ENM-502 HOMEWORK ASSIGNMENT 3

Dhruv Gupta - dgupta99@seas.upenn.edu

April 14 2022

1 Introduction

We aim to study the Lotka-Volterra equations that are used to model predator-prey dynamics. For populations x and y of prey and predator, respectively, we use the equations

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

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

to determine the evolution of the populations through time. The parameters a, b, c, d, p, q determine how the predator and prey species interact and thus play an important role in the problem. The steady states for this set of equations are also found. The numerical integration is done using implicit Euler (first order) and Newton's method is employed to solve the resulting equations. For the purposes of our numerical simulation, we work with the linear problem (p,q set to zero), and study the phase plots obtained for different initial values. We expect to see a centre phase plot around the steady state, since the predator-prey populations should not vanish. The initial value of the populations would also determine the overall behavior of the solution.

2 Problem Setup and Formulation

2.1 Implicit Euler Formulation

The problem is set up with the unknown vector

$$\mathbf{y} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{3}$$

and the differential equations as

$$f = \begin{pmatrix} (a - by)x - px^2 \\ (cx - d)y - qy^2 \end{pmatrix} \tag{4}$$

The implicit (or backward) Euler method is used to trace the solution for this problem. Using interpolating polynomials we derive the implicit Euler method, with step-size h, as

$$\mathbf{y}_{n+1} = \mathbf{y}_n + hf(\mathbf{y}_{n+1}) + O(h^2) \tag{5}$$

Implicit Euler is preferred over its explicit analogue because of its absolute numerical stability. The explicit methods place a bound on the time-step, while the implicit method has absolute stability.

At each instant, to find the solution at the next time-stamp we need a guess for the solution. In my code I use one step of explicit Euler

$$\tilde{\mathbf{y}}_{n+1} = \mathbf{y}_n + hf(\mathbf{y}_n) \tag{6}$$

to obtain a guess for $\mathbf{y}_{n+1}$. This ensures convergence when using Newton's method to solve the non-linear equation. Using (5) we can now set-up an equation to solve with Newton's method. On rearrangement (ignoring the error term) we find

$$R(\mathbf{y}_{n+1}) = \mathbf{y}_{n+1} - \mathbf{y}_n - hf(\mathbf{y}_{n+1}) = 0 \tag{7}$$

which can be set up as

$$\begin{aligned} \delta\mathbf{y}_{n+1}^{i+1} &= (\mathbf{J}^i)^{-1}.R^i \\ \mathbf{y}_{n+1}^{i+1} &= \mathbf{y}_{n+1}^{i} + \delta\mathbf{y}_{n+1}^{i+1} \\ J_{pq} &= \frac{\partial R_p}{\partial \mathbf{y}_q} \end{aligned} \tag{8}$$

for the i^{th} iteration of Newton's method.

2.2 Critical Points

Only one of the critical points is physically possible as the rest start from either a non-real or non-positive population. The critical points for the Lotka-Volterra equations are

$$x = 0,\ y = 0 \tag{9}$$

$$x = 0,\ y = -\frac{d}{q} \tag{10}$$

$$x = -\frac{a}{p},\ y = 0 \tag{11}$$

$$x = -\frac{aq + bd}{pq + bc},\ y = \frac{ac - pd}{pq + bc} \tag{12}$$

For positive parameters a, b, c, d, p, q, the first three solutions are physically impossible, and the fourth solution is possible if $ac > pd$. These conditions are imposed because we can not start with a non-positive population for either species. Therefore, we only consider the fourth critical point, which is $(1, 1)$ for the parameters $1, 1, 1, 1, 0, 0$.

2.3 Choosing Step-Size

An appropriate step-size can be chosen while keeping two important factors in mind; first, the numerical error accumulated as the solution evolves in time, and second, the time for computation. Using a step-size of 1e-3 proves to be difficult as the higher initial value problems fail to converge quickly (likely because the solution increases slowly here). Moving on to 1e-4 and 1e-5, we obtain figures 1 and 2 respectively.

