Duncan Di Mauro
UID: 805163177
CEE/MAE M20
October 30, 2019

Homework 4

1 Correlation Coefficient Between Two Populations

1.1 Introduction

The goal in this problem is to create a script that computes the population of two species under competition. The script should create a labelled graph that plots the populations of each species over a given time interval. Finally, the script should extract one period's worth of data for each species starting from a given time, and use that data to compute and print the value of the Pearson's correlation coefficient. This calculation should be done using array operations and/or iteration. The populations of these two species are governed by the following Lotka-Volterra equations:

$$\frac{dx}{dt} = \quad 0.4x - 0.018xy \tag{1}$$

$$\frac{dy}{dt} = -0.8y + 0.023xy \tag{2}$$

where *x*, *y* are the populations (thousands) of hare and lynx, respectively, and *t* is time in units of years.

1.2 Models and Methods

The forward Euler method was used on the given Lotka-Volterra equations to find the discretized governing equations for the two populations. The resulting equations are shown below:

$$x_{k+1} = x_k + \Delta t(0.4x_k - 0.018x_k y_k) \tag{3}$$

$$y_{k+1} = y_k + \Delta t(-0.8y_k + 0.023x_k y_k) \tag{4}$$

Where x_k and y_k represent the populations of hare and lynx at step k, and x_{k+1} and y_{k+1} represent the values of hare and lynx at step k + 1.

First, the script assigns all given values to variables (ex. `delta_t = 0.01`). Next, a value for the number of steps needed to go from time t = 0 to t = 10 was found, and was assigned to the variable `t_steps` (the method used here was the same method and reasoning used in homework 3). Then, the arrays that we will be computing (the population values for each moment of time) are pre-allocated in order to make the script run more efficiently. This is done by creating arrays consisting of zeroes with t_step entries (since that is the amount of population values our script will find). Additionally, an array for time is created, going from t_initial to t_final with t_step entries. Next, arbitrary negative values are assigned to variables `xpeak` and `ypeak`, representing the peak population values for hare and lynx. These are initially set to be negative values so that they can be overwritten later by larger values that might actually represent the data peaks.

Next, a `for` loop with the condition `for k = 1:t_steps` is used to record the population values for X and Y at every step, starting from their initial values until their values at the final step. The values of

the x-array and y-array at column *k* (each column represents a moment in time) are then set equal to the current values of x_k and y_k, as shown:

```
x(k) = xk;
y(k) = yk;
```

The values of `xk` and `yk` will be updated by each iteration of our `for` loop, and are originally set equal to their initial given values at the beginning of the script. This process is done by inserting equations (3) and (4) into the middle of our loop, and assigning them to variables `xkp1` and `ykp1`, which are equivalent to x_{k+1} and y_{k+1}. Then, at the end of the loop, x_k and y_k are set equal to these new population values. After the loop ends, the x-array values and y-array values are graphed against the time-array values on a labelled plot.

The values for the population peaks are found by using an `if` statement in the `for` loop. If `xk` is greater than `xpeak` (this statement is repeated for `ypeak` as well), then `xpeak` is assigned to the current value of `xk`. Then, the current time value is printed if 1) `xk` is the current peak value (this is necessary so that the command window is not flooded every iteration) and 2) the value of `xkp1` is less than the peak value (this is done to ensure that the value is at a peak, i.e., the population is not rising any more). These values are then written down, and a period for each species is determined by taking the average of the time differences between consecutive peaks. The periods for the x-population and y-population are then confirmed to be near-equal, and are rounded slightly to equal each other exactly (this makes finding the correlation coefficient much easier, as it allows our arrays representing one period of x and y to be the same size).

Next, the script extracts one period's worth of data for each species starting from *t* = 5.2. This is done using the following method explained in pseudocode:

```
x_extracted = x(step-value of initial time : step-value of initial
time + one period)
```

Then, the extracted arrays, along with their mean-values (found using the `mean` function in MATLAB), are plugged into the equation for Pearson's linear correlation coefficient

$$r_{xy} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}} \tag{5}$$

where overbar indicates the average value. In MATLAB, this is computed with algebraic and array operations, and the use of the `sum` function. Finally, the script prints this value.

