Ilia Kheirkhah

ENM 5020

4/11/2023

Homework 3

Solutions to the Modified Lotka-Volterra System of Differential Equations using Implicit Euler Integration

Contents

Introduction

The tools we have developed now have given us a method to find solutions to linear and nonlinear systems of equations. These problems took two forms: $Ax = b$ or $R(x) = 0$ respectively. We have shown that these tools are useful in solving steady state differential equations. However, many problems cannot be modeled as steady and evolve in time or some other variable. In these cases, we would like to integrate over some time scale. These problems, often written as $\dot{x} = f(x)$, are very common in many physical applications. Developing a method to solve these systems if the focus of this assignment.

In exploring this process, we will look at a Modified Lotka-Volterra System. These can be modeled with the following equations:

$$\frac{dx}{dt} = (a - by)x - px^2 \quad \textbf{(1)}$$

$$\frac{dy}{dt} = (cx - d)y - qy^2 \quad \textbf{(2)}$$

This system is used to model the population of a Predator-Prey interaction. The parameters $\{a, b, c, d\} > 0$ represent different interactions between the species while $\{p.q\} > 0$ represent the effects of overcrowding and competition. Given an initial condition, this model can predict how the populations will evolve over time.

Notice that this system is a nonlinear system of differential equations. These cannot, generally, be solved analytically. However, numerical solvers do not have this limitation and can give us solutions to almost any system. However, this comes with a price. As with all numeric methods, we pay the cost of accuracy. As we will discuss later, the method we will use to solve this problem numerically will have proportional to $O(\Delta t^2)$ locally and $O(\Delta t)$ globally. Higher accuracy methods will cost us more time and complexity. In addition to this intrinsic problem with numeric methods, numeric integrators have to worry about stability.

There are two major types of stability we will talk about. First, there is physical stability. This is associated with the nature of differential equations. For example, we know that some differential equations have solutions of e^x. In these cases, the solution blows up to infinity as time evolves. This is a physically unstable system. Regardless of the accuracy of our numeric system, any error we accrue will endlessly grow. These systems cannot reasonably be solved with numeric approaches.

Other differential equations, such as those with solution e^{-x}, are known as physically stable. In cases where we have a stable system, any error we accrue will not grow and should instead shrink over time. Given that we use the proper numeric scheme, we can get good results for physically stable problems. For any system we will study, including our Lotka-Volterra System, will be physically stable.

The other type of stability we care for is numeric stability. The question we seek to answer here is what step size, Δt, we can take in integrating without diverging. This divergence happens as a intrinsic property of some numeric schemes. As we will see, Explicit methods (Those depending on prior, known information) will have some maximum step size while Implicit methods (Those depending on future, unknown information) will be unbounded.

Methods

In this section we will first explore the nature of this differential equation. This will allow us to ensure that our system is stable. From here, we will then explore two numeric schemes: Implicit and Explicit Euler Integration. We will discuss their differences and how they will be applied to our problems.

Critical Points

Let us first explore the critical points of our differential equations. These points, also known as the steady state points, have the condition that $\dot{x} = 0$. If we start at or reach one of these points, any further integration in time will result in no change of state. To solve for our steady state solutions, we have to find the solution to the following system of equations:

$$0 = (a - by)x - px^2 \quad \textbf{(3)}$$

$$0 = (cx - d)y - qy^2 \quad \textbf{(4)}$$

By factoring appropriately we can break this into two equations. Equation (3) and (4)can be factored into:

$$0 = x(a - by - px) \quad \textbf{(5)}$$

$$0 = y(cx - d - qy) \quad \textbf{(6)}$$

Each equation has 2 solutions. The first is when the term outside the parenthesis is zero. The second is when the term inside the parenthesis is zero. This gives us a total of four total critical points. Let us go case by case and find each solution.

Trivial Solution

The first solution we should discuss happens when we look at the terms outside the parenthesis on both equations. It is then clear to see that a solution to this system is:

$$x = 0, y = 0 \quad \textbf{(7)}$$

This first solution is a trivial solution as it is clear that if we have no predators or prey, their population will not change over time.

X=0 Solution

The next solution we can look at is when we consider when $x = 0$ and see what the term in parentheses for equation (6) will result in. Plugging $x = 0$ into that equation, we see:

$$-d - qy = 0 \rightarrow y = -\frac{d}{q} \quad \textbf{(8)}$$

Since both d and q are both positive integers, this solution corresponds to a case when y, or the population of predators, is negative. This is clearly not physical, and therefore we will ignore it in the future. In a similar vein, we expect our populations to remain positive numbers. As such, we will only look at solutions in the first quadrant for all future analysis.