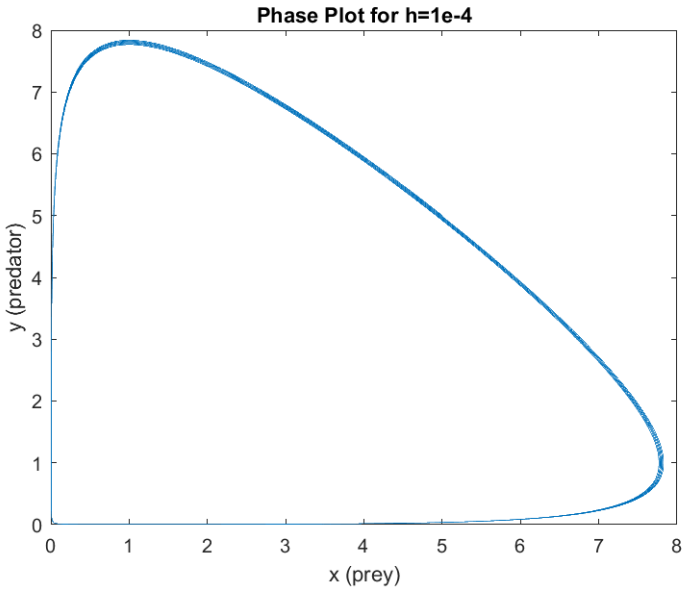

Figure 1: System Integrated For 60 seconds and h = 1e-4

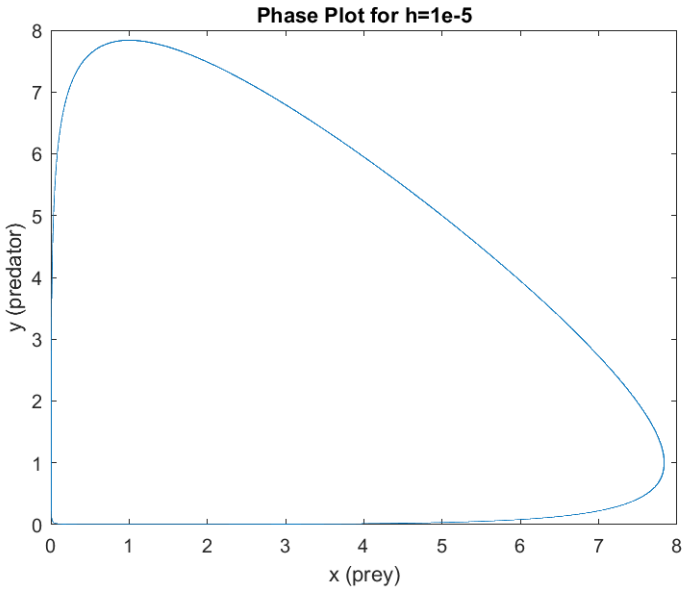

Figure 2: System Integrated For 60 seconds and h = 1e-5

It's clear that the solution for h=1e-4 starts spiralling in towards the steady state after just 4-5 loops. If run for longer, it would diverge from closed-loop solution. Hence it's not a great step-size to choose. The next lower order, h = 1e-5, has a much lower error in comparison. We can visualise

this by comparing the time-evolution of the populations as seen in figure **??**. The peaks in the higher time-step start to decrease, as a result of the numerical error that was discussed earlier.

