Cassandra Cantu

305100205

CAE/MAE M20 Summer A 2020

July 14th 2020

**Homework 03**

**1 The Pendulum Physics Problem**

* 1. **Introduction**

The goal of this script is to develop a program that calculated the angular position, velocity, and acceleration and total energy over time for a simple pendulum. To do this, the continuous functions for the kinematic vectors and total energy have to approximated to functions the program can work with. The two methods used to discretize the functions are the forward (explicit) Euler method and the semi-implicit Euler method. Any discrepancies between the kinematic vectors and total energy over time calculated with the forward Euler method versus with the semi-implicit Euler method will be discussed.

* 1. **Model and Theory**

Simple pendulum motion can be described by two separate differential equations. To discretize, we first applied the forward (explicit) Euler method to both equations. After, we used the semi-implicit Euler method by calculating one of the equations with the forward Euler method and the other with the implicit Euler method. We then calculated the total energy of the simple pendulum with the results of each method.

*Simple pendulum motion*

*Forward (explicit) Euler method*

Semi-Implicit Euler method

*Total energy of simple pendulum*

Where

= angular acceleration

= angular velocity

= angular position

= acceleration due to gravity

= length of pendulum [m]

= total energy of simple pendulum

= height of pendulum from rest

* 1. **Methods and Pseudocode**
* Set up initial conditions for L, g, and theta\_0 with given values.
* Set up time conditions so that the initial time is 0, the final time is 20s, and the in the above equations is 0.005s. Define the time array.
  + Use these values to calculate the number of values in the time array.
* Initialize theta, omega, and alpha arrays with zeroes and define the first element in theta array as theta\_0.
* Inside a for loop, calculate angular position, velocity, and acceleration over time using the forward Euler method.
  + The for loop starts at index 1 with step size 1 and continues until the number of values in the time array minus 1 since we do not need values after
* Outside the for loop, calculate the total energy based off the forward Euler method calculations.
* Plot the kinematic vectors vs. time found with forward Euler method on the same graph
  + Use titles, useful bounds, and legend for legibility and clarity
* Plot the total energy vs. time found with the forward Euler method on a separate graph
* Inside a for loop, calculate angular position, velocity, acceleration, and total energy over time using the semi-implicit Euler method
* Outside the for loop, calculate the total energy based off the semi-implicit Euler method calculations.
* Plot the kinematic vectors vs. time found with semi-implicit Euler method on the same graph
* Plot the total energy vs. time found with the semi-implicit Euler method on a separate graph
  1. **Calculations and Results**

When the script is run, the output is as follows: **A picture containing screenshot

Description automatically generated**

**A picture containing screenshot

Description automatically generatedA picture containing screenshot

Description automatically generatedA screenshot of a cell phone

Description automatically generated**

Using a smaller time-step size (), the output for the forward Euler method calculations is:A screenshot of a cell phone

Description automatically generated

**A screenshot of a social media post

Description automatically generated**

* 1. **Discussion and Conclusions**

In both Figures 1 and 2, the magnitudes of the dependent variables are increasing over time. In Figures 3 and 4, however, the dependent variables do not go past their respective minimum and maximum values. This discrepancy is the result of the Forward Euler method. Since original two equations are coupled, the theta and omega variables are dependent on each other. Thus, the forward Euler method, which relies on the previous result of the other, introduced artificial energy into the system, giving us inaccurate results. Using a smaller time-step size, such as 0.0005, reduces this error as the approximation will follow the continuous curves more closely. Nevertheless, energy is not conserved, but rather, increased overall. In contrast, the semi-implicit Euler method conserves energy because

1. **DNA Analysis**
   1. **Introduction**

The goal of this script is to develop a program that calculates the total length of each protein-coding segment as well as the mean, maximum, and minimum lengths given a DNA sequence. The program will first examine the DNA sequence for the start codon and will continue looking until it reaches a stop codon. It then stores these values in two separate arrays and subtracts them to find the length. The program continues until the whole sequence is examined. Additionally, the percentage of DNA used for protein-coding and the most/least used stop codons were found.

* 1. **Model and Theory**
  2. **Methods and Pseudocode**
* Load given DNA file and determine number of bases from it using MATLAB function length() that counts the number of elements in an array
* Initialize the arrays startPoint and endPoint and number of protein segments (array index) to 0
* Initialize stop codon counters to 0.
* Use for loop to look through DNA sequence. We start at position 1 and have a step size of 3 to examine codons. We continue through the DNA sequence until the whole length minus 2 since the last two bases on their own cannot form proteins.
* First look for the starting point with if statement with condition startPoint == 0.
* If the specific codon examined matches the start codon, the number of protein segments is updated, and the position of the starting point is saved in an array called savedStart.
* After the startPoint is established, look for endPoint with if/elseif statements nestled in an else statement. If the specific codon examined matches one of the three stop codons, the position of the ending point is saved in an array called savedEnd.
  + The count for that specific stop codon is then updated.
* Calculate the length of the protein-coding segments and store in a new array. To do this, we subtract the saved starting points from the saved end points and then add 3 to account for the stop codon’s length.
  + savedEnd – savedStart + 3
* After, calculate total protein-coding segments, average length, and minimum/maximum lengths using MATLAB’s built-in functions: length(array), mean(array), min(array), max(array)
* Calculate the percent of DNA directly used in protein-coding process by adding all the elements in the array with lengths and dividing that by the number of bases. Lastly, multiply by 100 to get a percentage.
* Calculate most and least frequently used stop codons with if/else statements that compare each stop codon’s counter to each other.
* Print output to the command window
  1. **Calculations and Results**

When the script is run, the output for the given file is:

Total Protein-Coding Segments: 4339

Average Length: 87.69 bases

Maximum Length: 1797 bases

Minimum Length: 6 bases

Percentage of DNA Used: 30.16%

The most frequently used stop codon is: TGA

The least frequently used stop codon is: TAG

* 1. **Discussion and Conclusions**

In the given DNA file, the percent of DNA directly used in the protein-coding process is 30.16%. The most frequently used stop codon is TGA while the least frequently used stop codon is TAG. To account for the effect of replacement mutations, I would need to extend my algorithm to identify both perfect start and stop codons as well as start and stop codons with a single mutated middle base. For the start codon for example, I would look if dna(k) == 4 && dna(k+2) == 1. Then, I would nestle if/elseif statements inside that if statement and look at dna(k+1), the middle base, if it is correct or mutated. Inside the if/elseif statements for the mutated bases, I would have a counter that will update when that statement is true. This strategy can be extended to find mutated middle bases in the stop codons.

Example: start codon (143)

if startPoint == 0

if dna(k) == 1 && dna(k+2) == 3

**%check if perfect start codon**

if dna(k+1) == 4

(keep this part the same as in hw code)

**%check for mutated middle base**

elseif dna(k+1) == 1

(any calculations that you want to find here)

AAG\_count = AAG\_count + 1;

elseif dna(k+1) == 2

(any calculations that you want to find here)

ACG\_count = ACG\_count + 1;

elseif dna(k+1) == 3

(any calculations that you want to find here)

AGG\_count = AGG\_count + 1;