M20 Homework 2

Colin Skinner - 505795313

2022/07/08

1 The Three Species Problem

1.1 Introduction

The goal in this problem is to evaluate a three species example of the Lotka–Volterra equations, a model of population growth in nature, by calculating each species populations per unit area. The code will integrate the functions to produce its output, and will display the population of all three species on a command line for each region $\Delta t = 0.005$ from $t = 0$ until $t = 12$, and a guess-and-check method will be used to find a possible balancing point of coexisting. Finally, the `tic` and `toc` command will be used to time the code for each iteration of Δt. All of these results will then be discussed.

1.2 Model and Theory

The population model used for this problem are the Lotka-Volterra Equations, also known as the predator-prey equations. Because there are three species, each equation considers all three variables, each with one exponential growth term and two predation terms.

$$\frac{dx}{dt} = 0.75x\left(1 - \frac{x}{20}\right) - 1.5xy - 0.5xz \tag{1}$$

$$\frac{dy}{dt} = y\left(1 - \frac{y}{25}\right) - 0.75xy - 1.25yz \tag{2}$$

$$\frac{dz}{dt} = 1.5z\left(1 - \frac{z}{30}\right) - xz - yz \tag{3}$$

Where

- $x,\ y,\ z$ = Population per unit area of Species X, Y, and Z respectively

While Equations 1-3 are the original equations, Equations 4-6 are the discretized variations, but to save space, their second term substitutes Equations 1-3.

$$x_{k+1} = x_k + \Delta t\left(\frac{dx}{dt}\right) \tag{4}$$

$$y_{k+1} = y_k + \Delta t\left(\frac{dy}{dt}\right) \tag{5}$$

$$z_{k+1} = z_k + \Delta t\left(\frac{dz}{dt}\right) \tag{6}$$

Where

- $x_k\ y_k,\ z_k$ = Population per unit area of Species X, Y, and Z at timestep k
- $x_{k+1},\ y_{k+1},\ z_{k+1}$ = Next iteration's population per unit area of Species X, Y, and Z at timestep $k+1$

1.3 Methods and Pseudo-code

The `MATLAB` script should carry out the following pseudo-code:

1. Set values for initial conditions
    - $x = 4.79,\ y = 2.49,\ z = 1.50$ at t=0

2. Set values for time slices (dt) and Final Time (tf)
    - $dt = 0.005,\ tf = 12.000$
3. Outputs initial header value
    - Looks like "`Time X Y Z`"
4. Begins timer of the for loop using "`tic`" command
5. Begin for loop from t=0 to tf (12.000) seconds, iterating by length of the time slice dt
    - If statement to evaluate whether or not t is divisible by 0.5, and if so, prints current time and population sizes to the command window ($x_0 = 4.79,\ y_0 = 2.49,\ z_0 = 1.50$)
6. Ends timer using "`tic`" command, and displays the time taken to run the for loop

1.4 Calculations and Results

Script output with default value of X,Y,Z equal to 4.79, 2.49, 1.50 respectively:

```
Which problem to test?
1
Time    X       Y       Z
0.0     4.79    2.49    1.50
0.5     1.98    0.88    0.37
1.0     1.50    0.63    0.23
1.5     1.31    0.54    0.18
2.0     1.21    0.49    0.16
2.5     1.16    0.47    0.15
3.0     1.13    0.46    0.14
3.5     1.10    0.45    0.13
4.0     1.09    0.45    0.13
4.5     1.07    0.45    0.13
5.0     1.05    0.46    0.13
5.5     1.02    0.47    0.12
6.0     0.99    0.49    0.13
6.5     0.94    0.52    0.13
7.0     0.87    0.55    0.13
7.5     0.78    0.61    0.14
8.0     0.67    0.69    0.14
8.5     0.53    0.81    0.16
9.0     0.37    1.00    0.17
9.5     0.22    1.30    0.17
10.0    0.10    1.77    0.16
10.5    0.03    2.51    0.11
11.0    0.00    3.67    0.05
11.5    0.00    5.43    0.01
12.0    0.00    7.82    0.00
Total time in loop:  0.0075 seconds
```

Modified script output with default values except Z is equal to 1.51 instead. This outputs nonzero populations for all species:

```
Which problem to test?
1
```

```
Time    X       Y       Z
0.0     4.79    2.49    1.51
0.5     1.99    0.88    0.38
1.0     1.50    0.63    0.23
1.5     1.31    0.53    0.18
2.0     1.22    0.49    0.16
2.5     1.17    0.46    0.15
3.0     1.15    0.45    0.14
3.5     1.13    0.44    0.13
4.0     1.13    0.43    0.13
4.5     1.13    0.43    0.12
5.0     1.13    0.43    0.12
5.5     1.13    0.42    0.12
6.0     1.14    0.42    0.11
6.5     1.16    0.42    0.11
7.0     1.17    0.41    0.10
7.5     1.20    0.41    0.10
8.0     1.23    0.40    0.09
8.5     1.27    0.39    0.09
9.0     1.33    0.37    0.08
9.5     1.42    0.34    0.07
10.0    1.54    0.31    0.06
10.5    1.73    0.27    0.05
11.0    2.01    0.21    0.04
11.5    2.43    0.15    0.02
12.0    3.05    0.09    0.01
Total time in loop:  0.0063 seconds
```