Y=0 Solution

Doing a similar process as before, we can arrive at the following equation and solution:

$$a - px = 0 \rightarrow x = \frac{a}{p} \quad \textbf{(9)}$$

For positive values of a, p, this solution will be within our domain of interest. However, once we start doing analysis of the problem with provided parameters, we will set $p = q = 0$. In this case, this steady state solution does not exist, and therefore we will actually not see its influence in our phase diagram. However, for more general system this point does exist.

Non-Trivial Solution

The previous two steady state solutions have modeled the necessary population of predators or prey that will achieve steady state when the populations do not interact. However, we are more interested in the behavior of the system when both populations are involved. This solution, where we do not set either

population to zero, is the most interesting solution to find. For this solution, we have the following system of equations:

$$a - by - px = 0 \quad \textbf{(10)}$$
$$cx - d - qy = 0 \quad \textbf{(11)}$$

This is a simple linear system of equations. Writing this in the form $Ax = b$ we can see that:

$$\begin{bmatrix} p & b \\ c & -q \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \\ d \end{pmatrix} \quad \textbf{(12)}$$

This can be solved by simply inverting the matrix. This gives us:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{pq + bc} \begin{bmatrix} q & b \\ c & -p \end{bmatrix} \begin{pmatrix} a \\ d \end{pmatrix} \quad \textbf{(13)}$$

Multiplying this out, we get:

$$x = \frac{aq + bd}{pq + bc}, y = \frac{ac - pd}{pq + bc} \quad \textbf{(14)}$$

Notice that if all of these are positive integers, these will both be nonzero solutions. This comes with the exception of if $ac = pd$. However, these will generally give us positive solutions.

For our real system, where $a = b = c = d = 1$ and $p = q = 0$ we can see that this solution becomes:

$$x = y = 1 \quad \textbf{(15)}$$

This is the primary steady state point that influences our system.

Recap

Compiling the results of the previous sections, we can look to Table 1 to get a good idea of how our system will behave. This analysis allowed us to see how one system can have many different steady state solutions. In addition, this shows the importance of understanding the physical system you are attempting to analyze. The Non-Physical Solution we found is, mathematically, a true steady state point. However, since we are dealing with populations, which cannot be negative, we knew that this was a case of math not representing life perfectly.

	$x = 0$	$a - by - px = 0$
$y = 0$	Trivial Solution (0,0)	Divergent Solution $\left(\frac{a}{p}, 0\right)$
$cx - d - qy = 0$	Non-Physical Solution $\left(0, -\frac{d}{q}\right)$	Non-Trivial Solution $\left(\frac{aq + bd}{pq + bc}, \frac{ac - pd}{pq + bc}\right)$

Table 1 – Steady State Solutions in the form (x,y)

Now that we have a good idea of the behavior of our system, we can now look at numeric methods for solving our system to look at states beyond the those which are steady.

Implicit Euler

Given a problem of the form $\dot{x} = f(x)$, we need a way of integrating this function numerically. If we want to integrate this from t_n to t_{n+1} we can then say:

$$\int_{t_n}^{t_{n+1}} \frac{dx}{dt}\, dt = \int_{t_n}^{t_{n+1}} f(x)\, dt \quad \textbf{(16)}$$

$$x_{n+1} = x_n + \int_{t_n}^{t_{n+1}} f(x)\, dt \tag{17}$$

Now if we apply the substitution $\alpha = \frac{t-t_n}{\Delta t}$ where $\Delta t = t_{n+1} - t_n$ the above integral can become:

$$x_{n+1} = x_n + \Delta t \int_0^1 \frac{dx(\alpha)}{dt} d\alpha \tag{18}$$

At this point, we need a method of calculating the integral at this point. Requires us to do some level of interpolation and potentially extrapolation. If we apply most simple method of doing this, such that $\frac{dx(\alpha)}{dt} = x_n' = f(x_n)$, we can then find the simples method of numeric integration, also known as Explicit Euler Integration:

$$x_{n+1} = x_n + \Delta t f(x_n) + O(\Delta t^2) \tag{19}$$

This took advantage of claiming the derivative says constant moving forward from x_n. This is like doing a flat line extrapolation scheme. A benefit of these schemes is that it is simple. This requires only vector addition and vector scaling, making it a very computationally cheap process. However, the problem with these methods is that they are not perfectly stable. As can be shown, a primary condition for convergence of the Explicit Euler Method is:

$$0 \leq \lambda_{max} \Delta t \leq 2 \tag{20}$$

Where λ_{max} is the largest magnitude of eigenvalues of our system. If we do not meet this criterion, our numeric scheme can diverge regardless of our physical stability. This can be an issue for multidimensional systems of differential equations. To fix this, we can shift to Implicit Integration.

