Benjamin Asdell
CEE/MAE M20
UID: 005114992

Homework 3

October 15, 2020

1. Population of Two Species Under Competition

1. Introduction

   The objective of this problem is to calculate and plot the varying population sizes of two species that are directly competing with each other. The algorithm takes in given constants and initialized values and calculates the change in each population size for each time step. The values are then added to an array and plotted over the total time duration.

2. Models and Methods

   This program does not take in any user inputs. All constants are hardcoded into the program as specified by the assignment. The following Lotka-Volterra equations are used as the basis of the algorithm for computing the population sizes of X and Y respectively from each time step to the next.

$$\frac{dx}{dt} = x(\alpha - \epsilon x - y)$$

$$\frac{dy}{dt} = y(-\gamma + \rho y - \delta x)$$

   where x, y correspond to the populations,in thousands, of X and Y, respectively. Using the forward Euler method, the Lotka-Volterra equations can be discretized to the following forms:

$$x(k) = x(k-1) * \Delta_t(\alpha - \epsilon x(k-1) - y(k-1))$$

$$y(k) = y(k-1) * \Delta_t(-\gamma + \rho y(k-1) - \delta x(k-1)$$

   where k signifies the current time step being calculated. The value of k corresponds to the iteration of the MATLAB `for` loop being executed to calculate that specific population value. Note that instances of k in these discretized equations does not imply multiplication, but rather that the x or y is being evaluated at that k value. The values of X and Y for each k are stored in their own arrays, which for a time step of 0.01 becomes a 1x1000 array. Similarly, the t array that holds all times spaced at intervals of the time step is also 1x1000. This means that MATLAB plots 2000 data points to create the final plot with a time step of 0.01.

3. Calculations and Results

   For the first example, the constants are all initialized according to the assignment's first instructions, including a time step of 0.01, initial X = 5, and initial Y = 2. The following is printed to the command window:

```
The final X population is: 1.54474 (thousands)
The final Y population is: 1.20446 (thousands)
```

Examining the runtime of the program given this specific input returns an elapsed time of 0.891473 seconds.

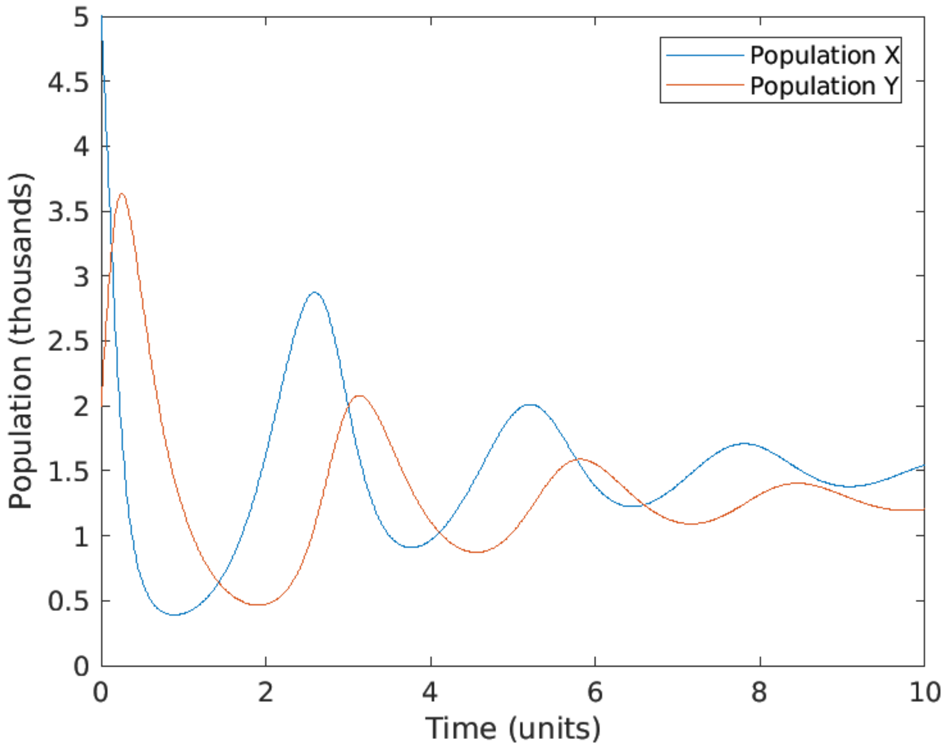

Figure 1: The result plot from this first example input. As with the previous competing species plot in lecture, one population peak lags behind the other population peak in time. From this plot, one can reasonably conclude that the peaks and troughs for both populations will eventually converge to distinct but constant values if the time duration is extended.