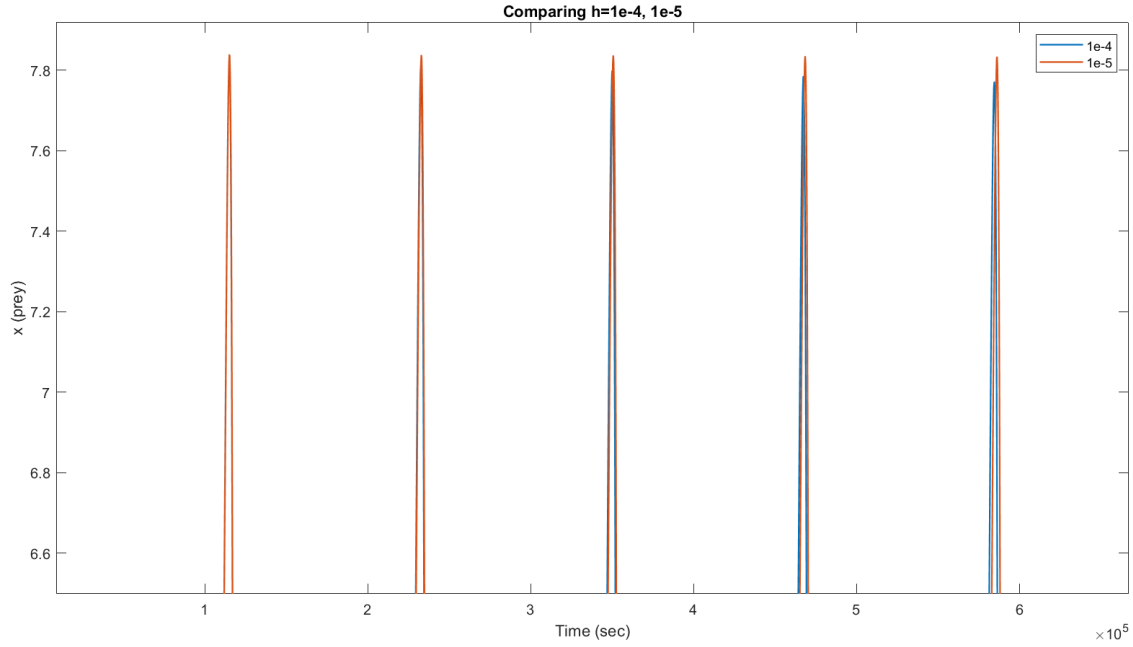

Figure 3: Visualising the numerical error in the higher time-steps

We could potentially get an even more accurate solution going a magnitude lower but it would take more time to integrate. Since h=1e-5 gives us a reasonably accurate solution for the purposes of this analysis, we can stick with it.

3 Results and Discussion

3.1 Error Analysis

Every step of the Euler method carries with it a local truncation error due to the fact that we cut-off the higher-order terms that come from the Taylor-series expansion of the objective function. Equation (5) shows that this error is of the order $O(h^2)$. Since this is the error that we introduce in our numerical solution with respect to the analytical solution, its absolute value can not be obtained unless an extremely small step-size is chosen or if we have the analytical solution already. Since either option is not viable, we use different step-sizes and compare the local error with a minimum step-size (h = 10^{-5} in this case) to verify its scaling.
Figure 4 shows the error-scaling for various solutions and for step-sizes {1e-5, 2e-5...10e-5}. The log-log plot has an average slope of 2.36, which is close to the expected slope of 2 (error due to

ignoring higher-order terms). The log-log set-up is

$$\begin{aligned}\log(err) &\propto \log(O(h^2)) \\ &\propto 2\log(h)\end{aligned}$$

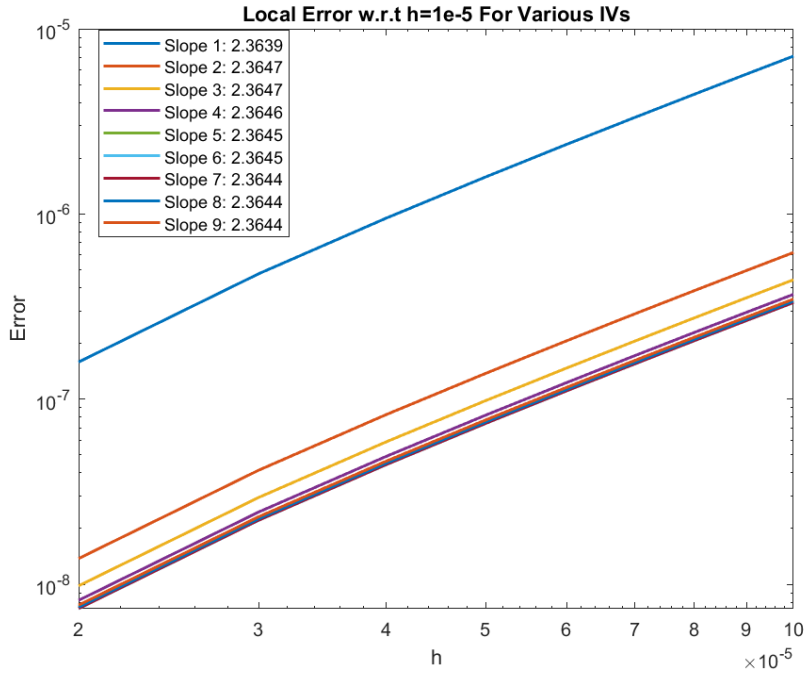

Figure 4: loglog plot of the local error vs step-size h for various initial-value solutions of the form (i,0.01), where $i \in 1, 2..9$

