APPM 2360 - Project 1
Fish Population Modeling

Sanjay Kumar Keshava, Ravin Chowdhury, Grace Cowan

Fall 2020

Introduction

Suppose you are given the task of stocking a lake in a suburban area with fish. You are told that fishing is a popular activity that is enjoyed by the locals, and you are given some information about the amount of fishing that occurs. You have the option of choosing between three species of trout, and an article containing details about each species is made available to you. Your goal is to have a steady supply of trout and prevent over-fishing, so you need to choose carefully. You have no clue where to start, and you are thoroughly confused! Worry not! Using the theory of differential equations, we will study the population dynamics of the different trout species to help you out! By assuming that the fish have no predators other than humans, we will model their growth using a modified version of the logistic equation that takes into account the amount of fishing that occurs. We will focus on using qualitative and numerical techniques to gain insight and gather data. By finding the stability of equilibrium solutions, creating direction fields, and plotting solution curves, we will study the long-term behaviour of the population for each trout species given different initial conditions. Furthermore, using Euler integration, we will obtain some concrete numbers that will be useful for decision making. Finally, based on our analysis we will recommend the species of trout that we think is the most suitable for the job, and we will conclude our study by looking at the weaknesses of our model.

1 The Logistic Equation

We will study the population of fish in the lake with a logistic growth model modified to show the effects of harvesting fish. When there is no harvesting, our model takes the form of the standard logistic equation (1).

$$\frac{dy}{dt} = r(1 - \frac{y}{L})y \tag{1}$$

Here, y represents the population of the fish in *100s of fish* as a function of t, the time in in *days*. Using dimensional analysis, we found that $\frac{dy}{dt}$ has units of *100s of fish per day*, the initial growth rate r has units of $\frac{1}{days}$ and L which is the carrying capacity has units of *100s of fish*.

We observe that $y = L$ and $y = 0$ are equilibrium solutions to (1) because both cause $\frac{dy}{dt}$ to equal 0. An equilibrium solution is one for which the rate of change of the dependent variable with respect to the independent variable is zero. This implies that an equilibrium solution is given by a constant value of the dependent variable and corresponding the solution curve is represented as a horizontal line. Mathematically speaking, if we look at a typical ODE with the form of equation (2), then an equilibrium solution is a function of y that satisfies both (2) and (3). This implies that an equilibrium solution must be of the form given by (4) where c is a constant.

$$\frac{dy}{dt} = f(t, y) \tag{2}$$

$$\frac{dy}{dt} = 0 \tag{3}$$

$$y = c \tag{4}$$

In the case of an autonomous ODE like the one we will be investigating, the rate of change of y with respect to t is no longer $f(t, y)$ but it is just $f(y)$. If we plot the curve $f(y)$ against y, we can find the equilibrium solutions by looking for the values of y at which $f(y)$ is zero (this occurs where the curve touches the horizontal axis).

We now go back to equation (1) and solve for y to get the general solution.[1] By assuming that the initial number of fish (in hundreds) present in the lake was y_0, we can extract the particular solution (5).

$$y(t) = \frac{y_0 e^{rt}}{1 + \frac{y_0}{L}(e^{rt} - 1)} \tag{5}$$

[1] See Appendix A for the full derivation

With harvesting taken into account, our model turns into (6). The extra term is the harvesting function $h(y)$ which will be discussed in detail in the next section.

$$\frac{dy}{dt} = r(1 - \frac{y}{L})y - \frac{py^2}{q + y^2} \tag{6}$$

After moving all the terms containing y to the left and separating variables in (6), we obtain the following integrals (7).[2] Since we need to integrate a rational function, we propose that the method of partial fractions can be employed to obtain an analytical solution.

$$\int \frac{L(q + y^2)dy}{(L - y)ry(q + y^2) - pq^2L} = \int dt \tag{7}$$

However, in our analysis, we will employ only qualitative and numerical techniques to extract the information we require.

2 The Harvesting Function

We now explore the harvesting function (8). $h(y)$ has units of *100s of fish per day* and it tells us how the number of fish harvested each day depends on the number of fish that are present.

$$h(y) = \frac{py^2}{q + y^2} \tag{8}$$

The constant p can thought to be proportional to the number of people fishing, and we can interpret q (which is related to the water temperature) as a constant that affects the probability of catching a single fish in unit time. As p grows larger (assuming the other quantities are fixed), it means more people go fishing, so more are caught each day, and consequently, *h(y)* becomes larger. Conversely, as q grows larger (assuming the other quantities are fixed), it means the probability of catching a single fish in unit time is decreasing, so fewer fish are caught per day, and *h(y)* decreases. To illustrate these features and study the behaviour of the harvesting function when y grows very large, we plotted graphs of *h(y)* against y for different values of p and q using MATLAB.[3]

[2] See Appendix B for the details
[3] See Appendix C for the associated code

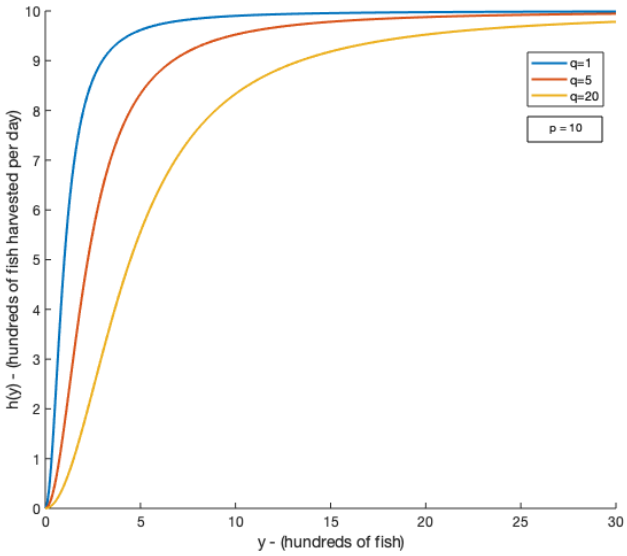

Figure 1: Graph of the harvesting function $h(y)$ versus y (for $p = 10$).