To do this, instead of making our interpolation function a flat line equal to the value at x_n, we will make it a flat line equal to the value of x_{n+1}. Even though we do not know this value currently, we will deal with the problem in a moment. For now, notice that this changes the form of our equation. To be more precise, Equation (19) changes into the following form:

$$x_{n+1} = x_n + \Delta t f(x_{n+1}) + O(\Delta t^2) \tag{21}$$

Notice that this change has not cost us any accuracy. In addition, if we do a stability analysis, we will notice that the condition we have is:

$$\left| \frac{1}{1 + \lambda \Delta t} \right| < 1 \tag{22}$$

For this derivation, we assume that we have positive eigenvalues. This means that we have unconditional stability when using Implicit Integration.

So far we have gained unconditional stability with the same level of accuracy. However, nothing comes for free. In this case, let us return to Equation (21). If we ignore the higher order terms and put everything to one side, we can see:

$$x_{n+1} - x_n - \Delta t f(x_{n+1}) = R(x_{n+1}) = 0 \tag{23}$$

This is generally a nonlinear system of equations that needs to be solved using Newton's Method. This is the cost we pay for unconditional stability.

Newton's Method for the Modified Lotka-Volterra System

Here, let us explore exactly how we will apply Newton's method to our system. Recall Equations (1) and (2) which gave us the form of the Modified Lotka-Volterra System and Equation (23) which gives us the nonlinear equation we need to solve. If we expand (23) into vector form, we get the following equations:

$$R(x_{n+1}) = \begin{pmatrix} x_{n+1} - x_n - \Delta t(a - by_{n+1})x_{n+1} - \Delta t p x_{n+1}^2 \\ y_{n+1} - y_n - \Delta t(cx_{n+1} - d)y_{n+1} - \Delta t q y_{n+1}^2 \end{pmatrix} \quad \textbf{(24)}$$

To set up Newton's Method, we need to find the Jacobian Matrix and set up the following equation:

$$J\delta x = -R \quad \textbf{(25)}$$

To calculate the Jacobian Matrix, we need to calculate each partial derivative as shown below:

$$J_{11} = \frac{\partial R_1}{\partial x_{n+1}} = 1 - \Delta t(a - by_{n+1}) - 2\Delta t p x_{n+1} \quad \textbf{(26)}$$

$$J_{12} = \frac{\partial R_1}{\partial y_{n+1}} = \Delta t b x_{n+1} \quad \textbf{(27)}$$

$$J_{21} = \frac{\partial R_2}{\partial x_{n+1}} = -\Delta t c y_{n+1} \quad \textbf{(28)}$$

$$J_{22} = \frac{\partial R_2}{\partial y_{n+1}} = 1 - \Delta t(cx_{n+1} - d) - 2\Delta t q y_{n+1} \quad \textbf{(29)}$$

At this point, all that is left is to determine a way of obtaining initial guesses for Newton's Method. Although some continuation scheme could be used to get faster convergence, we will simply use the converged solution at the previous time step as the initial guess for the next time step. This is to say $x_{n+1}^0 = x_n$. This is a simple method to implement and, for sufficiently small time-steps, will converge sufficiently fast. This will always work as at the first time-step, we will have an initial condition to go off of.

Results

In this section we will implement the discussed theory in MATLAB. First, we will explore our error scaling. We will then use some of this information to pick a step size we are comfortable with and then explore the time evolution and phase diagram of our Lotka-Volterra system.