For the second example, all constants are initialized the same as the first except the time step, which is changed to 0.001.

```
The final X population is: 1.53592 (thousands)
The final Y population is: 1.22068 (thousands)
```

Examining the runtime of the program given this specific input returns an elapsed time of 1.036938 seconds. This is relatively slower than the first example input, which is expected considering the program plots 20000 points with a time step of 0.001, compared to 2000 with a time step of 0.01. Also note that the population values do not change significantly from the first input, which correlates with the conclusion from the first plot.

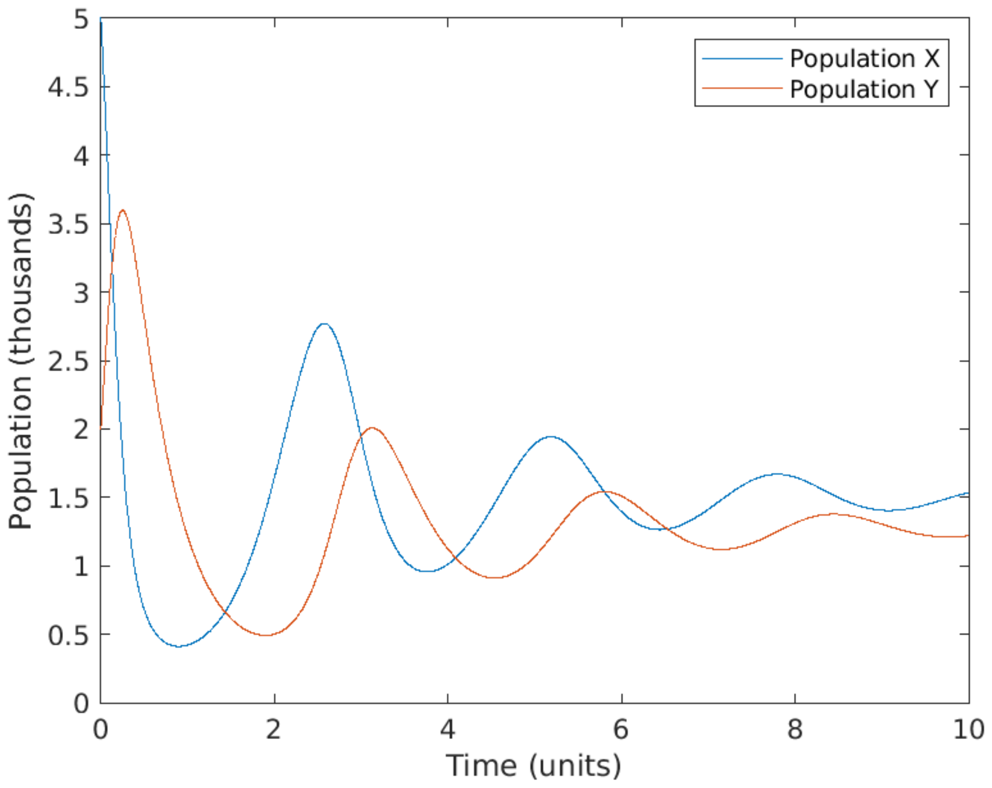

Figure 2: This plot from the second example shows that using a time step that is a tenth of the previous value has no significant impact on the shape of the plot. There are 10 times as many points plotted, which might create a smoother curve, but the overall shape and conclusion of convergence does not change.

```
The final X population is: 1.52736 (thousands)
The final Y population is: 1.03547 (thousands)
```

Examining the runtime of the program given this specific input returns an elapsed time of 0.704983 seconds, which is faster than the original example input. This is also to be expected considering MATLAB only plots 200 points per population (400 points overall), or a fifth of the computation and plotting from the original input.

This continues to agree with the conclusion that smaller time steps make the program take longer to execute.