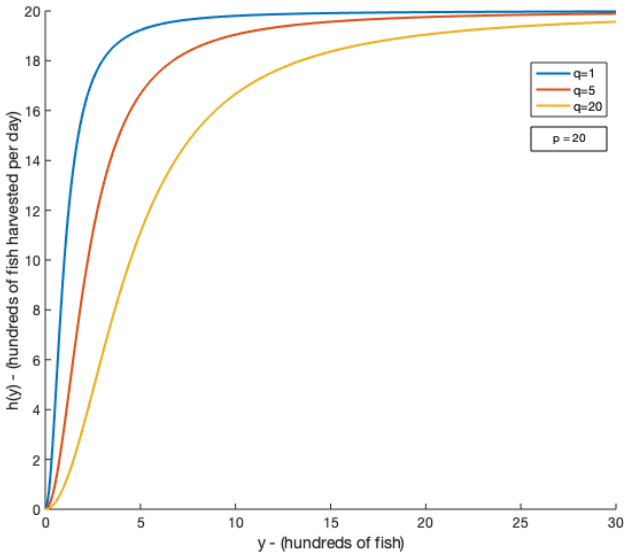

Figure 2: Graph of the harvesting function $h(y)$ versus y (for $p = 20$).

We notice that when y grows infinitely large, $h(y)$ asymptotically approaches the constant p irrespective of how big q is. Even if the probability of catching a single fish in unit time is low this is no longer a limiting factor – since there are so many more fish now, a person will get lucky pretty easily! Furthermore, in this regime, the only limiting factor is the maximum rate at which a person can fish (we assume this is a finite constant), and this is why $h(y)$ has a maximum value of p beyond which it cannot rise. Finally, when the number of fish in the lake approaches zero, locals can harvest fewer and fewer fish, and therefore the harvesting function must approach zero as well.

3 Qualitative and Numerical Analysis

Using the insight gained from the previous sections, we will now try to study the population dynamics of each trout species in detail. After doing some research, we found that all three species have the same intrinsic growth rate $r = 0.65$, but different carrying capacities. In particular, we will look at rainbow trout ($L = 5.4$), brown trout ($L = 8.1$), and brook trout ($L = 16.3$). Furthermore, by talking to the local people we learned that $p = 1.2$ and $q = 1$. We begin by plotting $f(y)$ versus y for each species of trout to estimate the equilibrium solutions graphically. MATLAB was used to generate the graphs.[4]

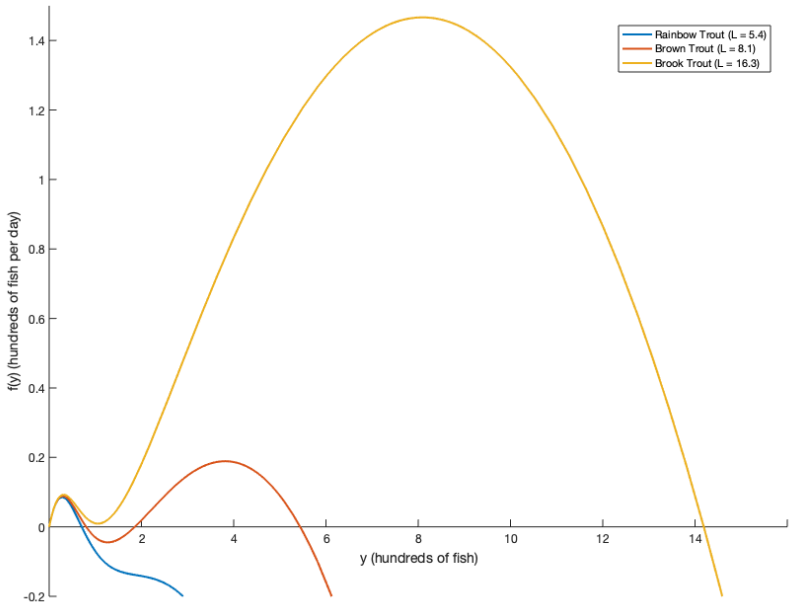

Figure 3: Graph of $f(y)$ versus y for the three trout species (zoomed in).

[4]See Appendix C for the associated code

We can find the equilibrium solutions for each trout species by looking for the values of y at which *f(y)* is zero (this occurs where the curves touch the horizontal axis). From figure (3) we can see that there are two equilibrium solutions for the rainbow trout and the brook trout, and four for the brown trout. To verify that we did not missed any equilibrium solutions, we used MATLAB to plot the same graph as above with a wider range on the vertical and horizontal axes.[5] This is shown in figure (4) below.

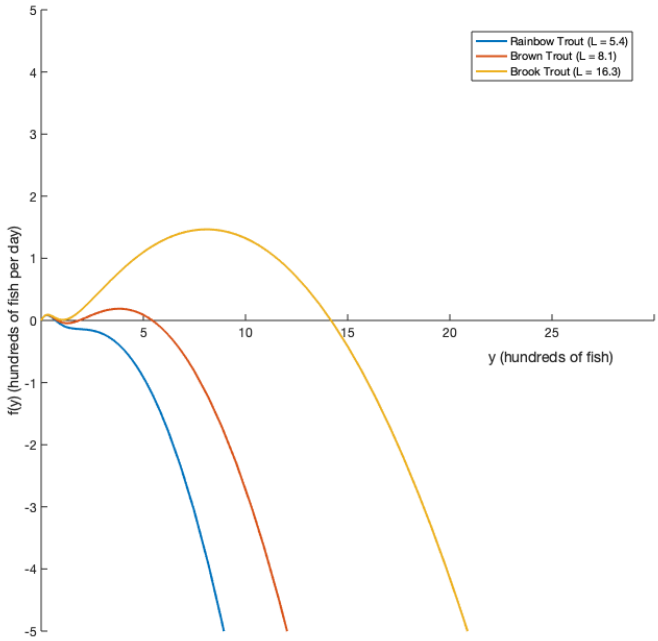

Figure 4: Graph of *f(y)* versus y for the three trout species (zoomed out).