Error

First is the error analysis. To find how our error scales, we first take a time step of Δt. This gives us a result that has error Δt^2. From here, we will then divide our step size by two. However, to reach the same time such that we can compare our results, we will need to take two time steps of this smaller time step. Doing this, we get error on the order of $2 \cdot \left(\frac{\Delta t}{2}\right)^2 = \frac{\Delta t^2}{2}$. Given this, we can how find the change

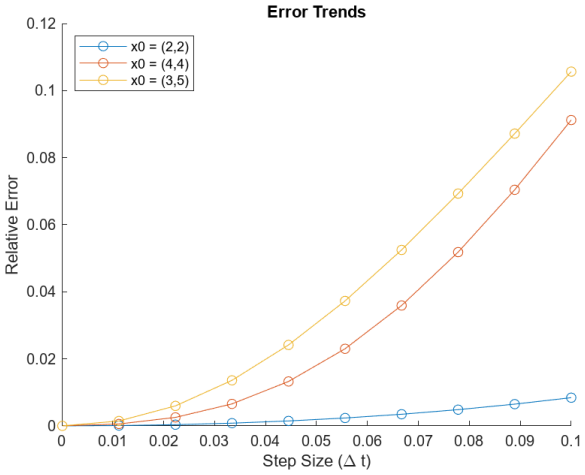

Figure 1 – True Error Graphs for Various Points

in error here to be $\frac{\Delta t^2}{2}$. Repeating this process for various step sizes, we can get a graph that shows our quadratic error trends. As we can see in Figure 1, all our graphs look roughly quadratic. Figure 2 below confirms this as we plot this data on a log-log graph which allows us to see the exponent of these curves as the slope of the log graph. This is a technique we have used since Assignment 1 so I do not believe there is a need to review this theory. As can be seen in Figure 2, all the plots have error slopes of roughly 2, confirming our Δt^2 error trends.

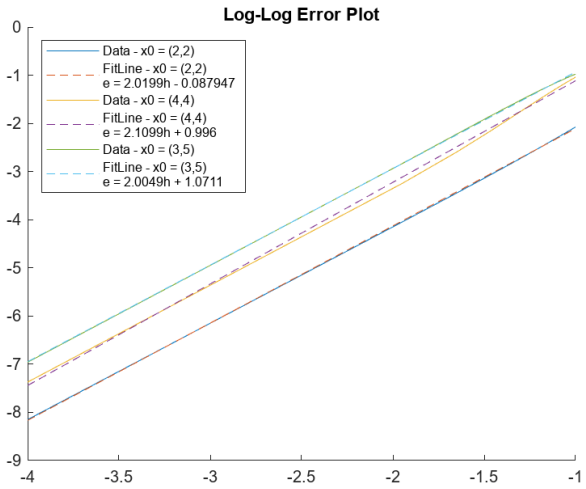

Figure 2 – Log-Log Error Plots

It is also interesting to notice that in both figures, the errors are note exactly the same for different start points. Although not captured in the error scaling, we have discussed this in our theory as we know functions with larger derivatives will have errors. This means that we see several parallel lines for the log-log plots depending on the starting point.

One thing to note here is that these plots actually cannot be used directly to make a decision on the step size we should use to get good results. Instead, I used the phase diagrams to make a decision on step size. As we will see later, all of our results are periodic and therefore form closed loops in the phase diagram. As such, I wanted to pick a step size that resulted in the phase diagrams not showing significant spiral. Larger time steps resulted in more error. This resulted in the graphs decaying into the steady state location over iterations. I wanted a step size that would make this effect noticeable for several periods to get good results. The first time step I found to do this was $\Delta t = 10^{-4}$.

Time Evolution

Now that we have selected a time step, we can go on to look at our system behavior. The first method we will use to do this is a normal plot of population against time. This gives us a good idea of the speed at which these trends happen.

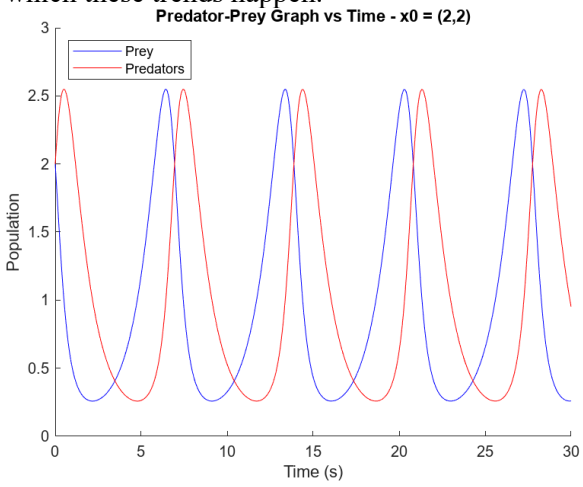

Figure 3a – Lotka-Volterra Evolution starting at (2,2)

