Zubin Mishra
UID: 604644805
CEE/MAE M20
April 25, 2017

Homework 3

1 The Three Species Problem

1.1 Introduction

The goal in this problem is to develop a program that will model the populations of three interacting species governed by Lotka-Volterra equations. This is accomplished by using the forward Euler method to get the discretized governing equations. Every so many time steps, results are printed to the screen in a table. It will be discussed if there are any equilibrium population values where all three species coexist.

1.2 Model and Methods

The script first establishes the Lotka-Volterra coefficients. The script then establishes the initial population values. The script then prints out the table headings and the first row of the table (the initial values). Next, the time-stepping parameters are set up. Finally, the script iterates through a for loop.

The for loop first calculates the next value for x, y, and z using the discretized governing equations:

$$x_{k+1} = x_k + \Delta t\left(1.2x_k\left(1-\frac{x_k}{4}\right)-1.3x_k y_k - 0.7\,x_k z_k\right)$$

$$y_{k+1} = y_k + \Delta t\left(y_k\left(1-\frac{y_k}{5}\right)-0.8x_k y_k - 1.2y_k\, z_k\right)$$

$$z_{k+1} = z_k + \Delta t\left(0.8z_k\left(1-\frac{z_k}{6}\right)-1.1x_k z_k - 0.9y_k\, z_k\right)$$

In code:

```
x_new = x_old + deltat*(a*x_old*(1-x_old/4) + b*x_old*y_old + c*x_old*z_old);
y_new = y_old + deltat*(d*y_old*(1-y_old/5) + e*x_old*y_old + f*y_old*z_old);
z_new = z_old + deltat*(g*z_old*(1-z_old/6) + h*x_old*z_old + i*y_old*z_old);
```

It then updates the old values to the ones just calculated. Whenever 0.5 units of time has passed, the population values are printed out using fprintf:

```
if (mod(k*deltat*2,1)==0)
        if (k*deltat == 10)
            fprintf('%.1f   %.2f   %.2f   %.2f\n', k*deltat, x_old, y_old,
z_old);
        else
            fprintf(' %.1f   %.2f   %.2f   %.2f\n', k*deltat, x_old, y_old,
z_old);
        end
    end
```

The case when the elapsed time is 10.0 is treated differently to maintain the table formatting. There are also the tic and toc functions at the beginning and end of the script to time it.

1.3 Results and Calculations

When the program is executed with initial values of x = 2, y = 2.5, and z = 3, the following output is printed to the screen:

Time	X	Y	Z
0.0	2.00	2.50	3.00
0.5	0.70	0.80	1.20
1.0	0.53	0.53	0.90
1.5	0.50	0.41	0.78
2.0	0.51	0.34	0.71
2.5	0.55	0.29	0.65
3.0	0.62	0.26	0.60
3.5	0.71	0.23	0.54
4.0	0.85	0.20	0.46
4.5	1.03	0.17	0.37
5.0	1.27	0.15	0.27
5.5	1.59	0.12	0.17
6.0	1.98	0.09	0.09
6.5	2.43	0.06	0.04
7.0	2.86	0.03	0.01
7.5	3.24	0.02	0.00
8.0	3.53	0.01	0.00
8.5	3.72	0.00	0.00
9.0	3.84	0.00	0.00
9.5	3.91	0.00	0.00
10.0	3.95	0.00	0.00

When the program is executed with initial values of x = 3, y = 2.5, and z = 2, the following output is printed to the screen:

Time	X	Y	Z
0.0	3.00	2.50	2.00
0.5	1.16	0.98	0.56
1.0	0.90	0.77	0.32
1.5	0.82	0.72	0.21
2.0	0.78	0.72	0.14
2.5	0.75	0.75	0.10
3.0	0.72	0.81	0.07
3.5	0.67	0.90	0.05
4.0	0.58	1.02	0.03
4.5	0.47	1.20	0.02
5.0	0.34	1.45	0.01
5.5	0.21	1.81	0.01
6.0	0.10	2.29	0.00
6.5	0.03	2.84	0.00
7.0	0.01	3.40	0.00
7.5	0.00	3.88	0.00
8.0	0.00	4.26	0.00
8.5	0.00	4.52	0.00

```
 9.0   0.00   4.70   0.00
 9.5   0.00   4.81   0.00
10.0   0.00   4.89   0.00
```

When the program is executed with initial values of x = 0.2, y = 0.5, and z = 0.6, the following output is printed to the screen:

```
Time      X      Y      Z
 0.0   0.20   0.50   0.60
 0.5   0.21   0.50   0.61
 1.0   0.21   0.50   0.62
 1.5   0.22   0.49   0.64
 2.0   0.22   0.48   0.65
 2.5   0.23   0.47   0.66
 3.0   0.24   0.45   0.67
 3.5   0.25   0.43   0.69
 4.0   0.26   0.41   0.71
 4.5   0.27   0.38   0.73
 5.0   0.29   0.34   0.75
 5.5   0.31   0.31   0.78
 6.0   0.34   0.27   0.81
 6.5   0.38   0.23   0.84
 7.0   0.42   0.19   0.87
 7.5   0.47   0.15   0.89
 8.0   0.53   0.12   0.89
 8.5   0.61   0.09   0.87
 9.0   0.71   0.07   0.83
 9.5   0.84   0.05   0.75
10.0   1.02   0.04   0.63
```