We notice that after a point, *f(y)* is negative and continually decreasing for all three graphs. This is because *f(y)* is dominated by the harvesting function *h(y)* for large values of y. This gives us confidence that there are in fact no other equilibrium solutions.

Using our plots, we graphically estimated the equilibrium solutions. Furthermore, we used the *fzero* function on MATLAB to find the solutions numerically with a higher level of accuracy. We found that there was strong agreement between our estimates and the values obtained through *fzero*. We know that if $f(y) < 0$ it means that $\frac{dy}{dt} < 0$ and the population is decreasing. On the other hand if $f(y) > 0$ it means that $\frac{dy}{dt} > 0$ and the population is increasing. Finally $f(y) = 0$ corresponds to an equilibrium solution where the population

[5] See Appendix C for the associated code

of fish is not changing in time. By noting down the sign of $f(y)$ on either side of each equilibrium solution, we were able to determine the stability of each solution. Table (5) that is given below summarizes the results. We notice that different values of L result in different numbers of equilibrium solutions. To find the bifurcation values (values of L at which the number of equilibrium solutions change) we used Desmos to plot *f(y)* against y and varied L by using a slider. By graphical estimation, we found that there were two bifurcation points – one at $L = 6.72$ (number of equilibrium solutions changes from 2 to 4), and the other at $L = 13.56$ (number of equilibrium solutions changes from 4 to 2).

Trout species	Equilibrium solutions - y values (*hundreds of fish*)		
	Graphical estimate (1dp)	Exact value (6dp)	Stability
Rainbow trout	0.0	0.000000	Unstable
	0.7	0.705046	Stable
Brown Trout	0.0	0.000000	Unstable
	0.8	0.801820	Stable
	1.9	1.856372	Unstable
	5.4	5.441808	Stable
Brook Trout	0.0	0.000000	Unstable
	14.2	14.189777	Stable

Figure 5: Summary of the equilibrium solutions and their stabilities

Next, to further investigate the stability of the different equilibrium solutions and look at the long-term behaviour we plot direction fields for each trout species along with solution curves corresponding to a variety of different initial conditions. The different colours (black and blue) were included merely for contrast. The plots below were generated on Python.[6]

[6]See Appendix D for the details

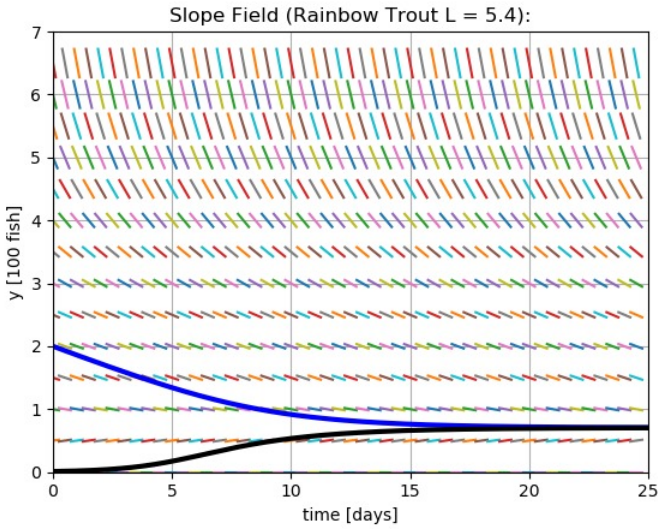

Figure 6: Slope field for rainbow trout with different initial conditions.

We begin with the rainbow trout. For our first initial condition, we started with just two fish. We notice that the solution curve that corresponds to this situation (the black line) moves away from the equilibrium solution at $y = 0$ and towards the one at $y = 0.7$. Moving on, by starting with the initial condition $y = 2$ (the blue line) we see that the population decreases and asymptotically approaches $y = 0.7$. All these observations lead to the conclusion that $y = 0.7$ is a stable equilibrium and $y = 0$ is unstable. It should be noted that in our analysis we do not investigate initial conditions corresponding to starting with a negative number of fish because this has no physical meaning. However, by looking at the sign of $f(y)$ when y is lower than zero, we were able to mathematically classify $y = 0$ as an unstable equilibrium solution (this applies to all the three species of trout since they all have $y = 0$ as an equilibrium solution). On the next page, we study the slope field for the brown trout.

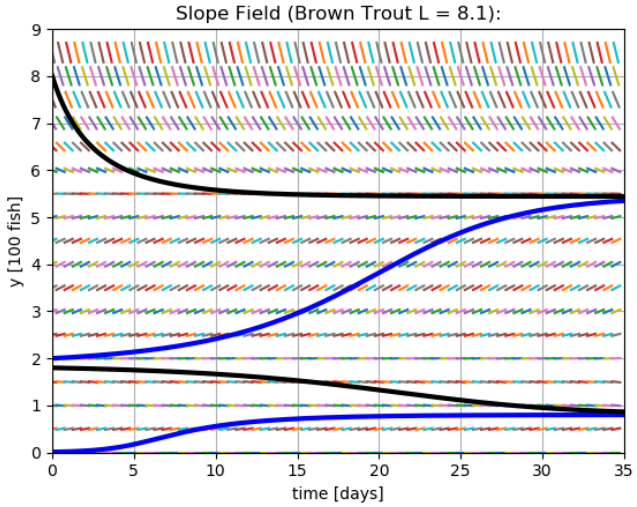

Figure 7: Slope field for brown brout with different initial conditions.