3.2 Time Evolution

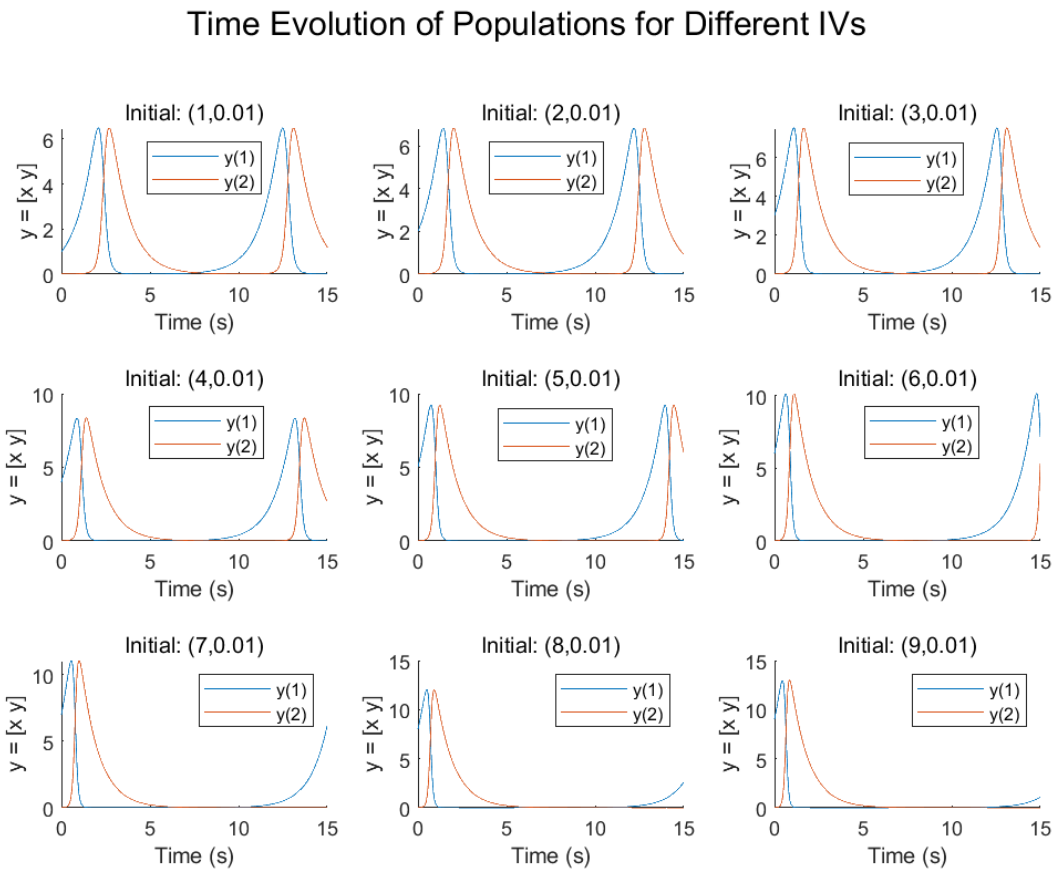

Figure 5: Time Evolution For IVs of the form (i, 0.01) and $h = 10^{-5}$

Figure 5 follows the evolution of prey-predator populations where the prey has a head-start on the predators ([i,0.01]). As we progress through time and the prey population increases due to breeding, the predators have more resources available. Hence, their population starts increasing as well. However, an important feature of all plots is the maximum population for each species. For the first case, we see how once the species population starts falling after reaching a certain bound (due to overpopulation - disease, crowding, etc.), and soon-after this causes a decline in the predator population as well. For a higher initial value the maximum population is reached more quickly and the dynamics takes place earlier. The 'resting' period between peaks is also greater for higher initial values. Also, we see how the