Modified script output with default values except Z is equal to 1.52 instead. X overshadows all other species:

```
Which problem to test?
1
Time    X       Y       Z
0.0     4.79    2.49    1.52
0.5     1.99    0.88    0.38
1.0     1.51    0.63    0.24
1.5     1.32    0.53    0.19
2.0     1.23    0.48    0.16
2.5     1.19    0.46    0.15
3.0     1.17    0.44    0.14
3.5     1.16    0.43    0.13
4.0     1.17    0.42    0.13
4.5     1.18    0.41    0.12
5.0     1.21    0.39    0.11
5.5     1.25    0.38    0.11
6.0     1.31    0.36    0.10
6.5     1.40    0.33    0.09
7.0     1.53    0.30    0.08
7.5     1.72    0.26    0.06
8.0     2.00    0.20    0.05
8.5     2.43    0.14    0.03
9.0     3.07    0.08    0.02
```

```
9.5    3.99   0.04   0.01
10.0   5.23   0.01   0.00
10.5   6.77   0.00   0.00
11.0   8.53   0.00   0.00
11.5   10.40  0.00   0.00
12.0   12.23  0.00   0.00
Total time in loop:  0.0063 seconds
```

Modified script output with default values except X is equal to 4.78 instead. Y overshadows all other species:

```
Which problem to test?
1
Time   X      Y      Z
0.0    4.78   2.49   1.50
0.5    1.98   0.88   0.37
1.0    1.49   0.63   0.23
1.5    1.30   0.54   0.18
2.0    1.20   0.50   0.16
2.5    1.14   0.48   0.15
3.0    1.10   0.47   0.14
3.5    1.07   0.46   0.14
4.0    1.04   0.47   0.13
4.5    1.01   0.48   0.13
5.0    0.97   0.50   0.13
5.5    0.92   0.52   0.14
6.0    0.85   0.56   0.14
6.5    0.75   0.62   0.15
7.0    0.64   0.70   0.16
7.5    0.50   0.83   0.18
8.0    0.35   1.02   0.19
8.5    0.20   1.31   0.20
9.0    0.09   1.76   0.18
9.5    0.03   2.48   0.13
10.0   0.00   3.60   0.06
10.5   0.00   5.32   0.01
11.0   0.00   7.67   0.00
11.5   0.00   10.55  0.00
12.0   0.00   13.65  0.00
Total time in loop:  0.0070 seconds
```

Elapsed time and final output with default values (4.79, 2.49, and 1.50) and a dt of 0.005 seconds:

```
Time   X      Y      Z
0.0    4.78   2.49   1.50
...    ...    ...    ...
12.0   0.00   7.82   0.00
Total time in loop:  0.0079 seconds
```

Elapsed time with default values and a new dt of 0.05 seconds:

```
Time   X      Y      Z
```

```
0.0     4.78    2.49    1.50
...     ...     ...     ...
12.0    0.10    1.95    0.08
Total time in loop:  0.0076 seconds
```

Elapsed time with default values and a new dt of 0.0005 seconds:

```
Time    X       Y       Z
0.0     4.78    2.49    1.50
...     ...     ...     ...
12.0    0.00    8.15    0.00
Total time in loop:  0.0088 seconds
```

Elapsed time with default values and a new dt of 0.0000005 seconds:

```
Time    X       Y       Z
0.0     4.78    2.49    1.50
...     ...     ...     ...
12.0    0.00    8.18    0.00
Total time in loop:  1.2706 seconds
```

1.5 Discussion and Conclusion

With the default starting numbers, only population Y remained. However, when the Z value was raised by just 0.01, the populations coexisted, but species Y and Z were both on a sharp downward curve. Additionally, when it was raised another 0.01 (to 1.52), only Species X remained. On the other hand, with other species at their default conditions, X was lowered just 0.01, but only Species Y remained. These results suggest that finding initial populations that allow all three species to coexist is challenging, and that these population equations are very fragile. One species having the greatest initial condition does not guarantee its survival.

When the value of dt was changed, the population values and the output of the `tic` and `toc` command both changed. After dt increased to 0.05—as the time slices become bigger—the three species coexisted, but when it was decreased to 0.0005, Species Y dominated and at $dt = 0.0000005$, it dominated even further.

Furthermore, when dt decreased to 0.0005, it took 0.0088 seconds, and when it was equal to 0.0000005, the code took 1.2706 seconds to execute. These results confirm the explanation that when time slices became smaller, the `tic` and `toc` command time increases because the code iterates more times over the loop to reach $t = 12$.