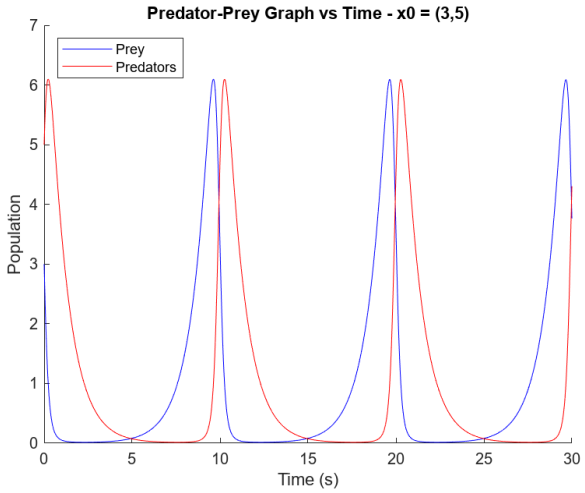

Figure 3b - Lotka-Volterra Evolution starting at (3,5)

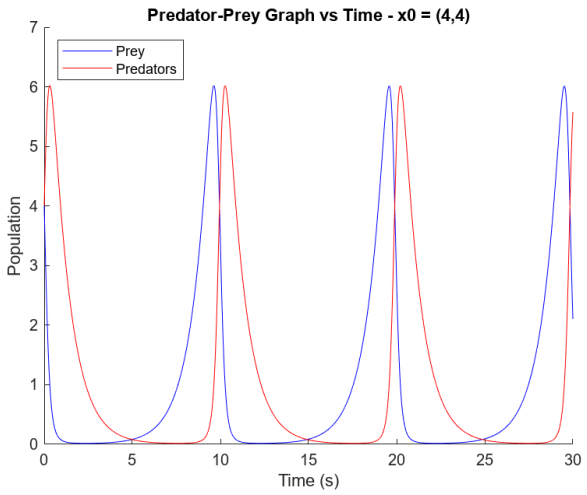

Figure 3c - Lotka-Volterra Evolution starting at (4,4)

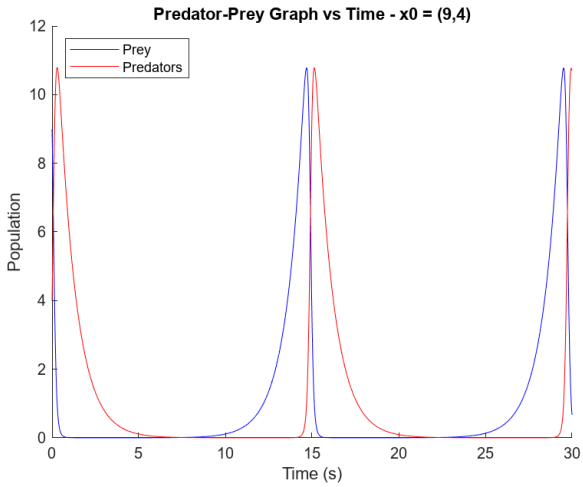

Figure 3d - Lotka-Volterra Evolution starting at (9,4)

As can be seen from these results, different starting positions can give either very different or very similar results. As we increase in distance from the steady state point, we generally see the periods increasing in time. In addition, we see the peaks occur for less time but be relatively higher. This is accompanied with the plots getting closer to and staying near 0 population for more time as we get further form the steady state point. This is clearly seen when we compare Figures 3a-3d. Figure 3a, which is the closest to the steady state point at (1,1) has relatively long peaks and many more periods than the other graphs. We see almost 4 complete cycles in the 30 second time frame we look at. On the other hand, Figures 3b and 3c are much more similar as they are a very similar distance from the origin. Here we see about 3 periods in our 30 second time frame. However, this is where we can see a more clear nonlinearity of our system in the extended time we spend at lower populations. Finally, Figure 3d really shows the nonlinearity exaggerated to the point that we spent the majority of our time near zero population. We also only see 2 periods, less than any other initial condition.

