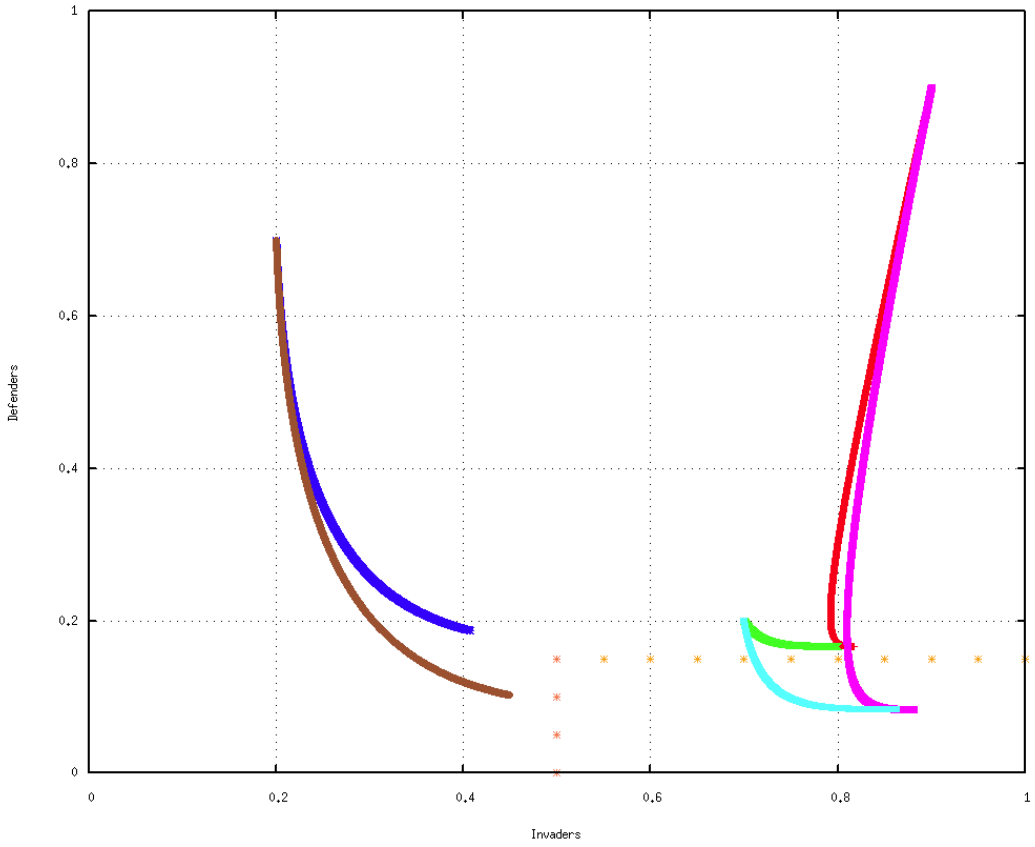

Figure 5: Simulations to compare a healthy host, $\mathbb{M} = 1.0$, with a unhealthy host, $\mathbb{M} = 0.7$. The box outlined by tan stars represents the death region, where the disease kills the host.

5 Summary and Conclusions

This project studied two threshold of collapse problems: an Economics/ Population problem and a Disease-Host problem. These problems were simulated using a seventh-order Runge-Kutta-Fehlberg scheme. The solutions of the collapse problems were plotted in phase space to explore their qualitative behavior.

In the model problems, three long-time behaviors appeared to present in all simulations. For example in the disease-host simulations, the patient may recover from a disease, the patient may live with a chronic-illness, or the patient may die. All three outcomes represent stable solutions of the governing system of ordinary differential equations. For the Disease-Host model an important application is to determine if the host dies (collapse) or if the host lives (non-collapse). This project studied idealized cases of the Economics/Population and Disease-Host problems because the data (rate constants) for these simulations are very hard to find and are very complex. Such models may also exhibit complex behavior including stable and unstable rapid growth and decay.

The original threshold of collapse models were developed by Don Tucker. Sam Baty and Peter Armijo, along with help from Roy Baty, developed a computer model and code to perform the calculations. A key concept that Sam and Peter developed while doing the project was the idea of using the Disease-Host model to compare solutions of a healthy host with solutions for an unhealthy host. Such a comparison allows the analysis of a fixed disease in the context of other medical conditions, e.g., a heart condition. One of the key things that the team learned was that conclusions can be drawn from the qualitative analysis of a problem, without knowing the exact or numerical answer. The qualitative study of ODEs can give the fundamental behavior of a problem evolving in time without the solution.