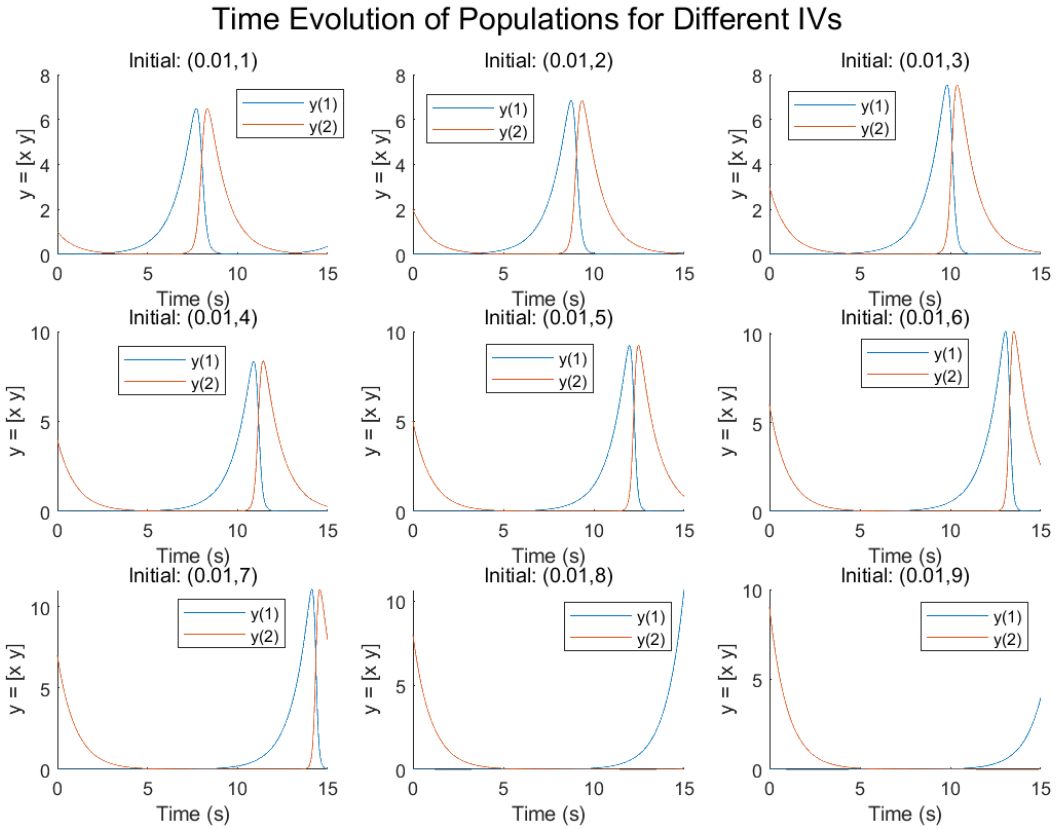

Figure 6: Time Evolution For IVs of the form (0.01, i) and h = 10^{-5}

Figure 6 shows the behavior of the system for an higher initial value of the predator population. This is different from what we saw earlier, as the predator population declines for whatever initial value is chosen (given a low prey population), whereas the prey population would first rise and then fall. Other aspects are similar, such as the 'resting' period, and the rise-and-fall of the predator-prey populations in harmony.