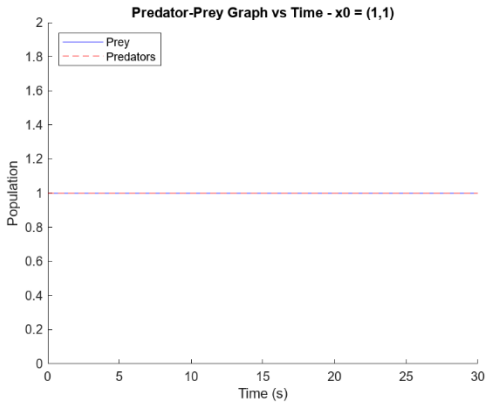

Figure 4 – Steady State Solution

If we now look at Figure 4, we see a line that stays flat. There is no dynamic behavior. This is a result of using the steady state point (1,1) as our initial condition. This means that our time derivative in both variables is zero so we expect no time evolution. Given that we have sufficiently small floating-point error, we would expect this to be the shape of the graph regardless of step size given the unconditional stability of implicit methods.

It is also good to see that the graphs look fairly consistent in terms of period and amplitude. This is a good sign that the step size we have chosen has not resulted in significant error that will skew how our graphs should behave.

Phase Diagram

Although time evolution graphs are great for visualizing data with specific initial conditions on a given time scale, it is not the best way to visualize the dynamics of a system for a large range of initial conditions for arbitrary time scales. In addition, some studies might not even care about time scales and

only care about the interactions between populations, regardless of time. In these cases, and many others, a phase diagram is more useful. Instead of plotting against time, we plot the two populations against one another. For our system, the phase diagram is shown in Figure 5. As we can see, the Prey population is on the x-axis while the predator population is on the y-axis. Each graph represents different initial conditions and the arrows on each graph represent the direction of positive time evolution.

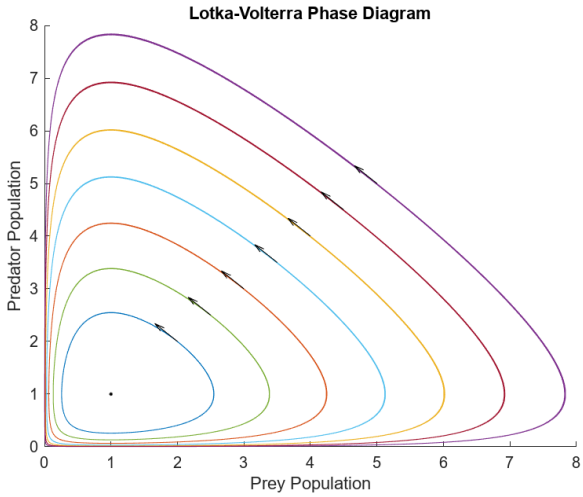

Figure 5 – Lotka-Volterra Phase Diagram

Looking at the phase diagram, we can see some of the interest graph behavior we noticed much more clearly. For example, as we get further from the steady state solution we can see the curves spending more population ranges close to the axis. This is the same as some population being close to zero. In addition, we can even see the basic behavior of the system in these graphs. For example, we can see from the direction of positive time that predator population increase and prey population decreases to some maximum. At this point, there is not enough pretty to keep all the predators alive so the predators start to die off. At some point, there are so few predators that the prey can start reproducing and repopulating. This gives the few prey more food and which allows for the population of predators to grow. This is the cycle that we see for any starting point in this system. If we included the overcrowding term we analyzed above ($p, q \neq 0$) we would expect these curves to spiral down to the steady point and eventually reach it and achieve steady state operations.

Conclusion

We have now took a detailed trip through the process of integrating functions using Implicit Euler Integration. To do this, we first picked and analyzed a Modified Lotka-Volterra system to represent a Predator-Prey interaction. Once we analyzed this system and confirmed physical stability, we then explored numerical methods for integrating our function such that we get time evolving results. This lead us to explore the concept of stability and discovering Implicit Methods, which guarantee numerical stability. Finally, we applied our tools to develop not only time evolving results, but also phase diagrams to show the general system behavior.

Although we have taken a good exploration of these topics, there are still many ideas that we have yet to explore. First, and most obviously, we can explore higher order methods for integration. Currently, we have to take about 300,000 time steps to get thirty seconds of time evolved data with a step size of 0.0001. Although this did not take a crazy amount of time given our system is only two equations, it still took upwards of a minute to run the entire program. If we needed time steps for any reason, this could quickly grow out of hand. Higher order functions would solve this problem without costing us significant accuracy. However, this comes with the added cost of complexity and slightly harder equations to solve. However, we know that using a second order scheme would give us Δt^3 scaling of our error. This means that we could take significantly larger time steps for similar accuracy. In quick calculations, I estimate our time steps could be about 20 times larger than what we have chosen without any loss of accuracy.

