Jeffrey Ding
UID: 104928991
CEE M20
April 20th, 2018

Homework 3

1. The Three Species Problem

1.1 Introduction

The goal of this problem is to model three groups of imaginary creatures using the Lotka-Volterra equations.

1.2 Model and Methods

The script starts by setting x, y, and z variables, representing the original populations at t = 0, using the following lines of code:

```
x = 2;
y = 2.49;
z = 1.5;
```

Then, the script runs the variables through the Lotka-Volterra equations in order to calculate a differential element for each variable in respect to time.

Afterward, the script prints the data while calculating it at the same time. Using the following for loop:

```
for t = 0:0.001:12
```

Which iterates from 0 to 12 in 0.001 increments, the script first decides whether or not to print the current values using the following if statement:

```
if rem(t, 0.5) == 0
      fprintf(' %5.1f   %5.2f   %5.2f   %5.2f\n', t, x, y, z);
End
```

Then, the script recalculates the differential elements using the Lotka-Volterra equations and recalculates x, y, and z using the Forward Euler's method. The following lines of code show this in action:

```
dxdt = 0.75*x*(1 - x/20) - 1.5*x*y - 0.5*x*z;
 dydt = y*(1 - y/25) - 0.75*x*y - 1.25*y*z;
```

```
dzdt = 1.5*z*(1 - z/30) - x*z - y*z;
x = x + dxdt * 0.001;
y = y + dydt * 0.001;
z = z + dzdt * 0.001;
```

1.3 Calculations and Results

When the program is executed, the following output is printed to the screen:

Time	X	Y	Z
0.0	2.00	2.49	1.50
0.5	0.59	1.43	0.75
1.0	0.26	1.29	0.65
1.5	0.12	1.29	0.65
2.0	0.06	1.33	0.67
2.5	0.03	1.37	0.70
3.0	0.01	1.40	0.72
3.5	0.00	1.42	0.74
4.0	0.00	1.42	0.75
4.5	0.00	1.42	0.77
5.0	0.00	1.39	0.79
5.5	0.00	1.35	0.82
6.0	0.00	1.28	0.88
6.5	0.00	1.15	0.99
7.0	0.00	0.94	1.21
7.5	0.00	0.63	1.66
8.0	0.00	0.28	2.66
8.5	0.00	0.05	4.79
9.0	0.00	0.00	8.55
9.5	0.00	0.00	13.72
10.0	0.00	0.00	19.23
10.5	0.00	0.00	23.72
11.0	0.00	0.00	26.67
11.5	0.00	0.00	28.33
12.0	0.00	0.00	29.19

The output shows that overtime, group x and y die out while z thrives.

1.4 Discussion

The output shows that in the default initial populations, x and y are wiped out by z. If experimented with multiple different initial populations, a pattern emerges that shows that one species, typically the most plentiful at the start but not always the case, will wipe

out the other two species. I was unable to find a peaceful balancing point and therefore can only conclude that one species will crowd out the other two.

When I timed the results, the default delta t of 0.001 produced a code that ran around 0.001 to 0.0015 seconds. If I used a delta t greater than that, the time would decrease, which makes sense because the amount of iterations would greatly diminish. Vice versa, if I used a delta t less than 0.001, the time would increase because there would be a greater amount of iterations.

Using the initial conditions, species Z survives. If the time skip is increased to 0.01, all three species survive and species Y becomes the dominant species. However, the first interpretation offers the most accurate interpretation of the simulation because it has a smaller difference between each step, thus more accurate steps because each difference in t would be more accurate to the real model. By increasing the time skip to 0.01, I would be losing an entire order of magnitude.

2. The Pocket Change Problem

2.1 Introduction

The goal of this problem is to calculate the average number of coins in change you would receive after a cash transaction.

2.2 Model and Methods

The script begins by setting a totalCoin variable to 0 that cracks the total number of coins. Then, the script iterates through all the total possible amounts of change from 99 cents to 0 cents using the following for loop:

```
for totalChange = 99:-1:0
```

In the for loop, the script has a remainingCash variable that starts at the totalChange amount. Then, the script uses a while loop that runs while remainingCash is greater than 0 and uses an if ladder to check the largest denomination of coin to subtract from remainingCash. The following code shows this:

```
while remainingCash > 0
    if remainingCash >= 25
        remainingCash = remainingCash - 25;
    elseif remainingCash >= 10
        remainingCash = remainingCash - 10;
    elseif remainingCash >= 5
        remainingCash = remainingCash - 5;
```

```
        else
            remainingCash = remainingCash - 1;
        end
        totalCoins = totalCoins + 1;
    end
```

Basically, if greater than 25 cents, subtract a quarter, greater than 10 cents, subtract a dime, greater than 5 cents, subtract a nickel, else subtract a penny. After each iteration of the while loop, add a coin to totalCoins.

At the end, print the average number of coins using the following line of code:

```
fprintf('Average Number of Coins = %.2f\n', totalCoins/100);
```

2.3 Calculations and Results

When the program is executed the following output is printed to the screen:

```
Average Number of Coins = 4.70
```

This actually makes a lot of sense because the most pennies you can make to give change is 4, which is needed for a lot of change combinations.

2.4 Discussion

The print result shows that you can expect around 4.7 coins from a cash transaction. This makes a ton of sense because you can have a most 4 pennies in change. Then, there are a bunch of combinations, but the basis of this average would revolve around 4 pennies and another coin.

If we were to eliminate the penny and round all change values to the nickel value, the average number of coins would be around 3 coins. Basically, two dimes and a nickel can make up a quarter, which means if something exceeds a quarter, we can use a quarter to substitute. The combinations will typically be attained using only three coins.

If we change a quarters value to 27 cents and a dime to 11 cents, the average number of coins decreases from 4.7 to 4.44. However, not all changed values result in an lower average. When I change the values to 50, 25, and 10, the average number of coins went up to 5.8. When I change the values to 50, 25, and 5, the average went to 5. The best combination I could come up with was 25, 10, and 3. This made the average go down to 4.22. The only disadvantage of this new system would be breaking down change for a 10 or 25, because there are no direct ways to make 10 or 25 unless we you a lot of pennies and nickels other than using the actual coins.