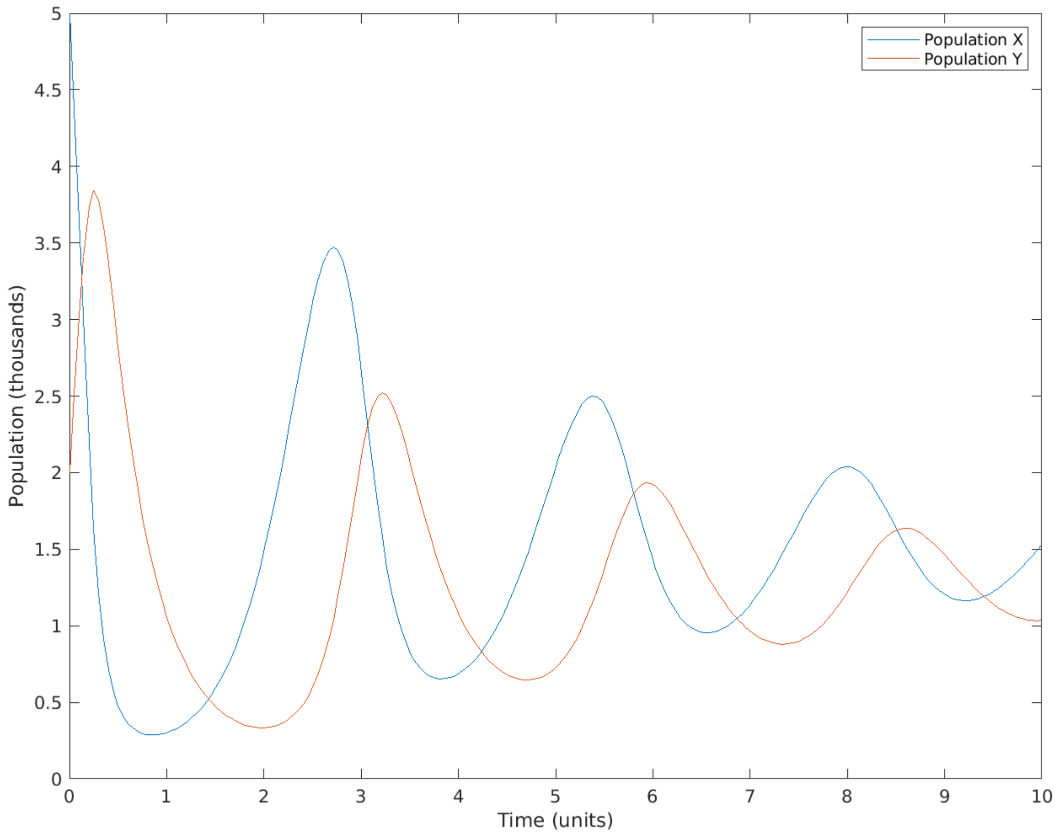

Figure 3: While this plot has only a fifth of the plotted points compared to the first example, the overall shape of the plot is relatively the same. It is definitely more discrete (less smooth), but the peaks occur at basically the same times in both plots. Even the final population values are similar, with a slight variation from the accepted 'true' value from the first example that is expected from having less time steps to evaluate.

4. Discussion

This program is a useful simulation of two directly related functions over time. While many factors were ignored in the competition of the two species, this program does allow the user to make conclusions about competing populations, especially based on the plots generated. On the other hand, this is a great introductory simulation using MATLAB to compute and plot thousands of values in mere seconds, whereas doing this simulation by hand would take a ridiculous amount of time and effort. This program gives a small taste of the versatility and power of MATLAB to make these types of simulations and facilitate conclusions on mathematical and physical trends. The elapsed time calculations using MATLAB's `tic toc` commands are also extremely useful in learning about and testing the limitations of MATLAB to compute over extremely large arrays or data sets. Smaller time steps in this problem led to less points being generated, which led to faster (but less accurate) simulation of the two populations.

2. Improved Population of Two Species Under Competition

1. Introduction

   The goal of this problem is to improve upon the previous simulation of two species competing by adding a new effect on Population X. This effect begins at t = 5, which is half the total time duration, and continues to affect population X until the final time value. The new final populations are then printed to the command window and a new plot is generated.