Second, one should consider limitations of Newton's method in these applications. In the previous homework, we learned that some systems require continuation schemes to increase the odds of convergence. Currently, we use zeroth order continuation as our time steps our small and our system

fairly easy to work with. With more complex system, one may need to explore continuation schemes if the convergence of Newton's Method with zeroth order convergence beings being the bottleneck for increasing our time steps instead of stability or accuracy.

Finally, I would be interested in how this could be applied to solve transient PDEs such as the heat equation we explored in homework 1 modified to include a transient term. This could likely be done fairly simply using finite differences to approximate the spatial derivatives and using Implicit methods to integrate through time. However, a question that comes to mind is if there is any limits to using Numeric integration to directly find solutions to the 1D heat equation with a forcing function. The form of that problem seems to fit the form necessary for numeric integration. This could also be extended to PDEs by seeing if there is a way to use separation of variables to get a system of 1D equations that can then be regularly integrated to give us solutions for both transient and steady differential equations for any sufficiently nice PDE. Of course, I think the primary limitation here is finding PDEs that, when separation of variables is applied, becomes a system that we can integrate given the tools we currently have developed. This seems to be a difficult limitation to meet, but could become simpler with some exploration.

Regardless of the magnitude of things yet to be explored, this assignment has given us good insight into numeric integration techniques both implicit and explicit. In addition, this has shown us the importance of understanding error limitations and having a both physically and numerically stable system. However, given that we had to implement a Newton's Method loop to complete this problem, we see the potential complexity and time that these problems can take to solve.

Appendix

Step Implicit Euler Function

```
% To run this function, pass in all the a,b,c,d,p,q parameters and an
% initial condition (2x1 vector) x0 and a step size h. This program will
% take one step of Implicit Euler.
function [x1,J,R] = stepImplicitEuler(a,b,c,d,p,q,x0,h,tol)
if(length(x0) ~= 2)
    disp("Initial Condition must be a 2x1 Vector");
    return
end

J = zeros(2);
R = zeros(2,1);

error = 2*tol;
x1 = zeros(2,1);
x1(1) = x0(1);
x1(2) = x0(2);

while error > tol
    % Fill in J matrix
    J(1,1) = 1 - h*(a-b*x1(2))+2*h*p*x1(1);
    J(1,2) = h*b*x1(1);
    J(2,1) = -h * c * x1(2);
    J(2,2) = 1 - h*(c*x1(1)-d) + 2*h*q*x1(2);

    % Fill in R Vector
    R(1) = x1(1) - h*(a-b*x1(2))*x1(1) + h*p*x1(1).^2 - x0(1);
    R(2) = x1(2) - h*(c*x1(1)-d)*x1(2) + h*q*x1(2).^2 - x0(2);

    % Solve J(dx) = -R
```

```
    dx = J\(-R);

    % Find Error
    error = sqrt(sum(dx.^2));

    % Alter x1
    x1 = x1 + dx;
end
end
```