1.3 Calculations and Results

When the program is executed, the following output is printed to the screen and the following figure is created (note that in order to save space in this report, I only show one segment of the peak population data that was printed):

```
Peak X Population Value: 103.635098
                   Time: 5.040000
```

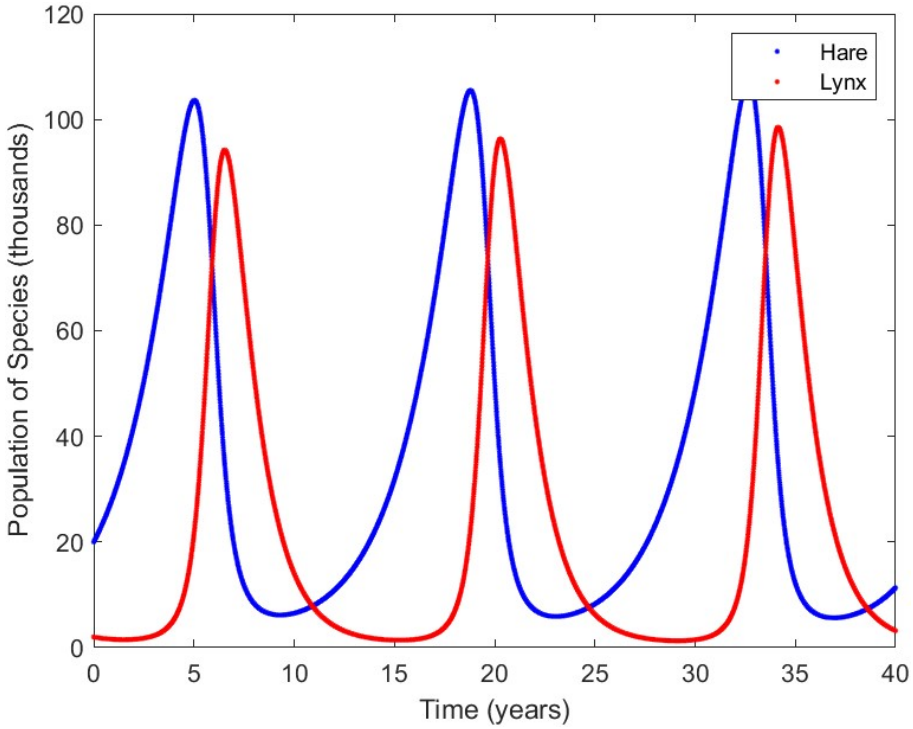
```
      Peak Y Population Value: 94.214548
                         Time: 6.550000

      Peak X Population Value: 105.508707
                         Time: 18.800000
```

…

…

```
The correlation coefficient rxy =  -0.000431
```



1.4 Discussion

There were various new techniques that I learned in this problem. I was able to further my understanding of arrays and learn how to fill an array with an updating value in a `for` loop, and I learned how to use various array operations and a few new functions (`sum`, `mean`, `sqrt`). Additionally, I learned one method of finding peak values of a data set. The correlation coefficient obtained in this problem is very small and is close to zero. This means that the populations for hare and lynx do not have a high linear correlation. In fact, they have almost zero linear correlation. This is to be expected, as the graph does not indicate any sort of linear relationship. For example, in the beginning of the graph, the lynx population stays relatively flat and goes down, while the hare population moves up at a very steep slope. Then, the hare population moves down at roughly the same steepness. If they had a linear correlation, the lynx population would stay relatively flat and go up (when the hare's slope is inverted, the lynx's should be too). However, this is not the case, as the Lynx population goes up rapidly.

2 Write a MATLAB program that simulates pendulum

2.1 Introduction

The goal in this problem is to create a script that simulates the angular motion of the pendulum for a given time interval. The motion should be simulated four times, once using the explicit Euler method and once using the semi-implicit Euler method for both damped and un-damped motion. The kinematic equation of a pendulum is:

$$\frac{d\omega}{dt} = -\frac{g}{L}\sin(\theta) \quad (6)$$

$$\frac{d\theta}{dt} = \omega \quad (7)$$

where ω is the angular velocity, g is the acceleration due to gravity (g = 10 m/s/s), L = 10m, and θ is positive in the anti-clockwise direction. When we take damping force into account, the kinematic equation becomes:

$$\frac{d\omega}{dt} = -\frac{g}{L}\sin(\theta) - \omega L \times d \quad (8)$$

$$\frac{d\theta}{dt} = \omega$$

where, in this equation, the damping force parameter is d = 0.01/m-s. Additionally, several plots should be created to graph angle vs. time – one for both methods with undamped motion, one for both methods with damped motion, and one for the semi-implicit damped motion with varying values of d.

