M20 Homework 2

Colin Skinner - 505795313

2022/07/15

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 diagram and equations are applicable for modeling the pendulum:

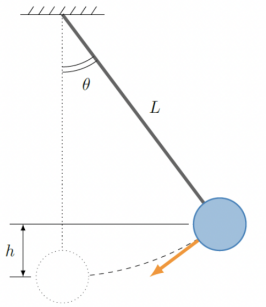


Figure 1: Swinging Pendulum

$$\frac{d\omega}{dt} = -\frac{g}{L} sin\,(\theta) = \alpha \tag{1}$$

$$\frac{d\theta}{dt} = \omega \tag{2}$$

Where

- g = Acceleration due to gravity, $9.81 m/s^2$
- L = Length of pendulum
- h = Height from equilibrium
- θ = Angular position of pendulum relative to the horizontal axis
- ω = Angular velocity of pendulum
- α = Angular acceleration of pendulum

To integrate these functions, both Forward and Backward Euler Methods are used. While Equations 1-2 are the original equations, Equations 3-4 are the explicit discretized equations. Equation 5 is the implicit discretized equation of θ_{k+1} considers ω_{k+1} instead of ω_k.

$$\omega_{k+1} = \omega_k + \Delta t\,(\theta_k) \tag{3}$$
$$\theta_{k+1} = \theta_k + \Delta t\,(\omega_k) \tag{4}$$
$$\theta_{k+1} = \theta_k + \Delta t\,(\omega_{k+1}) \tag{5}$$

Where

- $x_{k+1},\ y_{k+1},\ z_{k+1}$ = Next iteration's population per unit area of angle and angular velocity at timestep $k+1$

When calculating the integrated values of the differential equations, two methods were used. In the first, both equations used the Forward Euler Method. In the second, ω used Forward Euler but θ used Backward Euler.