2. Models and Methods

   Algorithmically, this problem uses the same basis as Problem 1. The discretized Lotka-Volterra equations are still used in the same way. The only alteration to the algorithm is the modification of the a value (seen as α in the Lotka-Volterra equations) to be time varying ONLY after t = 5. The modification is as such:

   $$\alpha(t) = \alpha * e^{-\frac{(t-t_l)^2}{2\sigma^2}}$$

   The assignment declares $t_l = 5$ (as above) and $\sigma = -0.4$. The standard time step of 0.01 is used for this problem.

3. Calculations and Results

   When the program is executed, the following is printed to the command window and the plot is the generated:

```
The final X population is: 0.14536 (thousands)
The final Y population is: 0.00080 (thousands)
```

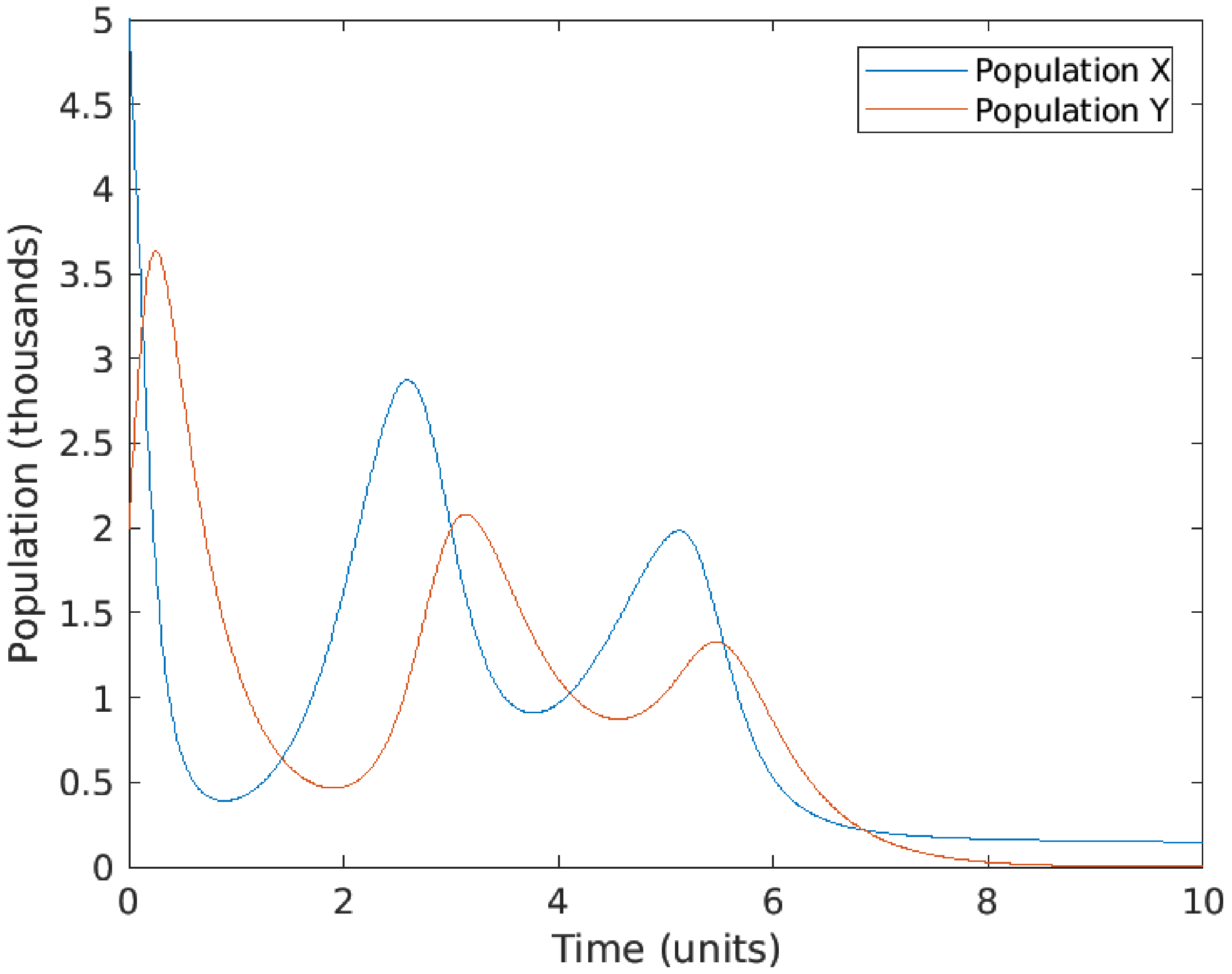

Figure 4: This plot is the representation of the improved algorithm for the two species under competition. Notice the sharp decline in Population X after the added effect begins at t = 5, followed by a slower, delayed decline in Population Y.