The direction field of the brown trout in figure (7) shows multiple equilibrium solutions. In all, there are four: $y = 0$, $y = 0.8$, $y = 1.9$ and $y = 5.4$ as predicted by our earlier analysis that is summarized on table (5). Looking at the initial conditions that are just above and below (apart from $y = 0$) each equilibrium solution, we see that $y = 0.8$ and $y = 5.4$ are stable, and all solution curves move towards one of the two. As for the other equilibrium solutions, we notice that the solution curves starting near $y = 1.9$ and $y = 0$ tend to move away indicating that they are unstable. As mentioned earlier, we are not considering initial conditions corresponding to cases where $y < 0$. Next we will conclude our study by looking at the direction field for the brook trout.

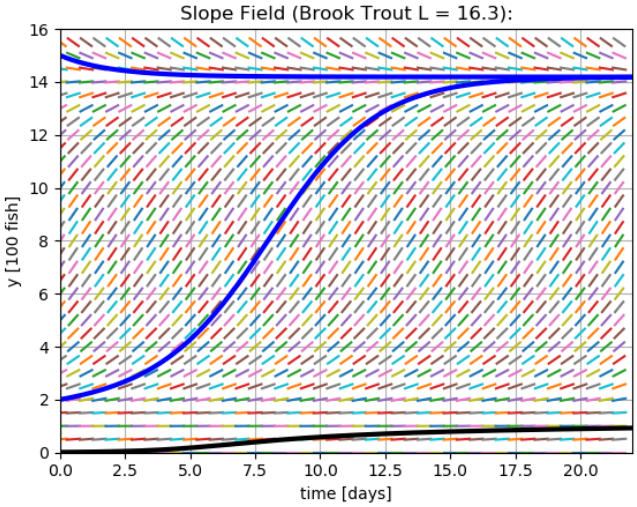

Figure 8: Slope field for brook trout with different initial conditions (short time range).

The direction field for brook trout is especially interesting. Looking at the short time range graph in figure (8), we notice it is similar to that of the other trout species in that $y = 0$ is an unstable equilibrium, and starting at just over 0 fish (the black line) gives us a solution curve that appears to approach $y = 0.5$. It may appear that this, along with $y = 14.2$ are both equilibrium solutions, but figure (9) below shows us that eventually the black line goes over and joins the stable equilibrium at $y = 14.2$. Although it initially looked like there was a new equilibrium solution that we may have missed, looking at the behaviour over a very long time period led to results that are consistent with our previous analysis that is summarized on table (5).

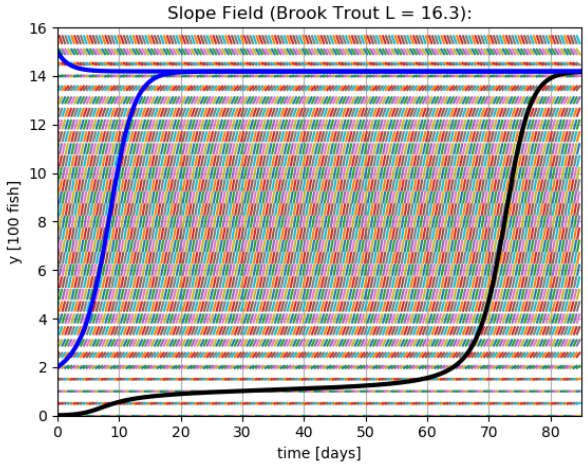

Figure 9: Slope field for brook trout with different initial conditions (long time range).

Overall from the direction fields, we see that different values of L lead to strikingly different population dynamics. Furthermore, for each value of L the initial conditions along with their position relative to the different equilibrium solutions dictate the future of the fish in a very specific manner.

Next, we employ numerical integration (Euler's method) to study the population of each trout species over a long time period. We begin with 100 fish (this means $y = 1$) and study the population over 60 days with a time step (h) of 0.01 days. Figure (10) on the next page shows our results. We used MATLAB to implement Euler's method and plot the graph.[7] After 60 days, we found that the values of y were $y = 0.705046$ (71 fish) for rainbow trout, $y = 0.801822$ (80 fish) for brown trout, and $y = 14.185897$ (1419 fish) for brook trout. These results closely match what we expected from out direction fields and previously determined equilibrium solutions. From our results, after 60 days the number of rainbow and brown trout have only decreased slightly from 100 fish and eventually stabilize near 71 and 80 respectively. The population of brook trout, on the other hand, seems to grow slowly for a while before exploding suddenly, ultimately stabilizing at around 1419 fish. This makes them much more appealing to populate the lake with. A large stable population means that the locals will routinely be able to fish larger quantities without fear of damaging the population.