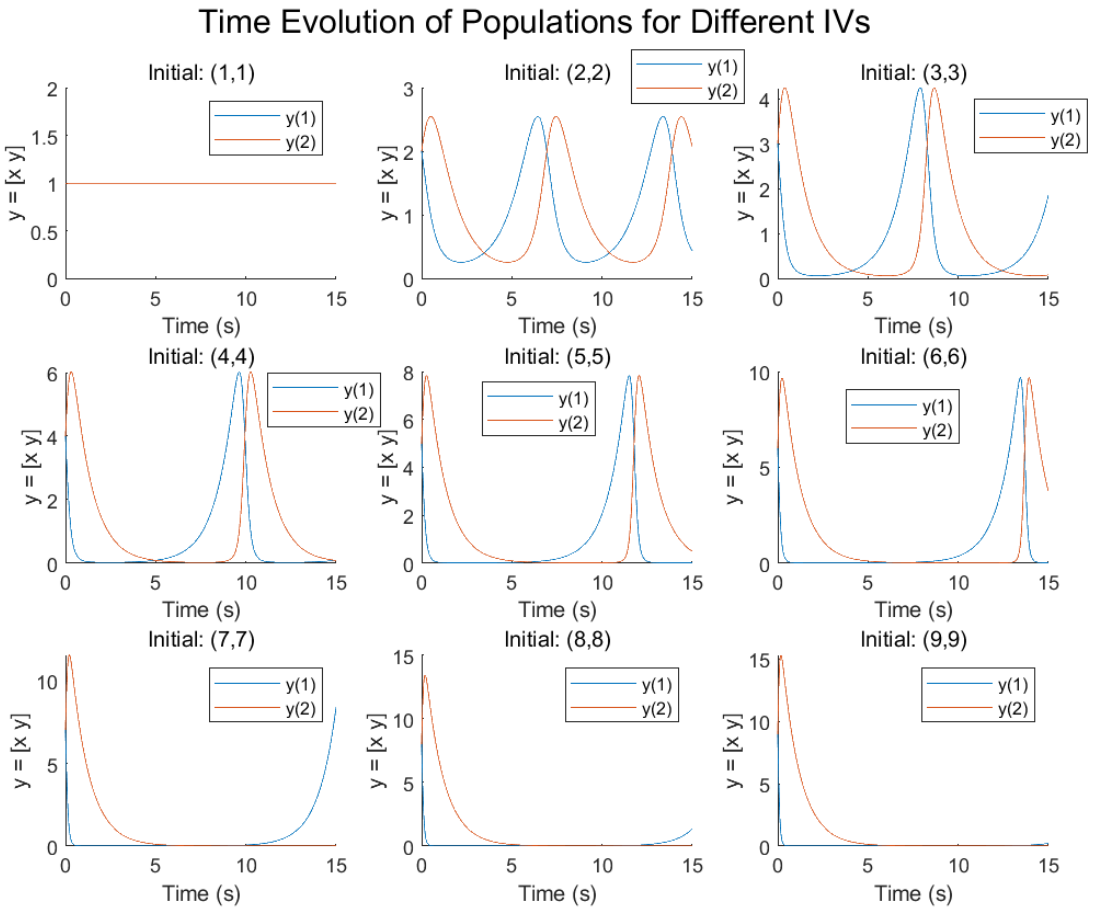

Figure 7: Time Evolution For IVs of the form (i, i) and h = 10^{-5}

Figure 7 shows the time evolution for equal initial values of predator-prey populations. This provides some context for the behavior in figure 6 where the predator population fell initially. Here we see the predator population increase as they have a sufficient supply of prey, and the prey population declines consequently, and very rapidly since they are outnumbered by the predators. Similar oscillatory behavior is also seen here. We also see that starting at (1,1) gives us a flat solution, i.e, the predator-prey populations remain constant. This critical point would represent perfect harmony between the co-dependent species. Mathematically, their populations remain unchanged at any moment in time, but realistically this could mean that for every death in the species a new member is born.

3.3 Phase Plots

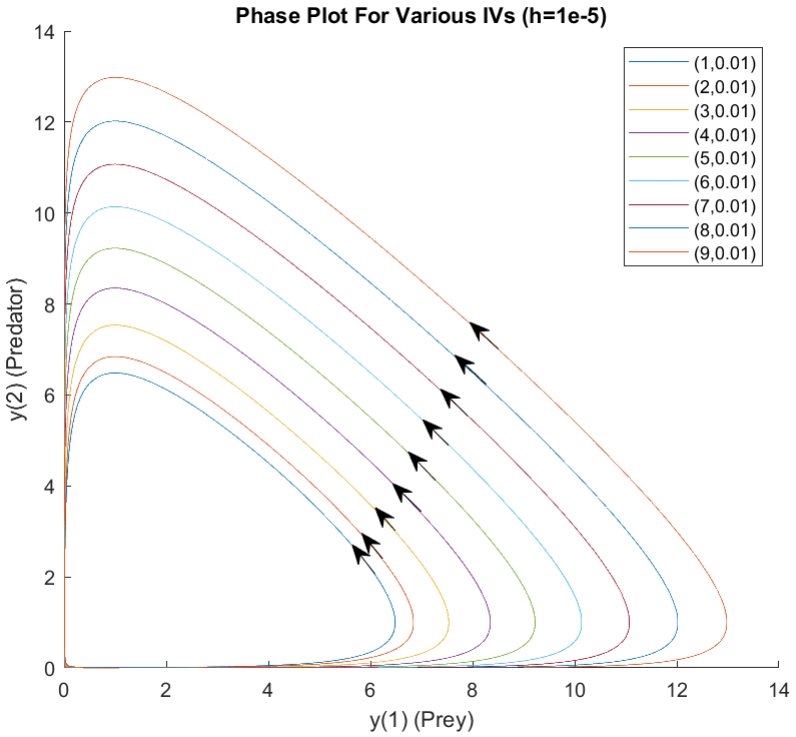

Figure 8: Phase plot shown for various initial values of the form (i,0.01)