References

[1] Sanchez, D. A., **Ordinary Differential Equations and Stability Theory: An Introduction**, Dover Publications Inc., New York, 1968.

[2] Arnold, V. I., **Ordinary Differential Equations**, The MIT Press, Boston, 1973.

[3] Battin, R. H. **An Introduction to the Mathematics and Methods of Astrodynamics**, AIAA Education Series, New York, 1987.

[4] Fehlberg, E., "Classical Fifth-, Sixth-, Seventh-, and Eighth-Order Runge-Kutta Formulas with Stepsize Control," NASA TR R-287, 1968.

[5] Baty, S. R., and Armijo, P. J., "Astrophysical N-Body Simulations of Star Clusters," Final Report, Team 68, Supercomputing Challenge 2009-2010.

[6] Johansen, A., and Sornette, D., "Finite-Time Singularity in the Dynamics of the World Population, Economic and Financial Indices," *Physica A*, **294**, 2001, pp. 465-502.

[7] Tucker, D. H., "Notes on: Threshold of Collapse," University of Utah, Department of Mathematics, June 2011.

[8] Tucker, D. H., "Notes on: An Evolving Model for Disease in a Host," University of Utah, Department of Mathematics, November 2011.

Appendix: Code Listing

```
/*  Threshold Collapse and Finite Time Singularity Code  */
/*  Seventh Order Variable Step Size Runge-Kutta-Felhberg
Scheme */

#include <stdio.h>
#include <math.h>

double fctn(int j, double t, double dt0, double y[]);

int main()
{
    int i, j, k, its;
    double err, tol;
    double alfa[13], c[11], chat[13], error[100];
    double beta0[13], beta1[13], beta2[13], beta3[13],
beta4[13], beta5[13], beta6[13];
    double beta7[13], beta8[13], beta9[13], beta10[13],
beta11[13], beta12[13];
    double f0[3], f1[3], f2[3], f3[3], f4[3], f5[3], f6[3];
    double f7[3], f8[3], f9[3], f10[3], f11[3], f12[3];
    double y0[100], y[100], ya1[100], ya2[100], yhat[100];
    double dt, dt0, t, t0;
    double delta, order, errY, errN;

    /* Initialize Parameters for Seventh Order Runge-Kutta-
Fehlberg Scheme */

    alfa[0]=0.0;
    alfa[1]=2.0/27.0;
    alfa[2]=1.0/9.0;
    alfa[3]=1.0/6.0;
    alfa[4]=5.0/12.0;
    alfa[5]=1.0/2.0;
    alfa[6]=5.0/6.0;
    alfa[7]=1.0/6.0;
    alfa[8]=2.0/3.0;
    alfa[9]=1.0/3.0;
    alfa[10]=1.0;
    alfa[11]=0.0;
    alfa[12]=1.0;

    c[0]=41.0/840.0;
    c[1]=0.0;
    c[2]=0.0;
    c[3]=0.0;
    c[4]=0.0;
    c[5]=34.0/105.0;
    c[6]=9.0/35.0;
```

```
c[7]=9.0/35.0;
c[8]=9.0/280.0;
c[9]=9.0/280.0;
c[10]=41.0/840.0;

chat[0]=0.0;
chat[1]=0.0;
chat[2]=0.0;
chat[3]=0.0;
chat[4]=0.0;
chat[5]=34.0/105.0;
chat[6]=9.0/35.0;
chat[7]=9.0/35.0;
chat[8]=9.0/280.0;
chat[9]=9.0/280.0;
chat[10]=0.0;
chat[11]=41.0/840.0;
chat[12]=41.0/840.0;

for(i=0;i<=12;i++){
 beta0[i]=0.0;
 beta1[i]=0.0;
 beta2[i]=0.0;
 beta3[i]=0.0;
 beta4[i]=0.0;
 beta5[i]=0.0;
 beta6[i]=0.0;
 beta7[i]=0.0;
 beta8[i]=0.0;
 beta9[i]=0.0;
 beta10[i]=0.0;
 beta11[i]=0.0;
 beta12[i]=0.0;
}

beta0[0]=0.0;

beta1[0]=2.0/27.0;

beta2[0]=1.0/36.0;
beta2[1]=1.0/12.0;

beta3[0]=1.0/24.0;
beta3[1]=0.0;
beta3[2]=1.0/8.0;

beta4[0]=5.0/12.0;
beta4[1]=0.0;
beta4[2]=-25.0/16.0;
```