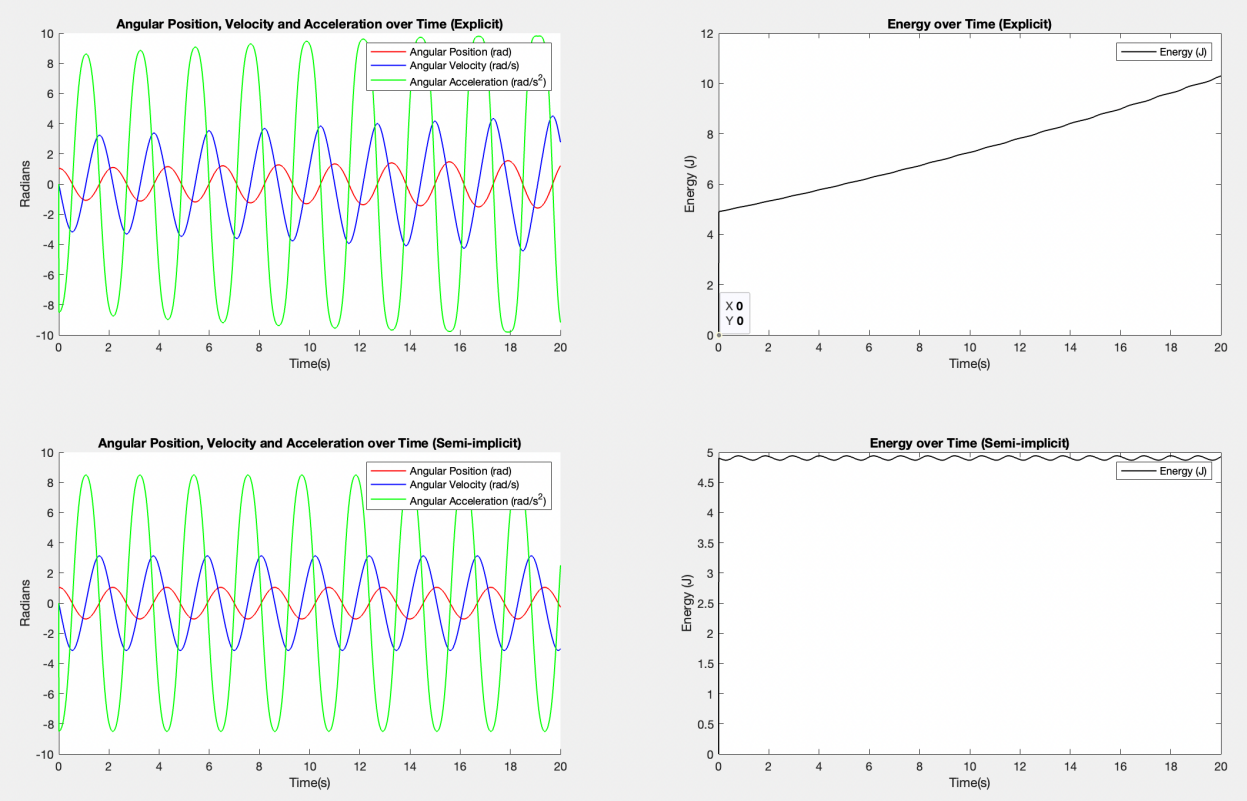
1.3 Methods and Pseudo-code

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

1. Set values for time slices (dt) and Final Time (tf), and creates iterations variable
   - $dt = 0.005,\ tf = 20.000,\ iter = tf/dt$
2. Create arrays for θ, ω, α, $Energy$, and $time$
   - Length from 1 to $iter$ to store 1 value for each time slice
3. Set values for initial conditions at t=0
   - $theta(1) = \pi/3,\ omega(1) = 0,\ alpha(1) = 0$
   - $L = 1,\ g = 9.81$
4. Begin for loop from k=1 to $iter$ to go through each array (This is the Explicit Forward Method)
   (a) Calculate new values for angular position, velocity, and acceleration using Equations 3, 4, and 1 (the last for angular acceleration, which is the derivative of angular velocity)
   (b) Calculate energy by considering the height, given by $L(1 - cos\theta)$
5. Graph a plot of angular position, velocity, and acceleration over time
6. Graph energy over time
7. Repeat steps 4-6, but using Equation 5 for the calculation of angular position (Implicit)

1.4 Calculations and Results

Script output with $L = 1$ and $g = 9.81$:



1.5 Discussion and Conclusion

Forward Euler did not conserve the energy over time. It continued on an upward trend for all of the time interval. Visually, the angular position, velocity, and acceleration all increase on the first graph. Taking a smaller time slice of $t = 0.005$ yielded an energy increase of around 0.5 Joules instead of almost 5 Joules when $t = 0.05$. At $t = 0.0005$ this change becomes nearly indistinguishable on the Energy vs. Time graph. With the semi-implicit method, the energy was conserved overall. Although it oscillates, its average stays close to the actual energy over time. With a smaller time slice of $t = 0.005$, the sine wave becomes smaller in amplitude, with its shape more akin to a like. At $t = 0.0005$, it becomes visually indistinguishable from a line.

2 DNA Analysis

2.1 Introduction

The goal in this problem is to evaluate a DNA strand to reveal protein-coding segments. The code will iterate through each codon (three-base segments) in the strand to find start and ending codons. The total number of the protein-coding segments, along with their average length, minimum length, and maximum length of the will all be calculated from the data gathered. A further analysis will uncover the percentage of the codons were used in protein-coding, compare the frequencies of each of the three stop codons, and consider mutations in stop codons.

2.2 Model and Theory

DNA strands are a 1D representation of genetic code that have four main nucleotide bases: A, C, G, T. Embedded in them are the building blocks for an entire living organism, based off of specific protein-coding segments of the DNA organized in groups of three bases called a codon. Each protein-coding segment starts with a specific codon ATG, and end with either TAA, TAG, or TGA. The length of each segment includes the start and end codon, and intermediate codons between when a segment is ended and when another begins are essentially disregarded.

2.3 Methods and Pseudo-code

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

1. Load DNA vector (chr1_sect.mat) into *dna* and sets *numBases* equal to its length
2. Set:
   - *startPoint* to -1 (no start point identified)
   - *maxLength* to 0 (increases for every maximum found), and *minLength* to the length of the dna vector (decreases for every minimum found)
   - Total combined length of all proten-coding segments (*totalLength*) to 0, and a counter with the number of those segments (*totalProteins*) to 0
   - 3 Counters for the number of times TAA, TAG, and TGA end a protein-coding segment
3. In a for loop that iterates variable k from 1 to $\underline{numBases - 2}$ (the start of the last codon) by 3
   (a) If start point is not identified (equal to -1)
      - If codon is the start codon ("ATG")
        - Set start point equal to k
      - If not, then it goes back to start of for loop
   (b) Else if start point is a stop codon ("TAA," "TAG," or "TGA")
      - Iterate counter for the stop codon that was detected
      - Set *length* of protein-coding codon to $k - startPoint + 3$
      - If *length* is larger than *maxLength*, set *maxLength* to *length*
      - If *length* is smaller than *minLength*, set *minLength* to *length*
      - Add *length* to *totalLength*
      - Reset *startPoint* to -1
      - Increments *totalProteins*
4. Set *mostUsedStop* to stop codon with max value
5. *averageLength* is equal to *totalLength*/*totalProteins*
6. Display total protein segments, average length, max and min value

7. Calculate and print percent of DNA used ($numBases - totalLength$)

8. Print most used stop codon

2.4 Calculations and Results

Script output for chr1_sect.mat vector:

```
Total Protein-Coding Segments:  4339
Average Length:  87.69
Maximum Length:  1797
Minimum Length:  6
Percent of DNA used in protein-coding process:  30.16
Stop codon used most:  TGA with 1897 uses.
```

2.5 Discussion and Conclusion

By analyzing the long vector of DNA, the code determined the number of protein-coding segments of 4339; average length of 87.69, maximum length of 1797, and minimum length of 6 codons; percentage of DNA used in protein-coding process of 30.16; and the stop codon with the most uses: TGA with 1897 uses. The percentage of DNA used in the protein-coding process is around 1/3, which seems slightly high, and the stop codon usage is over 40%, which is higher than 1/3 if all the codons were used equally.

A mutation would change the code slightly with the if statements for ending and beginning each segment. If the codon has two elements in common with ATG for starting the segment, or with TAA, TAG, or TGA for ending the segment then it would be considered a mutation but still a start or ending codon.