This result is interesting because the final population values are vastly different than the original simulation, even at the same time step as the first example of the previous problem. Clearly, the new effect on X is very detrimental to the success of Population X.

4. Discussion

This improvement upon the previous problem adds insight into the two species under competition. From the plot generated in this problem and the previous one, it can be concluded that Population Y is the predator species since predator peaks always lag behind prey peaks in time (as discussed in lecture). This is confirmed by the fact that Population Y also faces a decline in success after Population X experiences a sharp decline due to the new factor. If Population x were the predator species, the detrimental effect on X would probably cause a population surge in Population Y. However, Population Y fell also despite the new effect having no direct impact on its discretized Lotka-Volterra equation. Logically this makes sense because a declining prey population causes a shortage of food for the predator species, which will cause a decline in predator population after a short delay.

3. Permutation Calculator

1. Introduction

   The purpose of this problem is to create simple calculator for computing the number of permutations of a selected sample size there can be from a total number of elements. The program takes in user inputs for n and r, which are the mathematical standards for signifying the total number of elements and the sample size, respectively. The result of the calculation is then printed to the command window.

2. Models and Methods

   The basis of the calculation is the ${}_n\mathrm{P}_r$ formula from statistics.

   $$P(n,r) = \frac{n!}{(n-r)!}$$

   As the assignment states, the MATLAB built-in `factorial` function is not allowed, so the code to compute factorials is a custom written function. In place of the `factorial` function, the `prod` is used to find the product of all the integer values from 1 to the value being evaluated (either n or $n-r$). Other than this change, the permutation function can be written directly into MATLAB as it appears above.

3. Calculations and Results

   When the program is executed with n = 5 and r = 0, the following is printed to the command window:

```
Enter an integer n for the total number of objects:
5
Enter an integer r for the sample:
0
The number of permutations is 1.
```

   This result can be checked with a scientific calculator, but logically (r=0) the numerator and denominator will cancel and the result is just 1. The next calculation is an example of an input of n = 10 and r = 6.

```
Enter an integer n for the total number of objects:
10
Enter an integer r for the sample:
6
The number of permutations is 151200.
```

Checking again with a calculator, this is the correct result. There is nothing special about this case, but it demonstrates that the number of permutations increases significantly as n increases, which is important for the next input. The next input is specifically requested by the assignment, n = 1000 and r = 0.

```
Enter an integer n for the total number of objects:
1000
Enter an integer r for the sample:
0
The number of permutations is NaN.
```

This result is interesting because performing this permutation calculation on a scientific or graphing calculator will yield a result. The actual answer can be logically derived if one recognizes from the first example that any calculation with r = 0 should yield 1 as the answer. The reasoning behind MATLAB's answer of NaN (which stands for not an number) will be discussed below.

4. Discussion

This permutation calculator is a relatively simple script that directly converts a mathematical formula into MATLAB code. Any complexity in that conversion is due to the assignment's restriction on the `factorial` function. The most intriguing aspect of this script is its limitations due to MATLAB's internal coding.
By modifying the code slightly to allow the numerator and denominator values to be printed along with the final result, the third example input does make sense. Both the numerator and denominator, which would be equal to 1000!, evaluate to Inf. This is MATLAB's notation for an infinite value. Clearly, 1000! is not actually infinite, but it is unfathomably large. Even in scientfic notation, 1000! has an exponent above 2500. My conclusion is that MATLAB cannot store every digit of such a large number within any data types, and therefore reverts the value to Inf so it does not cause a memory allocation issue. Since the example input would then become $\frac{Inf}{Inf}$, it returns NaN, or a null value that signifies the value is undefined. While some mathematicians would argue over the value of that evaluated expression, MATLAB takes the most accepted and simplest route by stating the value is undefined.