In order to find the population of the species at a moment in time, the differential Lotka-Volterra population equations must be integrated. When the discretized equations, which are only estimates, execute with a
smaller dt, the results seemed to approach an asymptote where $X = Z = 0$ and $Y = 8.18$, and these outputs make sense. As dt becomes smaller, the output values will approach the true integrated value; therefore, the instance when $dt = 0.0000005$ was the most accurate to nature.

2 The Pocket Change Problem

2.1 Introduction

The goal in this problem is to calculate the average number of coins received in change after a cash transaction. The code will calculate the minimum number of coins required for each amount of change from 0 cents to 99 cents, then the average by dividing the total number across all transactions by 100. The coins used will be US Standard coins (1, 5, 10, and 25 cent coins), and the average value will be printed to the command window. A further analysis will be conducted if pennies are removed from the equation altogether (reducing the possible values to multiples of 5). Finally, the values of each of the coins will be modified to determine if there is an optimal system that reduces the average number of coins per transaction.

2.2 Model and Theory

The following are the applicable and simple equations.

$$sum = sum + q + d + n + p \tag{7}$$

$$amtOwed = 25q + 10d + 5n + p \tag{8}$$

Where

- sum = Total number of coins across all transactions
- q, d, n, p = Number of quarters, dimes, nickels, or pennies in that transaction
- $amtOwed$ = Amount owed in each transaction

When quarters are taken from the $amtOwed$:

$$q = floor(amtOwed/25) \tag{9}$$

$$amtOwed = amtOwed - 25q \tag{10}$$

Where $amtOwed$ is decreased after the multiples of each coin are removed. For all other coins, Equations 9 and 10 will simple replace q with d, n, or p, and 25 will be replaced with the value for each coin (10, 5, and 1 respectively).

To average the coins across all transactions there is another simple equations:

$$avg = sum/100 \tag{11}$$

Where

- avg = Average number of coins across all transactions. What the script aims to answer.

2.3 Methods and Pseudo-code

The `MATLAB` script should carry out the following pseudo-code:

1. Assigns variable sum to 0
2. Begins for loop that iterates i from 0 to 99
   (a) Sets $amtOwed$ to i
   (b) q is set to the maximum whole number of quarters than can fit in $amtOwed$ with a floor function
   (c) Subtracts the total value of all quarters from $amtOwed$
   (d) Repeats steps (a)-(c) with dimes and nickels

(e) The number of pennies at the end of the loop is *amtOwed* (should be less than 5)

(f) Add total number of coins used for that transaction to *sum*

3. Calculates the average number of coins by dividing the value of sum by 100 and prints to command window

2.4 Calculations and Results

Default script output for average number of coins per transaction:

```
Which problem to test?
2
Average Number of Coins = 4.70
```

Script output if pennies are removed, and the loop iterates from 0 to 95 by increments of 5 (20 total iterations):

```
Which problem to test?
2
Average Number of Coins = 2.70
```

Script output if coin values are adjusted to find an optimal monetary system. Quarters, dimes, nickels, and pennies are worth 27, 11, 5, and 1 respectively:

```
Which problem to test?
2
Average Number of Coins = 4.44
```

Another script output if coin values are adjusted to find an optimal monetary system. Quarters, dimes, nickels, and pennies are worth 29, 11, 3, and 1 respectively:

```
Which problem to test?
2
Average Number of Coins = 4.12
```

2.5 Discussion and Conclusion

The output showed that the average number of coins for transactions between 0 and 99 cents with US Standard coin values is 4.70 coins.

When pennies are removed, the amount shrinks to 2.70 coins on average. Because the penny is worth such an insignificant amount compared to the next biggest coin, the nickel, it has to increase to a value of 5 to begin counting nickels. Meanwhile, in the penniless scenario, the second smallest coin is only twice the smaller one, so the nickel only needs to increase to 2 before the dime begins counting.

This can also be proven two ways. First, the script is edited so that the loop iterates from 0 to 95 in increments of 5 (resulting in 20 calculations). Pennies are removed, and the calculation for the average number of coins divides the sum by 20 instead of 100. The second method uses the original script but sets the number of pennies equal to 0 for each of the 100 loops. Because pennies need to be 5 to "graduate" to nickels, each separate combination of dimes, nickels, and quarters appears 5 times consecutively. According to the first method, there are 20 different calculations from 0:5:95, so dividing by the initial 100 will account for adding each coin combination 5 times.

Another way that the average number of coins can be reduced is by changing the values of the coins themselves. The first trial was conducted with quarters and dimes changed to 27 and 11 percent. This resulted in 4.44 coins on average. Then, guess and check yielded a more optimal system of 29, 11, 3, and 1 from greatest to least, which resulted in 4.12 coins on average. However, this system is not ideal because when prime numbers are introduced, the coin values are not all multiples of each other, which makes the system harder to memorize or conceptualize. Currently, there is a coin shortage, which makes having an optimal system more important, but there are sacrifices that must be made for the sake of practicality.