[7]See Appendix C for the details

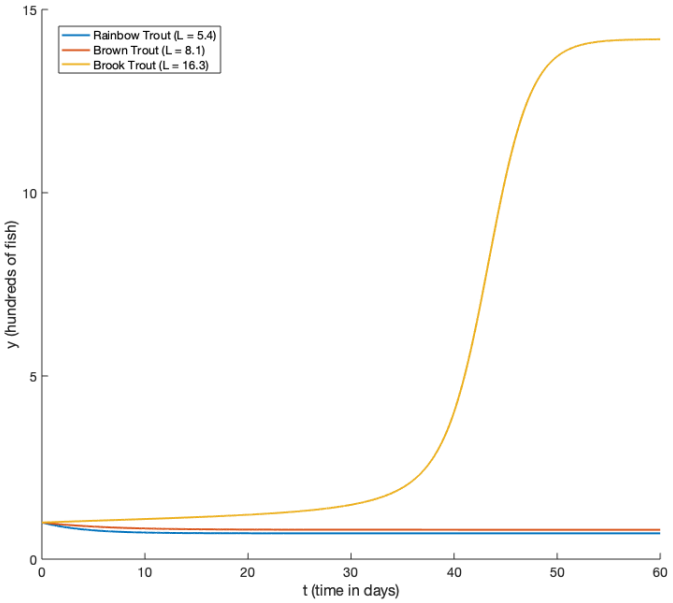

Figure 10: Results from Euler's Method

The mathematical reasons for the success of the brook trout are twofold. The rainbow trout has a very small carrying capacity of 540 fish without harvesting and brown trout has a slightly larger one at 810 fish. These numbers mean that an even lower stable population will be present after introducing harvesting. They both fall short of the carrying capacity of brook trout which is 1630 fish without harvesting. Brown trout, moreover, has a very complex direction field with multiple equilibrium solutions, and if we were to introduce a larger quantity to try and get a larger stable population, we would run into many equilibrium solutions which are very restrictive. The direction field of the brook trout is more forgiving and as long as the fish have do not die out, we are guaranteed a stable population of around 1419 fish even though it might take some time to reach that number.

So far, we have been assuming that a greater number of final fish is our final goal. This is never the only factor taken into consideration. Suppose instead we were interested in the greatest mass of fish in the lake. In this case, we would have to take into account the weights of the different trouts. Rainbow

and brown trouts are larger, at 12 and 20 kg each respectively, while brook trouts weight only 4 kg (which explains the large carrying capacity). The stable mass of fish in the lake in each case would be 840 kg, 1600 kg and 5680 kg for rainbow, brown and brook trout respectively showing that brook trout still wins out. Next, suppose we wanted to limit the number of trout in the lake to leave capacity for other wildlife such as snapping turtles. The brook trout 'explosion' might come at the cost of all other wildlife and may need to be avoided. To conclude, with the limited information we have been given, our simple model suggests that populating the lake with brook trout would be the best move.

4 Weaknesses of Our Model

This model works if and only if the variety of assumptions we have made hold true. As we know from nature, humans are not the only predator of any kind of fish, and local animals such as eagles and bear will greatly affect how the population of fish fluctuates. In order to account for this extra piece, it would be best to insert a function akin to $h(y)$ where the function of the fish being eaten by animals in the wild is also affected by how many fish are available. In the harvesting function $h(y)$, the constant q is a set parameter that would be unlikely to reflect accurately in long-term trends. It seems that q is highly dependent on the season and will undergo cyclic changes. As a result, we may wish to write q as some periodic function of time. After doing some quick research it appears that rainbow, brown, and brook trout all have different intelligence levels. The brown trout is considered the 'smartest' trout and after enough fish of a brown trout population are caught, they are harder to fish as they start recognizing the bait as a potential threat. Once again we might wish to look at q and perhaps assign a different value to each species of trout. We could go on and on with more suggestions as to how to improve our model, but after a point it might become impractical to actually implement everything, and it may also no longer be relevant to the question that we are trying to answer.

5 Conclusions

Using our trout population model which is described by a differential equation, we determined several important things that helped us recommend the trout species to populate the lake with. This was done using qualitative and numerical techniques. We explored three species in specific – rainbow trout, brown trout and brook trout. All three species had different carrying capacities in the absence of fishing. When we considered the effects of fishing, the remaining parameters used were kept the same for all three species (the values of p, q and r). By looking at the individual models, we found the equilibrium solutions for each fish species. Each species had an unstable equilibrium at $y = 0$. The rainbow trout and brook trout each had a stable equilibrium, at $y = 0.7$ and $y = 14.2$ respectively. The brown trout had the most complex behaviour, having one more unstable solution (in addition to $y = 0$) at $y = 1.9$ and two stable solutions at $y = 0.8$ and $y = 5.4$. Since the only variable changing the behaviour of our model for different trout species was the carrying capacity L, we were able to scroll through different values for L and see when the number of equilibrium solutions of our differential equation changed (we call these bifurcation values). The bifurcation values were at $L = 6.72$ and $L = 13.56$ (L in *100s of fish*). Next, we used Euler's method by starting with 100 fish for each model and predicted how many fish would be in the lake after 60 days. The rainbow and brown trout had their populations decrease to 71 and 80 fish respectively, but the population of brook trout skyrocketed to 1419 fish and stabilised. Using our results we determined that it would be most beneficial for the locals to populate the lake with brook trout. They would be able to harvest the most fish, and the brook trout having only one stable equilibrium solution much higher than any other species would ensure their sustainability on fishing.

6 Appendices

A Fish Growth Without Harvesting

$$\frac{dy}{dt} = r(1 - \frac{y}{L})y$$

$$\frac{dy}{y(1 - \frac{y}{L})} = rdt \quad \text{..separating variables}$$

$$\int \frac{dy}{y(1 - \frac{y}{L})} = \int rdt \quad \text{..integrating both sides}$$

$$\int (\frac{1}{y} - \frac{1}{y - L})dy = \int rdt \quad \text{..splitting into partial fractions}$$

$$\ln y - \ln (y - L) = rt + C \quad \text{..completing the integration}$$

$$y(t) = \frac{y_0 e^{rt}}{1 + \frac{y_0}{L}(e^{rt} - 1)} \quad \text{..making y the subject of the formula}$$

$y(0) = y_0$ was taken as the initial condition.

B Fish Growth With Harvesting

$$\frac{dy}{dt} = r(1 - \frac{y}{L})y - \frac{py^2}{q + y^2}$$

$$\frac{dy}{r(1 - \frac{y}{L})y - \frac{py^2}{q+y^2}} = dt \quad \text{..separating variables}$$

$$\frac{L(q + y^2)dy}{(L - y)ry(q + y^2) - pq^2L} = dt \quad \text{..simplifying LHS}$$

$$\int \frac{L(q + y^2)dy}{(L - y)ry(q + y^2) - pq^2L} = \int dt \quad \text{..starting integration}$$

C MATLAB Code

The following code was written on MATLAB and used to generate plots that have been used in the write-up.