Code Page 2/10

```
beta4[3]=25.0/16.0;

beta5[0]=1.0/20.0;
beta5[1]=0.0;
beta5[2]=0.0;
beta5[3]=1.0/4.0;
beta5[4]=1.0/5.0;

beta6[0]=-25.0/108.0;
beta6[1]=0.0;
beta6[2]=0.0;
beta6[3]=125.0/108.0;
beta6[4]=-65.0/27.0;
beta6[5]=125.0/54.0;

beta7[0]=31.0/300.0;
beta7[1]=0.0;
beta7[2]=0.0;
beta7[3]=0.0;
beta7[4]=61.0/225.0;
beta7[5]=-2.0/9.0;
beta7[6]=13.0/900.0;

beta8[0]=2.0;
beta8[1]=0.0;
beta8[2]=0.0;
beta8[3]=-53.0/6.0;
beta8[4]=704.0/45.0;
beta8[5]=-107.0/9.0;
beta8[6]=67.0/90.0;
beta8[7]=3.0;

beta9[0]=-91.0/108.0;
beta9[1]=0.0;
beta9[2]=0.0;
beta9[3]=23.0/108.0;
beta9[4]=-976.0/135.0;
beta9[5]=311.0/54.0;
beta9[6]=-19.0/60.0;
beta9[7]=17.0/6.0;
beta9[8]=-1.0/12.0;

beta10[0]=2383.0/4100.0;
beta10[1]=0.0;
beta10[2]=0.0;
beta10[3]=-341.0/164.0;
beta10[4]=4496.0/1025.0;
beta10[5]=-301.0/82.0;
beta10[6]=2133.0/4100.0;
```

```
beta10[7]=45.0/82.0;
beta10[8]=45.0/164.0;
beta10[9]=18.0/41.0;

beta11[0]=3.0/205;
beta11[1]=0.0;
beta11[2]=0.0;
beta11[3]=0.0;
beta11[4]=0.0;
beta11[5]=-6.0/41.0;
beta11[6]=-3.0/205.0;
beta11[7]=-3.0/41.0;
beta11[8]=3.0/41.0;
beta11[9]=6.0/41.0;
beta11[10]=0.0;

beta12[0]=-1777.0/4100.0;
beta12[1]=0.0;
beta12[2]=0.0;
beta12[3]=-341.0/164.0;
beta12[4]=4496.0/1025.0;
beta12[5]=-289.0/82.0;
beta12[6]=2193.0/4100.0;
beta12[7]=51.0/82.0;
beta12[8]=33.0/164.0;
beta12[9]=12.0/41.0;
beta12[10]=0.0;
beta12[11]=1.0;

/* Perform Integration */
/* Integration Parameters */

t=0.0;
dt=0.0005;
dt0=dt;

tol=0.0000000001;
delta=0.0;

its=10000;

for(i=1;i<=3;i++){
 f0[i]=0.0;
 f1[i]=0.0;
    f2[i]=0.0;
 f3[i]=0.0;
    f4[i]=0.0;
 f5[i]=0.0;
 f6[i]=0.0;
```

Code Page 4/10

```
      f7[i]=0.0;
      f8[i]=0.0;
      f9[i]=0.0;
      f10[i]=0.0;
      f11[i]=0.0;
      f12[i]=0.0;
     }

     /* Define Initial Data */

     y0[0]=0.0;

     /* Input Initial Data Here */

     y0[1]=1.90;
     y0[2]=1.80;

     for(i=1;i<=2;i++){
     y[i]=y0[i];
     }

     t=t0;

     /* Integration Loop Here */

     for(j=1;j<=its;j++){

     /* Compute F0 */

     t=t0+alfa[0]*dt;

     for(i=1;i<=2;i++){y[i]=y0[i];}
     for(i=1;i<=2;i++){f0[i]=fctn(i, t, dt0, y);}


     /* Compute F1 */

     t=t0+alfa[1]*dt;

     for(i=1;i<=2;i++){y[i]=y0[i]+dt*beta1[0]*f0[i];}
     for(i=1;i<=2;i++){f1[i]=fctn(i, t, dt0, y);}


     /* Compute F2 */

     t=t0+alfa[2]*dt;

     for(i=1;i<=2;i++){y[i]=y0[i]+dt*(beta2[0]*f0[i]+beta2[1]*
f1[i]);}
```

