Homework 3
Report

Jesse Harper
UID: 204669893

Problem 1:

Introduction 1.1

Computer programming is very well suited for use of numerical methods such as the forward Euler approach to finding good approximate solutions for initial value problems. This problem provides a multi-variable initial value problem to be solved using the forward Euler method. In this report, the populations of three imaginary species X,Y, and Z were solved for as functions of time by solving their governing Lotka-Volterra equations. This report covers the results of the program, as well as a discussion of these results.

Models and Methods 1.2

The forward Euler method for approximating the solution of a differential equation and initial value problem was used to solve for X,Y, and Z as functions of time. The following equations show the discretized solutions to the governing differential equations. Δt is the size of the time step.

$$X(k+1) = X(k) + (0.75 \times X(k) \times (1 - \tfrac{X(k)}{20}) - 1.5 \times X(k) \times Y(k) - 0.5 \times X(k) \times Z(k)) \times \Delta t$$
eqn 1.1

$$Y(k+1) = Y(k) + (Y(k) \times (1 - \tfrac{Y(k)}{25}) - 0.75 \times X(k) \times Y(k) - 1.25 \times Y(k) \times Z(k)) \times \Delta t$$
eqn 1.2

$$Z(k+1) = Z(k) + (1.5 \times Z(k) \times (1 - \tfrac{Z(k)}{30}) - X(k) \times Z(k) - Y(k) \times Z(k)) \times \Delta t$$
eqn 1.3

In this problem, the assigned time step is 0.001 s. The size of the time step is small to ensure that the errors due to the use of this first order approximation are small. However if runtime tight or precision is not as important the time step can be reduced to improved runtime. The following section goes over some sample calculations and results.

Calculations and Results 1.3

The following is a sample of the output from the program, run with the initial conditions provided in the problem statement as well as with a time step of 0.001s.

X_0	Y_0	Z_0
2	2.49	1.5

Table 1.1.1

Sample program output

```
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
```

By varying the time step, delta t, and using the tic and toc matlab functions, runtimes were calculated for each size of delta t. These values can be seen below

```
time step of 0.1000 seconds gives a runtime of 0.0003 seconds
time step of 0.0100 seconds gives a runtime of 0.0004 seconds
time step of 0.0010 seconds gives a runtime of 0.0043 seconds
time step of 0.0001 seconds gives a runtime of 0.0233 seconds
```

The following table shows 3 different initial conditions and the dominant species that appears as a result.

X_0	Y_0	Z_0	Dominant
3	3	3	Z
1	1	3	Z
1	3	1	Y

Table 1.1.2

The following table shows the difference in calculation result for two different time steps.

X_f	Y_f	Z_f	Δt
0	0	29.19	0.001s
0	21.27	0	0.01s

Table 1.1.3

Discussion 1.4

This program, and its changing performance, show the impact initial conditions and time step fidelity have on the results and accuracy of forward Euler numerical methods. In this particular problem the time step size of 0.001s and the initial conditions described in table 1.1.1. Yields the result that species Z dominates over the other three. However, variations in the initial conditions and time step size change the results: sometimes by huge margins.

Guessing several different initial conditions, while holding time step size constant, shows that you can change the outcome of the dominant species. However, there appears to be no way to set initial conditions such that all three species end up cohabitating. This can be seen in table 1.1.2.

By using the tic and toc matlab functions, the run time of the simulation can be seen. As the time step shrinks, the runtime of the program grows. The difference between runtimes for 0.1s and 0.01s is negligible, but the difference between 0.001s and 0.0001s is much large which implies that the runtime growth, as a function of time for time step does not scale linearly.

Problem 2:

Introduction 2.1

This problem solved for the average number of coins given as change for any transaction. Using foundation of this program, a short study was also performed to examine how changing or removing values of the coins in distribution would affect the average number of coins returned for any amount of change. Pennies were coin removed from circulation for the first study, and the value of quarters, dimes, and nickels were all changed in the second. It was found that both removing pennies and changing the value of the other change amounts both could reduce the average number of coins required to provide change for a given transaction.

Models and Methods 2.2

This problem was simplified by assuming that for every amount of change possible, between 0-99 cents, only the minimum number of coins is returned. This was accomplished by using an algorithm which, for every change amount, iteratively tried decrementing the highest value coin amount from that value until the remaining value was lower than that coin denomination. For each decrement, a counter would keep track of the number of coins used. This algorithm would then proceed to the next biggest value of currency, while counting coins used, until the total change was removed. At the end, the number of coins is used as a hash index for an array that keeps a score of the number of times a particular number of coins was returned for a change amount.

For example, if for a change value of 58 cents was examined by the algorithm, it was find that 6 coins were needed to change the value. So then the array which stored the scores, called "coins[]" would be updated in the following example.

```
coins( 6 + 1) = coins( 6 + 1) + 1;
```

The exception would be the case that no coins were need to change the value. In this case the first element of the array is incremented.

Once the full spectrum of change values was examined, a weighted average of the distribution was calculated and returned to the command window in following format.

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

$$average = (\sum_{i}^{k} i * n)/m \qquad \textbf{eqn 2.2.1}$$

Where *n* is the number of times *i* coins were required for change and *m* was the total number of coins required for the entire distribution.

Calculations and Results 2.3

The first few calculations done by this program are performed to study the average number of coins returned as change for the traditional base system of coins. The following table shows the results of these trials. The first one is with no modification. The second is with pennies removed from circulation.

Lowest Coin Value	Average Returned Coins
0.01	4.70
0.05	2.70

Table 2.2.1

The following table shows the result of modifying the base value of all the coins.

Quarter	27
Dime	11
Nickel	3
Penny	1
Average	4.16

Table 2.2.2

Discussion 2.4

It was found, as can be seen in table 2.2.1, that the average number of coins returned in a transactions is 4.70. It was also found, and can be seen in the same table, that the average can be lowered to 2.70 by removing pennies from circulation. An intuitive explanation as to why this is possible is that by removing them you reduce the largest number of coins which can be returned in order to provide change for a transaction. With pennies the largest number of coins which can be returned is 9 where as the largest without is only 5. By reducing the largest weights in the average, the average becomes smaller. Changing the value of each coin has the same effect