DIFFEQ PROJECT 1

Table of Contents

Harvesting function plots %%

```
% format long;

% y = 0:0.1:30;
% figure (1);
% set(gca, 'FontSize', 12)
% p = 10;
% hold on;
% for q = [1, 5, 20]
%     y_h = (p.*(y.^2))./(q + (y.^2));
%     plot (y, y_h, 'LineWidth', 2)
% end
% % title('Harvesting function verus population - (p = 10)',
 'FontSize', 14)
% ylabel('h(y) - (hundreds of fish harvested per day)','FontSize', 14)
% xlabel('y - (hundreds of fish)','FontSize', 14)
% legend('q=1', 'q=5', 'q=20')
% hold off;
%
%
% figure (2);
% set(gca, 'FontSize', 12)
% p = 20;
% hold on;
% for q = [1, 5, 20]
%     y_h = (p.*(y.^2))./(q + (y.^2));
%     plot (y, y_h, 'LineWidth', 2)
% end
% % title('Harvesting function verus population - (p = 20)',
 'FontSize', 14)
% ylabel('h(y) - (hundreds of fish harvested per day)','FontSize', 14)
% xlabel('y - (hundreds of fish)','FontSize', 14)
% legend('q=1', 'q=5', 'q=20')
% hold off;
```

Plots of f(y) vs y and calculation of equilibrium solutions using fzero %%

```
% format long;
```

```
% L = 5.4 -- 2 eq sols -- y = 0 -- 0.69 < y < 0.72
% L = 8.1 -- 4 eq sols -- y = 0 -- 0.79 < y < 0.82 -- 1.84 < y < 1.87
 -- 5.43 < y < 5.46
% L = 16.3 -- 2 eq sols -- y = 0 -- 14.17 < y < 14.21

% we can use fplot to plot our graphs
% we can use fzero to get accurate values for the equilibrium
 solutions.

% p = 1.2;
% q = 1;
% r = 0.65;
% figure(5);
% set(gca, 'FontSize', 12)
% hold on;
%
% for L = 5.4
%     rootFunction = @(y) (r.*y.*(1-(y./L))) - ((p.*(y.^2))./(q +
 (y.^2)));
%     fplot(rootFunction,[0 16],'LineWidth', 2);
%     xlim([0 16]);
%     ylim([-0.2 1.5]);
%     ax = gca;
%     ax.XAxisLocation = 'origin';
%     ax.YAxisLocation = 'origin';
% end
%
% % root finding 5.4
% b1_54 = [0.69, 0.72];
% yr1_54 = round (fzero(rootFunction,b1_54),6)
% fyr1_54 = rootFunction(yr1_54);
% % plot (yr1_54, fyr1_54, 'om');
%
%
%
% for L = 8.1
%     rootFunction = @(y) (r.*y.*(1-(y./L))) - ((p.*(y.^2))./(q +
 (y.^2)));
%     fplot(rootFunction,[0 16],'LineWidth', 2);
%     xlim([0 16]);
%     ylim([-0.2 1.5]);
%     ax = gca;
%     ax.XAxisLocation = 'origin';
%     ax.YAxisLocation = 'origin';
% end
%
% % root finding 8.1
% b1_81 = [0.79, 0.82];
% yr1_81 = round (fzero(rootFunction,b1_81),6)
% fyr1_81 = rootFunction(yr1_81);
% % plot (yr1_81, fyr1_81, 'om');
%
% b2_81 = [1.84, 1.87];
```

```
% yr2_81 = round (fzero(rootFunction,b2_81),6)
% fyr2_81 = rootFunction(yr2_81);
% % plot (yr2_81, fyr2_81, 'om');
%
% b3_81 = [5.43, 5.46];
% yr3_81 = round (fzero(rootFunction,b3_81),6)
% fyr3_81 = rootFunction(yr3_81);
% % plot (yr3_81, fyr3_81, 'om');
%
%
% for L = 16.3
%     rootFunction = @(y) (r.*y.*(1-(y./L))) - ((p.*(y.^2))./(q +
 (y.^2)));
%     fplot(rootFunction,[0 16],'LineWidth', 2);
%     xlim([0 16]);
%     ylim([-0.2 1.5]);
%     ax = gca;
%     ax.XAxisLocation = 'origin';
%     ax.YAxisLocation = 'origin';
% end
%
%
% % root finding 16.3
% b1_163 = [14.17, 14.21];
% yr1_163 = round(fzero(rootFunction,b1_163),6)
% fyr1_163 = rootFunction(yr1_163);
% % plot (yr1_163, fyr1_163, 'om');
%
% % plotting common root for all
% % plot(0,0,'om');
%
% hold off;
% % title('Graphs of the rate of change of population vs population
 for 3 species of trout','FontSize', 14);
% xlabel('y (hundreds of fish)','FontSize', 14);
% ylabel('f(y) (hundreds of fish per day)','FontSize', 14);
% legend('Rainbow Trout (L = 5.4)', 'Brown Trout (L = 8.1)', 'Brook
 Trout (L = 16.3)');
%
%
% figure(6);
% set(gca, 'FontSize', 12)
% hold on;
%
% for L = [5.4, 8.1, 16.3]
%     rootFunction = @(y) (r.*y.*(1-(y./L))) - ((p.*(y.^2))./(q +
 (y.^2)));
%     fplot(rootFunction,[0 50],'LineWidth', 2);
%     xlim([0 30]);
%     ylim([-5 5]);
%     ax = gca;
%     ax.XAxisLocation = 'origin';
%     ax.YAxisLocation = 'origin';
% end
```

```
%
% xlabel('y (hundreds of fish)','FontSize', 14);
% ylabel('f(y) (hundreds of fish per day)','FontSize', 14);
% legend('Rainbow Trout (L = 5.4)', 'Brown Trout (L = 8.1)', 'Brook
 Trout (L = 16.3)');
```