Code Page 5/10

```
    for(i=1;i<=2;i++){f2[i]=fctn(i, t, dt0, y);}

    /* Compute F3 */

    t=t0+alfa[3]*dt;

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(beta3[0]*f0[i]+beta3[1]*
f1[i]+beta3[2]*f2[i]);}
    for(i=1;i<=2;i++){f3[i]=fctn(i, t, dt0, y);}


    /* Compute F4 */

    t=t0+alfa[4]*dt;

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(beta4[0]*f0[i]+beta4[1]*
f1[i]+beta4[2]*f2[i]+beta4[3]*f3[i]);}
    for(i=1;i<=2;i++){f4[i]=fctn(i, t, dt0, y);}


    /* Compute F5 */

    t=t0+alfa[5]*dt;

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(beta5[0]*f0[i]+beta5[1]*
f1[i]+beta5[2]*f2[i]+beta5[3]*f3[i]+beta5[4]*f4[i]);}
    for(i=1;i<=2;i++){f5[i]=fctn(i, t, dt0, y);}


    /* Compute F6 */

    t=t0+alfa[6]*dt;

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(beta6[0]*f0[i]+beta6[1]*
f1[i]+beta6[2]*f2[i]+beta6[3]*f3[i]+beta6[4]*f4[i]+beta6[5]*f
5[i]);}
    for(i=1;i<=2;i++){f6[i]=fctn(i, t, dt0, y);}


    /* Compute F7 */

    t=t0+alfa[7]*dt;

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(beta7[0]*f0[i]+beta7[1]*
f1[i]+beta7[2]*f2[i]+beta7[3]*f3[i]+beta7[4]*f4[i]+beta7[5]*f
5[i]+beta7[6]*f6[i]);}
    for(i=1;i<=2;i++){f7[i]=fctn(i, t, dt0, y);}
```

```
    /* Compute F8 */

    t=t0+alfa[8]*dt;

    for(i=1;i<=2;i++){
    ya1[i]=beta8[0]*f0[i]+beta8[1]*f1[i]+beta8[2]*f2[i]+beta8
[3]*f3[i]+beta8[4]*f4[i]+beta8[5]*f5[i]+beta8[6]*f6[i];
    ya2[i]=beta8[7]*f7[i];}

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(ya1[i]+ya2[i]);}
    for(i=1;i<=2;i++){f8[i]=fctn(i, t, dt0, y);}


    /* Compute F9 */

    t=t0+alfa[9]*dt;

    for(i=1;i<=2;i++){
    ya1[i]=beta9[0]*f0[i]+beta9[1]*f1[i]+beta9[2]*f2[i]+beta9
[3]*f3[i]+beta9[4]*f4[i]+beta9[5]*f5[i]+beta9[6]*f6[i];
    ya2[i]=beta9[7]*f7[i]+beta9[8]*f8[i];}

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(ya1[i]+ya2[i]);}
    for(i=1;i<=2;i++){f9[i]=fctn(i, t, dt0, y);}


    /* Compute F10 */

    t=t0+alfa[10]*dt;

    for(i=1;i<=2;i++){
    ya1[i]=beta10[0]*f0[i]+beta10[1]*f1[i]+beta10[2]*f2[i]+be
ta10[3]*f3[i]+beta10[4]*f4[i]+beta10[5]*f5[i]+beta10[6]*f6[i]
;
    ya2[i]=beta10[7]*f7[i]+beta10[8]*f8[i]+beta10[9]*f9[i];}

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(ya1[i]+ya2[i]);}
    for(i=1;i<=2;i++){f10[i]=fctn(i, t, dt0, y);}


    /* Compute F11 */

    t=t0+alfa[11]*dt;

    for(i=1;i<=2;i++){
    ya1[i]=beta11[0]*f0[i]+beta11[1]*f1[i]+beta11[2]*f2[i]+be
ta11[3]*f3[i]+beta11[4]*f4[i]+beta11[5]*f5[i]+beta11[6]*f6[i]
;
    ya2[i]=beta11[7]*f7[i]+beta11[8]*f8[i]+beta11[9]*f9[i]+be
```

```
ta11[10]*f10[i];}

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(ya1[i]+ya2[i]);}
    for(i=1;i<=2;i++){f11[i]=fctn(i, t, dt0, y);}


    /* Compute F12 */

    t=t0+alfa[12]*dt;

    for(i=1;i<=2;i++){
    ya1[i]=beta12[0]*f0[i]+beta12[1]*f1[i]+beta12[2]*f2[i]+be
ta12[3]*f3[i]+beta12[4]*f4[i]+beta12[5]*f5[i]+beta12[6]*f6[i]
;
    ya2[i]=beta12[7]*f7[i]+beta12[8]*f8[i]+beta12[9]*f9[i]+be
ta12[10]*f10[i]+beta12[11]*f11[i];}

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*(ya1[i]+ya2[i]);}
    for(i=1;i<=2;i++){f12[i]=fctn(i, t, dt0, y);}


    /* Advance Solution in Time by dt */

    for(i=1;i<=2;i++){
    ya1[i]=c[0]*f0[i]+c[1]*f1[i]+c[2]*f2[i]+c[3]*f3[i]+c[4]*f
4[i]+c[5]*f5[i];
    ya2[i]=c[6]*f6[i]+c[7]*f7[i]+c[8]*f8[i]+c[9]*f9[i]+c[10]*
f10[i];}

    for(i=1;i<=2;i++){y[i]=ya1[i]+ya2[i];}


    /* Compute Solution +1 Order for Variable Step */

    for(i=1;i<=2;i++){
    ya1[i]=chat[0]*f0[i]+chat[1]*f1[i]+chat[2]*f2[i]+chat[3]*
f3[i]+chat[4]*f4[i]+chat[5]*f5[i];
    ya2[i]=chat[6]*f6[i]+chat[7]*f7[i]+chat[8]*f8[i]+chat[9]*
f9[i]+chat[10]*f10[i]+chat[11]*f11[i]+chat[12]*f12[i];}

    for(i=1;i<=2;i++){yhat[i]=ya1[i]+ya2[i];}

    /* Error for Step Size Update */

    for(i=1;i<=2;i++){error[i]=y[i]-yhat[i];}

    err=0.0;
    for(i=1;i<=2;i++){err+=error[i]*error[i];}
    err=sqrt(err);
```