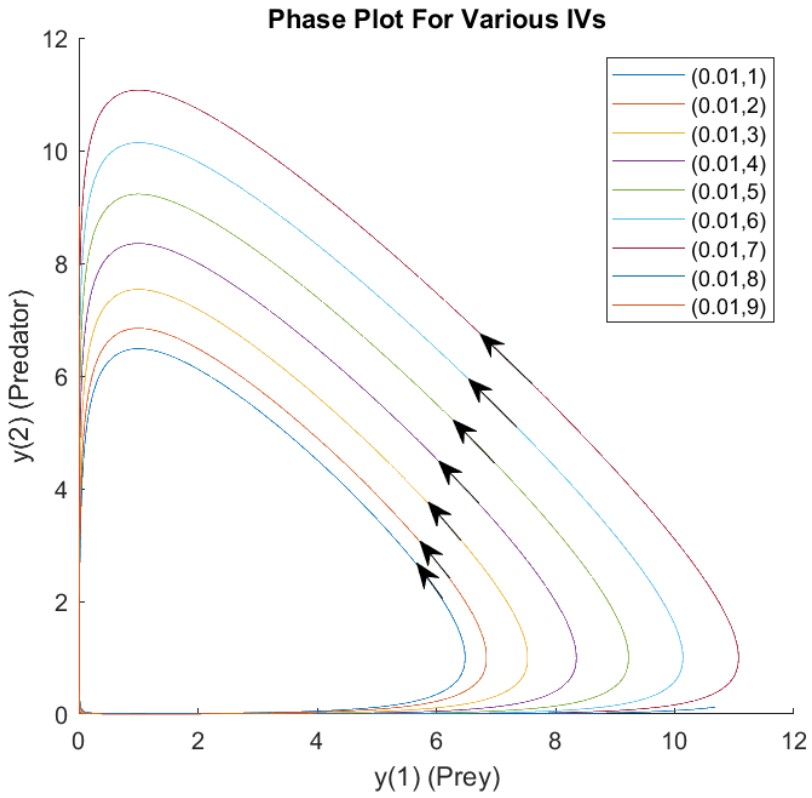

Figure 9: Phase plot shown for various initial values of the form (0.01,i)

Figures 8 and 9 show the phase plots for initial values of the form (i,0.01) and (0.01,i), corresponding to figures 5 and 6 respectively. We can see how there are phases where either species has a population explosion while the other is dormant. Additionally, they are all seemingly centered around the steady state, but it isn't quite visible.

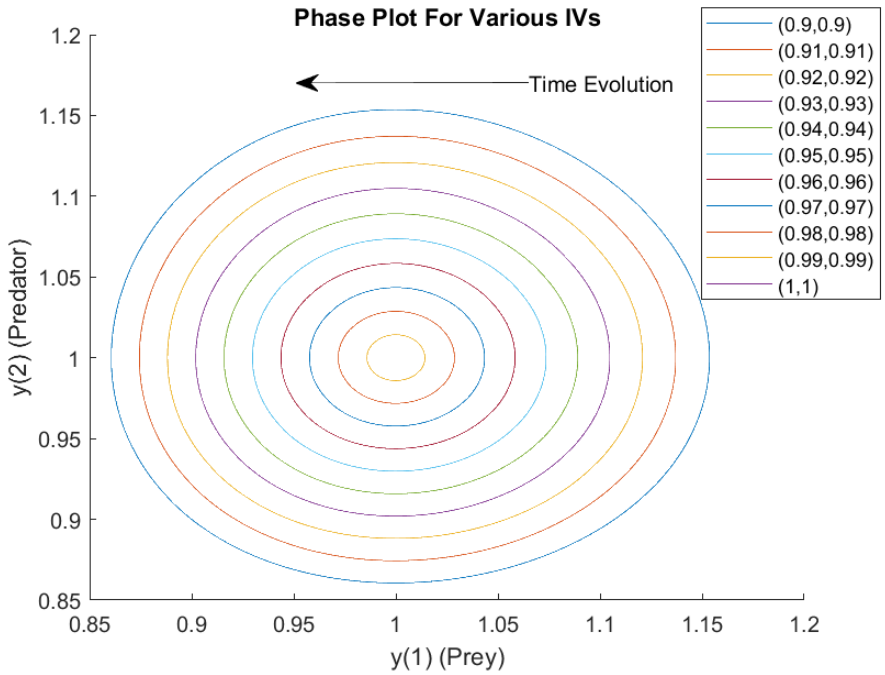

Figure 10: Phase plot shown for initial values close to the steady state. Time evolution is anti-clockwise

The critical point of the system is evident in figure 10, where all solutions near the point (1,1) are centered around it, almost symmetrically (circular) as it seemingly converges to the critical point. There is no 'resting' period as was seen in higher initial value solutions.

4 Conclusions

The implicit Euler method works fairly well in simulating the Lotka-Volterra model, finding oscillatory solutions for the predator-prey populations. While we've ignored the quadratic terms in the system (p,q=0), the problem-solving algorithm that has been developed through Euler and Newton's is general enough to solve that problem as well. Using a better predictor-corrector method will allow us to solve even more complex systems without having to worry about numerical stability (which is assured in implicit Euler regardless). The error-scaling for this method is evident, and higher-order Adams-Bashforth methods can be used to obtain a lower error ($O(h^3)$ or so).

A A Note On Timing Analysis

The time for computing the solution can be a little long (1-2 minutes), if running multiple simulations. This has to do with the fact the time-gradient of the solution is small at the asymptotic parts (when either of x or y approaches zero). At this point one might benefit, computationally, from using an adaptive Euler method. This method would essentially compute the gradient and

take reasonably larger steps if the gradient is too small, and then come back to a normal step if the gradient is within bounds.