Main Function

```
clc; clear; close all;
% Parameters
a = 1;
b = 1;
c = 1;
d = 1;
p = 0.0;
q = 0.0;
tol = 1e-6;

% Local Error Trends
points = 10;
hset = linspace(.0001,.1,points);
x0Set = [2 2; 4 4; 3 5];
errorSet = zeros(points,2,length(x0Set)); % h, error, Initial Condition

% Get Errors
for j = 1:length(x0Set)
    x0 = x0Set(j,:);
    for i = 1:points
        h = hset(i);
        errorSet(i,1,j) = h;
        x1 = stepImplicitEuler(a,b,c,d,p,q,x0,h,tol);
        h = h/2;
        x1mid = stepImplicitEuler(a,b,c,d,p,q,x0,h,tol);
        x1ref = stepImplicitEuler(a,b,c,d,p,q,x1mid,h,tol);

        e = x1 - x1ref;
        errorSet(i,2,j) = sqrt(sum(e.^2));
    end
end

% Raw Data
figure;
hold on;
title('Error Trends');
xlabel('Step Size (\Delta t)');
ylabel('Relative Error');
for i = 1:length(x0Set)
    plot(errorSet(:,1,i),errorSet(:,2,i),'o-','DisplayName', ...
        "x0 = (" +num2str(x0Set(i,1))+","+num2str(x0Set(i,2))+")");
end
legend('Location','northwest');
%exportgraphics(gcf,"trueError.png");
% Loglog Data, With Equations
figure;
hold on;
fitInfo = zeros(length(x0Set),2); % (Initial Condition), (m,b) for mx=b;
logError = log10(errorSet); % Log h, Log e, Initial Condition
for i = 1:length(x0Set)
```

```
    displayText = "x0 = ("+num2str(x0Set(i,1))+","+num2str(x0Set(i,2))+")";
    f = polyfit(logError(:,1,i),logError(:,2,i),1);
    equationText = "e = "+num2str(f(1))+"h ";
    if f(2) < 0
        equationText = equationText + "- "+num2str(abs(f(2)));
    else
        equationText = equationText + "+ "+num2str(abs(f(2)));
    end
    fitInfo(i,1) = f(1);
    fitInfo(i,2) = f(2);
    plot(logError(:,1,i),logError(:,2,i),'-','DisplayName',"Data - "+displayText);
    plot(logError(:,1,i),f(1)*logError(:,1,i)+f(2),'--','DisplayName', ...
        "FitLine - "+displayText+newline+equationText);
end
lgnd = legend('Location','northwest');
lgnd.FontSize = 8;
title("Log-Log Error Plot");
%exportgraphics(gcf,"logErorr.png");
%x,y plots vs time
x0Set = [2 2; 4 4; 3 5; 9 4; 1 1];
h = .0001; % Picked because of visual loop closure
tend = 30;
results = zeros(round(tend/h),3,length(x0Set)); % (Time Index), (x,y,t), (Initial
Condition)


for j = 1:length(x0Set)
    x0 = x0Set(j,:);
    t = 0;
    titleText = "Predator-Prey Graph vs Time - x0 =
("+num2str(x0(1))+","+num2str(x0(2))+")";
    for i = 1:round(tend/h)
        t = t + h;
        x1 = stepImplicitEuler(a,b,c,d,p,q,x0,h,tol);
        results(i,1,j) = x1(1);
        results(i,2,j) = x1(2);
        results(i,3,j) = t;
        x0 = x1;
    end
    if (x0(1) == 1 && x0(2) == 1)
        figure;
        hold on;
        plot(results(:,3,j),results(:,1,j),'b-','DisplayName','Prey');
        plot(results(:,3,j),results(:,2,j),'r--','DisplayName','Predators');
        legend('Location','northwest');
        xlabel('Time (s)');
        ylabel('Population');
        title(titleText);
    else
        figure;
        hold on;
        plot(results(:,3,j),results(:,1,j),'b-','DisplayName','Prey');
        plot(results(:,3,j),results(:,2,j),'r-','DisplayName','Predators');
        legend('Location','northwest');
        xlabel('Time (s)');
        ylabel('Population');
        title(titleText);
    end
    %exportgraphics(gcf,titleText+".png");
end

% Phase Diagram
x0set = [2 2; 3 3; 4 4; 5 5;2.5 2.5; 3.5 3.5; 4.5 4.5];
```

```
steps = 500000;
xresults = zeros(3, steps+1, length(x0set)); % (x,y,t), (steps), (Initial Conditions)
for i = 1:length(x0set)
    x0curr = x0set(i,:);
    xresults(1:2,1,i) = x0curr;
    xresults(3,1,i) = 0;
    t = 0;
    for j = 1:steps
        t = t + h;
        [x1,J,R] = stepImplicitEuler(a,b,c,d,p,q,x0curr,h,tol);
        xresults(1,j+1,i) = x1(1);
        xresults(2,j+1,i) = x1(2);
        xresults(3,j+1,i) = t;
        x0curr = x1;
    end
end
% Derivative Calculator
dxdt = (a-b.*x0set(:,2)).*x0set(:,1) - p.*x0set(:,1).^2;
dydt = (c.*x0set(:,1)-d).*x0set(:,2) - q.*x0set(:,2).^2;

dt = h;

dx = dxdt * dt;
dy = dydt* dt;

mags = sqrt(dx.^2+dy.^2);

dx = dx./mags;
dy = dy./mags;
figure;
hold on;
for i = 1:length(x0set)
    hold on;
    plot(xresults(1,:,i),xresults(2,:,i),'DisplayName',"x0 = ("+num2str(x0set(i,1))+",
"+num2str(x0set(i,2))+")")
end

xset = x0set(:,1);
yset = x0set(:,2);
quiver(xset,yset,dx,dy,.3,'k');
plot(1,1,'k.');
xlabel('Prey Population');
ylabel('Predator Population');
title('Lotka-Volterra Phase Diagram');
%exportgraphics(gcf,"phaseDiagram.png");
```