2.2 Models and Methods

The discretized governing equations obtained using the forward Euler method are shown below for damped and undamped motion:

(Undamped) $$\omega_{k+1} = \omega_k - \Delta t\left(\frac{g}{L}\sin(\theta_k)\right) \quad (9)$$

(Damped) $$\omega_{k+1} = \omega_k - \Delta t\left(\frac{g}{L}\sin(\theta_k) + \omega_k L \times d\right) \quad (10)$$

(Damped and Undamped) $$\theta_{k+1} = \theta_k + \Delta t(\omega_k) \quad (11)$$

The implicit portion of the equations obtained using the semi-implicit Euler method (the other equations are the same as above) is shown below:

(Damped and Undamped) $$\theta_{k+1} = \theta_k + \Delta t(\omega_{k+1}) \quad (12)$$

In these equations θ_k and ω_k represent the angle (rad) and angular velocity, respectively, of the pendulum at step k, and θ_{k+1} and ω_{k+1} represent the values at step k + 1.

First, the script assigns all given and initial values to variables. It is important to note that all four simulations having different variables for $\omega_k, \omega_{k+1}, \theta_k$, and θ_{k+1}, and all have different variables for their arrays representing the angle at a certain moment in time. Next, a value for the number of steps needed to go from time t = 0 to t = 50 was found using the same method as problem 1. Then, arrays

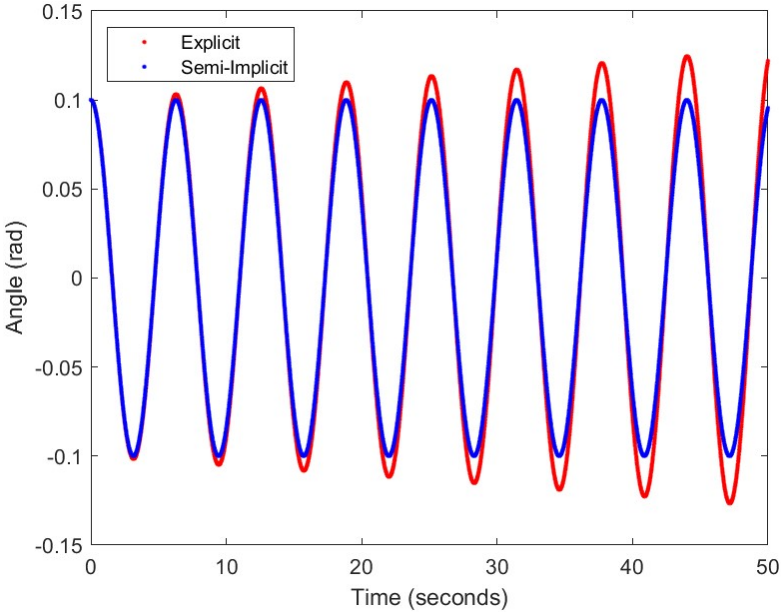
containing angular values for all four simulations are pre-allocated in the same fashion as problem 1. Additionally, an array for time is created, going from t_initial to t_final with t_step entries.

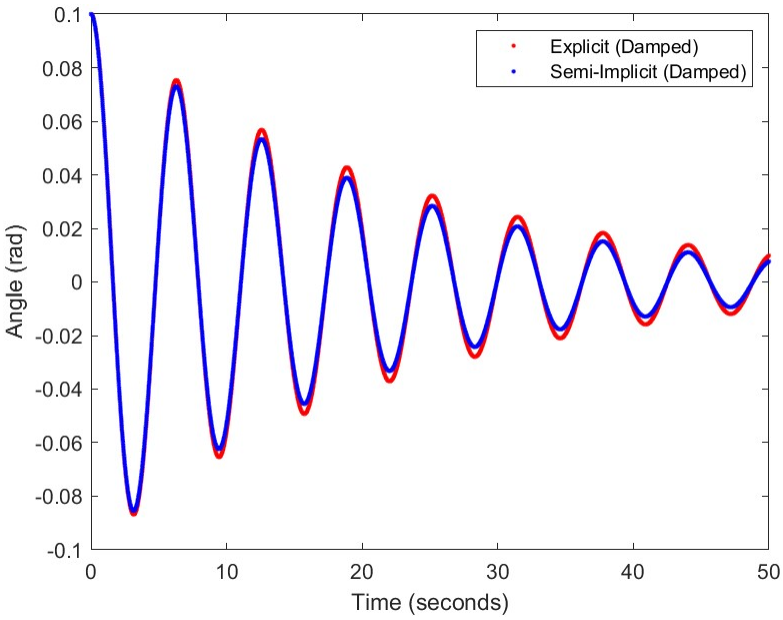
Next, a `for` loop with the condition `for k = 1:t_steps` is used to record angular values for all four simulations at every step. The values of θ and ω for each simulation are obtained, updated, and assigned to their respective array values in the same fashion as the population values/arrays in problem 1. It is important to note that when coding for the semi-implicit method, the value for ω_{k+1} should be obtained in the line before the value of θ_{k+1} is obtained. This way, equation (12) can utilize the correct, updated value of ω_{k+1}. After the loop ends, the arrays containing undamped angular values (both methods) are plotted against time on one graph, and the arrays containing damped angular values (both methods) are plotted against time on another graph.