B Code

B.1 main.m

```
clear
clc
lvpar = [1,1,1,1,0,0];%a,b,c,d,p,q
h=1e-5;
tol = 1e-5;
miter = 20;
% steady states: (aq+bd; ac-pd)/(pq+bc) : (1,1)
tspan=0:h:15; %15 secs is enough to plot entire evolution for lower initial values
ysoln = zeros(length(tspan),2); %To store all solutions
iv = 1:1:9;
for i = 1:length(iv)
    yinit = [i,0.01];
    y = integrator(yinit,tspan,h,lvpar,miter,tol); %integrator runs the newton's method
    ysoln(:,(i*2)-1) = y(:,1);
    ysoln(:,i*2) = y(:,2);
end
```

B.2 integrator.m

```
function y= integrator(yinit,tspan,h,lvpar,miter,tol)
y=zeros(length(tspan),2);
y(1,:)=yinit; %the first solutin will be the solution converged using yinit
for i=1:(length(tspan)-1)
    yk = newton(lvpar,y(i,:),h,miter,tol);
    y(i,:) = yk';
    y(i+1,:)=yk'; %using y_n to find the next solution
end
end
```

B.3 newton.m

```
function yf = newton(lvpar,y0,h,miter,tol)
J = jacobian(lvpar,y0,h);
R = residual(lvpar,y0,y0,h); %using the previous point as the guess for the next point
dy = J\-R';
yk=y0';
iter = 0;
while (norm(dy)>tol)
    yk = yk + dy;
    f = lv(lvpar,yk);
    yk1 = yk+h*f; %using one step of explicit method
```

```
    % solves convergence issues
    J = jacobian(lvpar,yk,h);
    R = residual(lvpar,yk1,yk,h); %Using explicit euler solution
    dy = J\-R';
    iter = iter+1;
    if(iter>miter)
        disp('Max Iterations Reached')
        %yf = newton(lvpar,y0,h/2,miter,tol);
        %Used to to adaptive step-sizing
        %but the solns aren't evenly spaced
        return;
    end
end
yf = yk+dy;
end
```

B.4 lv.m

```
function f = lv(lvpar,y)
f = zeros(2,1);
f(1) = (lvpar(1)-lvpar(2)*y(2))*y(1) -lvpar(5)*(y(1)^2);
f(2) = (lvpar(3)*y(1)-lvpar(4))*y(2) -lvpar(6)*(y(2)^2);
end
```

B.5 jacobian.m

```
function J = jacobian(lvpar,yk,h)
r11 = 1-h*(lvpar(1) - lvpar(2)*yk(2) - 2*lvpar(5)*yk(1));
r12 = -h*(-lvpar(2)*yk(1));
r21 = -h*(lvpar(3)*yk(2));
r22 = 1-h*(lvpar(3)*yk(1) - lvpar(4) - 2*lvpar(6)*yk(2));
J = [r11 r12; r21 r22];
end
```

B.6 residual.m

```
function R = residual(lvpar,yk1,yk,h)
f = lv(lvpar,yk1);
R(1) = yk1(1)-yk(1)-h*f(1);
R(2) = yk1(2)-yk(2)-h*f(2);
end
```

B.7 erroranalysis.m

```
load('solns\1e5_1.mat'); %loads highest-resolution solution
y1 = ysoln;
err = zeros(9,9);
for i = 2:10 %Loops over the step-sizes
```

```
    load(['solns\1e5_',num2str(i),'.mat']); %loads subsequent step-size solns
    yi = ysoln();
    for j = 1:9 %Loops over all the IV solutions
        %the error in step = h
        le1 =[y1(252001,2*j-1)-y1(252001-i,2*j-1),  y1(252001,2*j)-y1(252001-i,2*j)];
        %error in step = i*h
        lej = [yi((252000/i)+1,2*j-1)-yi((252000/i),2*j-1),  yi((252000/i)+1,2*j)-yi(252000/i,2*j)]
        err(i-1,j) = norm(lej-le1);
    end
end

h=2e-5:1e-5:10e-5;
slope = zeros(9,1);
figure()
for i=1:9
    slope(i) = log(err(9,i)/err(1,i))/log(h(9)/h(1));
    loglog(h,err(:,i),DisplayName=['Slope ',num2str(i),': ',num2str(slope(i))],LineWidth=1.5)
    hold on
end
hold off
ylabel('Error')
xlabel('h')
title('Error w.r.t h=1e-5 For Various IVs')
legend()
```