Finding the bifurcation points %%

```
% format long;

%  We used trial and error by varying L, plotted graphs on Desmos to
 get the bifurcation points
%  6.68 < L < 6.76     (2 to 4 as L rises)
%  13.52 < L < 13.59   (4 to 2 as L rises)
```

Euler's method calculation %%

```
% format long;

% Euler's method for numerically solving a differential equation %

% clear
% clc
% format longG
% for L = [5.4, 8.1, 16.3]
%     dy_dt = @(t,y) (0.65.*y.*(1-(y./L))) - ((1.2.*(y.^2))./(1 +
 (y.^2)));        % The differential equation dy/dt as a function of t
 and y (f(t,y))
%     y = 1;
            % y value for the initial value problem
%     t = 0;
            % t value for the inital value problem
%     n = 60/0.01;
            % number of iterations
%     h = 0.01;
            % step size
%
%     y_values = zeros(n+1,1);              % vector to store y values
%     t_values = zeros(n+1,1);              % vector to store t values
%     n_values = (0:1:n)';                  % vector to store n values
%
%     y_values (1) = y;                     % put in intial values
%     t_values (1) = t;                     % put in intial values
%
%     for i = 1:1:n                         % for loop implementing
 Euler's method
%         y = y + dy_dt(t,y)*h;
%         t = t+h;
%         y_values (i+1) = y;
%         t_values (i+1) = t;
%     end
%
%     disp (y_values(end));
```

```
%     figure(1);
%     set(gca, 'FontSize', 14)
%     ax = gca;
%     ax.XAxisLocation = 'origin';
%     ax.YAxisLocation = 'origin';
%     hold on;
%     plot (t_values, y_values,'LineWidth', 2);
%     ylabel('y (hundreds of fish)','FontSize', 16);
%     xlabel('t (time in days)','FontSize', 16);
% end
% legend('Rainbow Trout (L = 5.4)', 'Brown Trout (L = 8.1)', 'Brook
 Trout (L = 16.3)');
```