When the program is executed with initial values of x = 4, y = 5, and z = 5, the following output is printed to the screen:

```
Time      X      Y      Z
 0.0   4.00   5.00   5.00
 0.5   0.75   1.04   1.15
 1.0   0.52   0.70   0.79
 1.5   0.46   0.59   0.64
 2.0   0.44   0.53   0.56
 2.5   0.44   0.51   0.50
 3.0   0.46   0.50   0.45
 3.5   0.48   0.50   0.40
 4.0   0.51   0.52   0.35
 4.5   0.55   0.54   0.31
 5.0   0.58   0.56   0.26
 5.5   0.61   0.60   0.21
 6.0   0.63   0.65   0.16
 6.5   0.64   0.71   0.13
 7.0   0.63   0.79   0.09
 7.5   0.59   0.89   0.07
 8.0   0.52   1.03   0.05
 8.5   0.42   1.22   0.03
 9.0   0.30   1.50   0.02
 9.5   0.18   1.87   0.01
10.0   0.08   2.37   0.01
```

When the program is executed with initial values of x = 4, y = 3, and z = 2, the following output is printed to the screen:

Time	X	Y	Z
0.0	4.00	3.00	2.00
0.5	1.36	1.07	0.42
1.0	1.02	0.84	0.21
1.5	0.90	0.79	0.13
2.0	0.83	0.80	0.08
2.5	0.77	0.85	0.05
3.0	0.69	0.93	0.04
3.5	0.59	1.06	0.02
4.0	0.47	1.24	0.02
4.5	0.33	1.50	0.01
5.0	0.19	1.88	0.01
5.5	0.09	2.36	0.00
6.0	0.03	2.92	0.00
6.5	0.01	3.48	0.00
7.0	0.00	3.95	0.00
7.5	0.00	4.30	0.00
8.0	0.00	4.55	0.00
8.5	0.00	4.72	0.00
9.0	0.00	4.83	0.00
9.5	0.00	4.89	0.00
10.0	0.00	4.94	0.00

In the first two cases and the last case, one species crowds out the other two. In the third and fourth case, it appears that all three species are coexisting, but when looking at times past 10.0, it can be seen that one species ends up crowding out the other two.

1.4 Discussion

Through guessing and checking, I was not able to find initial population values that would result in all three species coexisting. However, such a point does exist and can be found by setting the continuous governing equations to 0 and solving the resulting system of equations. Using Wolfram Alpha, the three species are found to coexist at x = 493/2417, y = 1329/2417, z = 1464/2417.

When using the tic and toc commands to time the code, using a timestep of 0.01 results in an elapsed time of about 0.026 seconds. Using a larger timestep, like 0.25, resulted in a similar elapsed time. However, the values calculated using the forward Euler method went to negative infinity with the larger timestep. When using a smaller timestep the elapsed time increases (with a timestep of 0.00001, the elapsed time increased to 0.2 seconds).

2 The Pocket Change Problem

2.1 Introduction

The goal in this problem is to develop a program that will calculate the average number of coins you can expect to receive in your change after a cash transaction. This is accomplished by using a for loop to iterate through each possible value of change, and repeated floor and mod functions to find the least number of coins required to make each amount of change. The average is printed to the screen and it will be discussed how changing or removing coin denominations might change the average value.

2.2 Model and Methods

The script first establishes the values for each coin and initializes the counter variable to 0. The script then enters a for loop that iterates from 0 to 99. Each loop, a variable is set equal to the current iterator value.

```
remaining = i;
```

It is then seen how quarters could fit into that value using the floor function. The result is added to the count:

```
count = count + floor(remaining/q);
```

It is then found how much change is left that needs to be made up of lower denomination coins using the mod function:

```
remaining = mod(remaining, q);
```

The same process is repeated for each lower denomination coin (dimes, nickels, and pennies). After the for loop is complete, the average is calculated:

```
avg = count/100;
```

The result is then printed out using fprintf.

2.3 Results and Calculations

With denominations of 25, 10, 5, and 1, the following output is printed to the screen:

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

When eliminating the penny, the following output is printed to the screen:

```
Average Number of Coins = 2.70
```

Eliminating pennies from circulation results in a markedly lower average number of coins.

With denominations of 1, 5, 11, and 27, the following output is printed to the screen:

```
Average Number of Coins = 4.44
```

With denominations of 1, 7, 17, and 37, the following output is printed to the screen:

```
Average Number of Coins = 4.64
```

With denominations of 1, 3, 11, and 23, the following output is printed to the screen:

```
Average Number of Coins = 4.36
```

With denominations of 1, 3, 11, and 29, the following output is printed to the screen:

```
Average Number of Coins = 4.12
```

With denominations of 1, 3, 11, and 37, the following output is printed to the screen:

```
Average Number of Coins = 4.10
```

Changing the coin denominations to other values can result in a more optimal monetary system than the currently existing one.

2.4 Discussion

The average number of coins is much lower if we imagine eliminating pennies from circulation. This is because pennies act as a kind of filler. For all the values in between multiples of 5, the number of coins required is the number of coins required for the next lower multiple of 5 plus some number of pennies. This necessarily brings the average number of coins required up, since more coins are always needed, until reaching the next multiple of 5.

When leaving pennies alone and modifying the other three denominations, I could reduce the average number of coins to 4.10 using 1, 3, 11, and 37. I had a hunch that some combination of prime numbers would work best for this problem, and {1, 3, 11, 37} resulted in the lowest average in my testing while operating on this hunch. The disadvantage to this system is that it is much more difficult to think in terms of 3s, 11s, and 37s than it is to think in terms of 5s, 10s, and 25s. It is a rather unintuitive system if ease of use is also a goal for the monetary system.