Code Page 8/10

```
    /* Solution Time Advance */

    for(i=1;i<=2;i++){y[i]=y0[i]+dt*y[i];}

    /* Solution for Next Time Step */

    t0=t0+dt;

    for(i=1;i<=2;i++){y0[i]=y[i];}

    /* Error for Simple Oscillator */

     /* errY=y[1]-cos(t0);
     errN=y[2]-sin(t0); */


    /* Print Output */

    /* printf("%12u %14.6e %14.6e %14.6e %14.6e %14.6e
%14.6e %14.6e\n", j, t0, err, dt, y[1], y[2], errY, errN); */

    printf("%12u %14.6e %14.6e %14.6e %14.6e %14.6e\n", j,
t0, err, dt, y[1], y[2]);


    /* Update Step Size */

    if(err != 0.0){delta=tol/err;}
    if(err == 0.0){delta=1.0;}
    order=1.0/7.0;
    delta=0.84*pow(delta,order);

    if(dt <= 0.1){
    dt=delta*dt;
    if(delta <= 0.01) {dt=0.01*dt;}
    if(delta >= 4.00) {dt=4.00*dt;}}
    if(dt > 10.0*dt0){dt=10.0*dt0;}
    }

    return 0;
}

double fctn(int j, double t, double dt0, double y[])
{
    double a, b, f;
    double kr, lr, l0, nr, mr1, mr2;

    /* Population Model */
```

Code Page 9/10

```
    a=1.0;
    b=1.0;

    if (j == 1) {f=b*y[1]*(y[2]-y[1]);}
    if (j == 2) {
     if (y[2] > 0.01*dt0) {
            if (y[1] < y[2]) {f=0.0;}
          if (y[1] >= y[2]) {f=-a*(y[1]-y[2]);}}
     if (y[2] <= 0.01*dt0) {f=0.0;}
    }

    /* Simple Oscillator */

     /* if (j==1) {f=-y[2];}
     if (j==2) {f=y[1];}  */

    /* Disease Host Model */

     /* kr=0.01;
     lr=0.01;
     l0=0.01;
     nr=1.0;
     mr1=0.01;
     mr2=1.0;

     if (j == 1) {f=-kr*y[1]*y[1]+kr*nr*y[1]-lr*y[1]*y[2];}
     if (j == 2) {f=mr1*mr2*y[1]-(mr1+l0)*y[1]*y[2];} */


    return f;
}
```

Code Page 10/10