Published with MATLAB® R2019b

D Python code

The following code was adapted from https://pypi.org/project/SlopeFields/ and used to generate the slope fields for different species of trout.

```
from __future__ import print_function # used for 2,3 compat
from __future__ import division # used for 2,3 compat
import argparse # used to parse arguments
import matplotlib.pyplot as plt # used for graphing
import numpy as np # used for fast arrays and np.arange
import math # used for math.sqrt
from math import * # used for when we eval() math expressions
from sys import argv # used to grab the first argument (aka the
    equation)

np.seterr(all='ignore') # disable numpy warnings

parser=argparse.ArgumentParser(description='Generate a slope field
    for a given function. ')
parser.add_argument('--xMin', dest='xMin', help='Minimum x value. ')
parser.add_argument('--xMax', dest='xMax', help='Maximum x value. ')
parser.add_argument('--yMin', dest='yMin', help='Minimum y value. ')
parser.add_argument('--yMax', dest='yMax', help='Maximum y value. ')

parser.add_argument('--initX', dest='initX', help='The initial x
    value. ')
parser.add_argument('--initY', dest='initY', help='The initial y
    value. ')

parser.add_argument('--dX', dest='dX', help='The dX value used in
    euler\'s method. ')
parser.add_argument('--line', dest='line', help='If you want to draw
    a line connecting the dots. Note this may cause problems on
    functions with asymptotes. ', action='store_true')
parser.add_argument('--approximate', dest='approximate', help='If
    you want to approximate f(a). ')
parser.add_argument('--output', dest='output', help='The location
    where you would like to save the output image')
parser.add_argument('L', help='value of L')
parser.add_argument('trout', help='name of trout')
args = parser.parse_args()

initX = 0 # starting x value
if args.initX:
    initX = float(args.initX)
initY = 1 # starting y value
if args.initY:
    initY = float(args.initY)
dX = .0001 # the value that is used for dX in the euler's method
    calculation
```

```
if args.dX:
    dX = float(args.dX)
numberPoints = 1000000 # the max number of points we calculate when
    doing euler's method
step = .5 # the dX and dY between the slopes on the slope field
lengthSlope = .5 # the length of the slope

xMin = -10 # use to adjust the domain and range of the graph
if args.xMin:
    xMin = float(args.xMin)
xMax = 10
if args.xMax:
    xMax = float(args.xMax)
yMin = -10
if args.yMin:
    yMin = float(args.yMin)
yMax = 10
if args.yMax:
    yMax = float(args.yMax)

equation =
    "(0.65*(1-y/"+args.L+")*y-(1.2*y**2)/(1+y**2))"#args.equation.replace('^','**')

def getPoint(x, y): # gets a point to be plotted on the slope field
    return [x, y, dydx(x,y)] # returns xVal, yVal, slope

def dydx(x, y): # calculates the slope at a given point
    return eval(equation)

def calcPointsFromPointAndSlope(x, y, slope, len): # in order to
    plot a slope at a point, we need to generate the two points we
    want connected by the line
    a= math.sqrt((len*len)/(4+4*slope*slope)) # a is calculated
        based off of the len of the line and the slope so as to
        ensure all the slopes have a constant length
    x1 = x-a
    x2 = x+a
    y1 = y-a*slope
    y2 = y+a*slope
    return [(x1,y1),(x2,y2)] # returns a list of two points

def getPointsFromEulersMethod(initX, initY, dX, numberPoints,
    equation): # returns a list of points that were found using
    euler's method
    def getRow(x, y, dx): # returns a row in the the euler's method
        table
       return [x, dx, y, dydx(x,y)*dx]

    def floatingEquals(a, b):
       return abs(a-b) <= dX
```

```
haveFoundApproxSolution = False

# we calculate forward and backward from the initX point
forwardTable = [getRow(initX, initY, dX)] # initial list that
    will hold all of the rows for euler's method going forward
currCountPoints = 0 # the number of points calculated so far
approximateAnswer = False
while currCountPoints < numberPoints: # while we have calculated
    fewer than numberPoints
    y = forwardTable[-1][2] # the y value
    try:
        denom = eval(equation.split('/')[1]) # calculate the denom
    except:
        denom = 1 # if no denom, then the denom is 1
    if (denom < -.1 or denom > .1): # if denom!~=0: used so as to
        avoid problems with function definition
        try:
            row = getRow(forwardTable[-1][0]+forwardTable[-1][1],
                forwardTable[-1][2], forwardTable[-1][1]) # get
                the next row based off of the last row
        except:
            break
        row[2] = row[2]+forwardTable[-1][3] # update the y value
            in the row by addying the previous dy
        forwardTable.append(row) # add the row to the table
    if abs(forwardTable[-1][0]) > max([abs(xMin), abs(xMax)]): #
        if the x value > 10 or < -10, then we have gone past the
        axes on the graph so break
        break
    if args.approximate and not haveFoundApproxSolution:
        if floatingEquals(forwardTable[-1][0],
            float(args.approximate)):
            print("f("+args.approximate+")="+str(forwardTable[-1][2]))
            haveFoundApproxSolution = True
            approximateAnswer = forwardTable[-1][2]
    currCountPoints += 1
currCountPoints = 0 # reset for calculating backwards
backwardTable = [getRow(initX, initY, dX)]
while currCountPoints < numberPoints:
    y = backwardTable[-1][2]
    try:
        denom = eval(equation.split('/')[1])
    except:
        denom = 1
    if (denom < -.1 or denom > .1):
        try:
            row =
                getRow(backwardTable[-1][0]-backwardTable[-1][1],
                backwardTable[-1][2], backwardTable[-1][1])
        except:
```

```
                break
            row[2] = row[2]-backwardTable[-1][3]
            backwardTable.append(row)
        if abs(backwardTable[-1][0]) > max([abs(xMin), abs(xMax)]):
            break
        if args.approximate and not haveFoundApproxSolution:
            if floatingEquals(backwardTable[-1][0],
                float(args.approximate)):
                print("f("+args.approximate+")="+str(backwardTable[-1][2]))
                haveFoundApproxSolution = True
                approximateAnswer = backwardTable[-1][2]
        currCountPoints += 1
    fullTable = backwardTable[::-1] + forwardTable # full table of
        points is both forward and backward combined
    return ((args.approximate, approximateAnswer), [(item[0],
        item[2]) for item in fullTable]) # we only need the x and y
        coordinates (not the dx and dy)

def getListOfPointsAndSlopes(step): # returns a list of points and
    slopes for plotting on the slope field
    table = [] #holds a list of tuples: (x, y, dy/dx)
    for x in np.arange(xMin, xMax, step): # x in range between -10
        and 10 incremented by the step size
        for y in np.arange(yMin, yMax, step): # same but for y
            try:
                table.append(getPoint(x,y))
            except ZeroDivisionError: # if the slope has a divide by
                zero error, we pass
                pass
            except ValueError:
                pass
    return table

table = getListOfPointsAndSlopes(step) # get list of points and
    slopes to be put on the slope field

ax = plt.subplot() # create the axes

for pointSlope in table: # iterate through the point and slope pairs
    points = calcPointsFromPointAndSlope(pointSlope[0],
        pointSlope[1], pointSlope[2], lengthSlope) # get the points
        rather than a point slope pair
    ax.plot([points[0][0], points[1][0]],
        [points[0][1],points[1][1]]) # graph a line between those
        points

ax.grid() # enable the grid layout on the graph
ax.set_xlim([xMin, xMax]) # set the axes limits
ax.set_ylim([yMin, yMax])
```

```
out = getPointsFromEulersMethod(initX, initY, dX, numberPoints,
    equation) # get all of the points from euler's method
points = out[1]
xPoints = [x for [x,y] in points] # break them into lists of x
    values and y values
yPoints = [y for [x,y] in points]

out2 = getPointsFromEulersMethod(0, 0.01, dX, numberPoints, equation)
points2 = out2[1]
xPoints2 = [x for [x,y] in points2] # break them into lists of x
    values and y values
yPoints2 = [y for [x,y] in points2]

out3 = getPointsFromEulersMethod(0, 15, dX, numberPoints, equation)
points3 = out3[1]
xPoints3 = [x for [x,y] in points3] # break them into lists of x
    values and y values
yPoints3 = [y for [x,y] in points3]

'''
if out[0][1]:
    print("Here")
    ax.plot((out[0][0], out[0][0]), (yMin, yMax), 'b', linewidth=2,
        zorder=100)
    ax.plot((xMin, xMax), (out[0][1], out[0][1]), 'b', linewidth=2,
        zorder=100)
    plt.title("Slope Field: " + equation.replace('**','^')+" and " +
        "f(" + str(initX) + ")=" + str(initY) +
        "\nf("+str(out[0][0])+")="+str(out[0][1]))
'''

if args.line:
    ax.plot(xPoints, yPoints, 'b-', linewidth=3) # graph the points
        from euler's method
    ax.plot(xPoints2, yPoints2, 'k-', linewidth=3) # graph the
        points from euler's method
    ax.plot(xPoints3, yPoints3, 'b-', linewidth=3) # graph the
        points from euler's method
else:
    #ax.plot(xPoints, yPoints, 'ro', linewidth=1)
    None

if not out[0][1]:
    plt.title("Slope Field ("+args.trout+" Trout L = "+args.L+"):")

if args.output:
    plt.savefig(args.output)

plt.ylabel("y [100 fish]")
plt.xlabel("time [days]")
```

```
plt.show() # display the graph
```