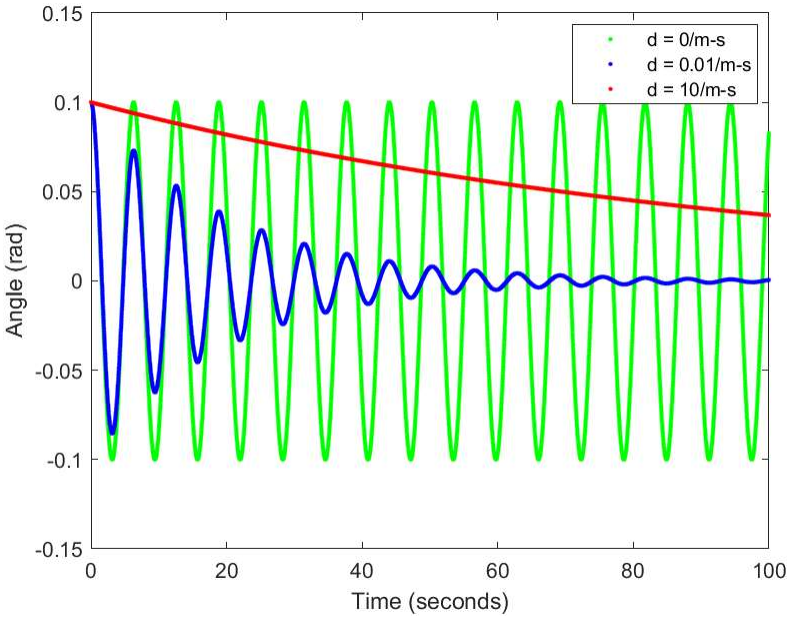
Finally, another graph is created (this is done on a separate .m file) utilizing the semi-implicit damped simulation and three different values of *d* (*d* = 0, *d* = 0.01, and *d* = 10). This is done in the same way that all the other graphs are created and utilizes different variables for all three semi-implicit damped simulations.

2.3 Calculations and Results

When the program is executed, the following figures are created. Additionally, the figure utilizing the semi-implicit damped simulation with three different values of *d* is shown here:







2.4 Discussion

In terms of the code, this problem is very similar to problem 1 and does not require any more knowledge than problem 1. The part that makes this problem interesting is the use of the semi-implicit Euler method (as discussed in the methods section) and the results of the graphs created. Other than these two things, there was nothing new learned through this problem. In the first graph, we notice that the pendulum's oscillation amplitude stays the same for the semi-implicit Euler method, but increases over time for the explicit Euler method. The pendulum's amplitude should stay the same, so we can conclude that the use of the explicit Euler method introduces artificial energy into the system. When

looking at the graph for the damped observation, we notice that both the explicit and semi-implicit Euler methods result in a decrease of oscillation amplitude over time. This is due to the damping factor introduced to the system. Additionally, the explicit Euler method still has larger oscillation amplitudes than the semi-implicit method, so we can assume that it is still introducing artificial energy into the system. When looking at the final graph comparing semi-implicit damped pendulum simulations with different values for d, we notice that when d = 0 the graph is the same as if it were not damped, and when d = 0.01 it is the same as the previously graphed semi-implicit damped motion. However, when d = 10, the pendulum's angle slowly decreases and approaches 0, with no oscillation taking place. We can therefor conclude that in this case, the pendulum is either overdamped or critically damped.