Jeffrey Ding
UID: 104928991
CEE M20
April 27th, 2018

# Homework 4

## 1. The Pendulum Problem

### 1.1 Introduction

The goal of this problem is to model the motion and total energy of a simple pendulum as a function of time.

### 1.2 Model and Methods

The script begins by setting the constants of the pendulum, which would be the length and gravitational acceleration constant. Then, the script creates an array to represent time from t=0 to t=20 using this line of code:

```
t = linspace(0,20,4001);
```

and three arrays full of zeros using these lines of code:

```
theta = zeros(1, 4001);
w = zeros(1, 4001);
a = zeros(1, 4001);
```

representing position, angular velocity, and angular acceleration, respectively. Then, the script adds (-g/L) to all values in the a array and sets the first values of theta and a arrays using the following lines of code:

```
a = a + (-g/L);
theta(1) = pi/3;
a(1) = a(1) * sin(theta(1));
```

Then, the script uses the for loop like this one:

```
for i = 2:4001
end
```

and runs the following calculations:

```
w(i) = w(i-1) + a(i-1)*0.005;
```

```
theta(i) = theta(i-1) + w(i)*0.005;
a(i) = a(i) * sin(theta(i));
```

over and over again in order to simulation semi-implicit euler’s method. Then, I calculate the total energy using the following lines of code:

```
h = (L*cos(theta)) - L*cos(pi/3);
E = g*h + 0.5*(L*w).^2;
```

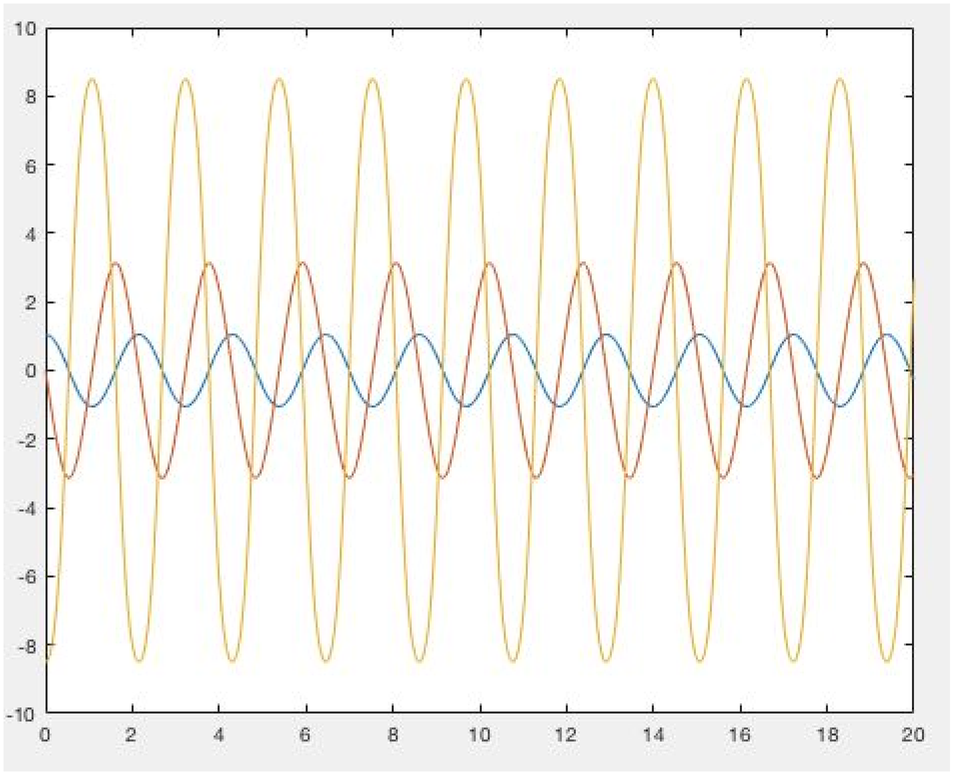
Which creates an array of heights using the law of sines and an array that represents total energy by using all the values in the angular velocity array to find kinetic energy and adding whatever the the potential energy using the height array.

Then, the script graphs the the kinematics of the pendulum on one graph and the total energy of the pendulum on another graph using the following lines of code:

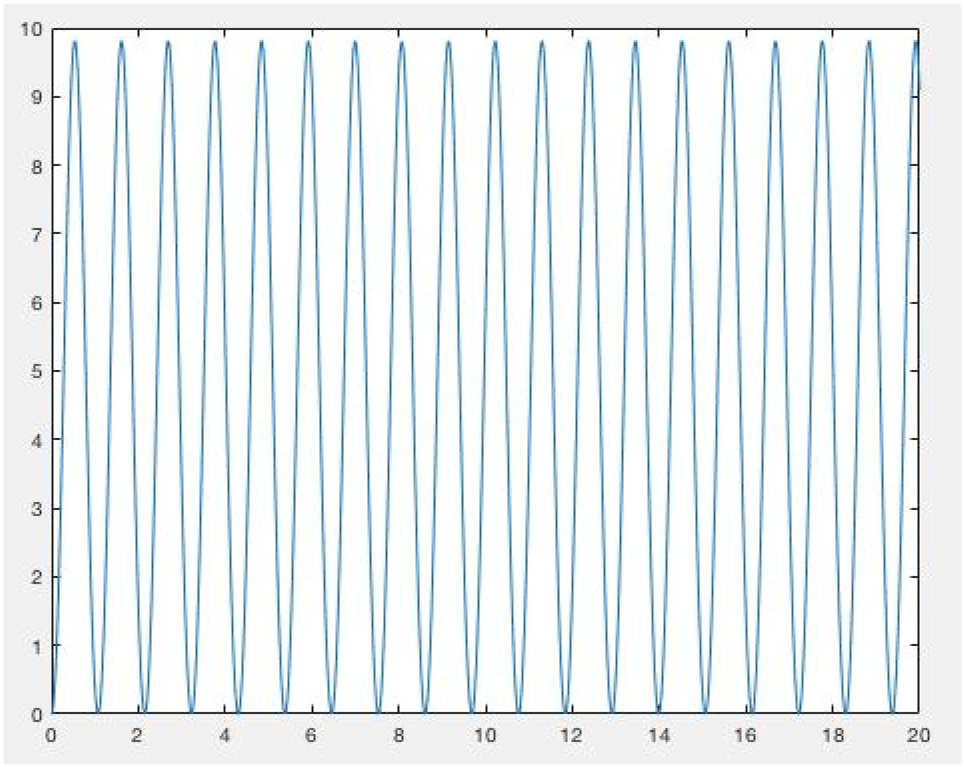
```
figure(1);
plot(t, theta, t, w, t, a);
figure(2);
plot(t, E);
```

**1.3 Calculations and Results**

When the program is executed, the graphs are created:



10
8
6
4
2
0
-2
-4
-6
-8
-10
0
2
4
6
8
10
12
14
16
18
20



10
9
8
7
6
5
4
3
2
1
0
0
2
4
6
8
10
12
14
16
18
20

The output shows the both kinematics and energy are conserved.

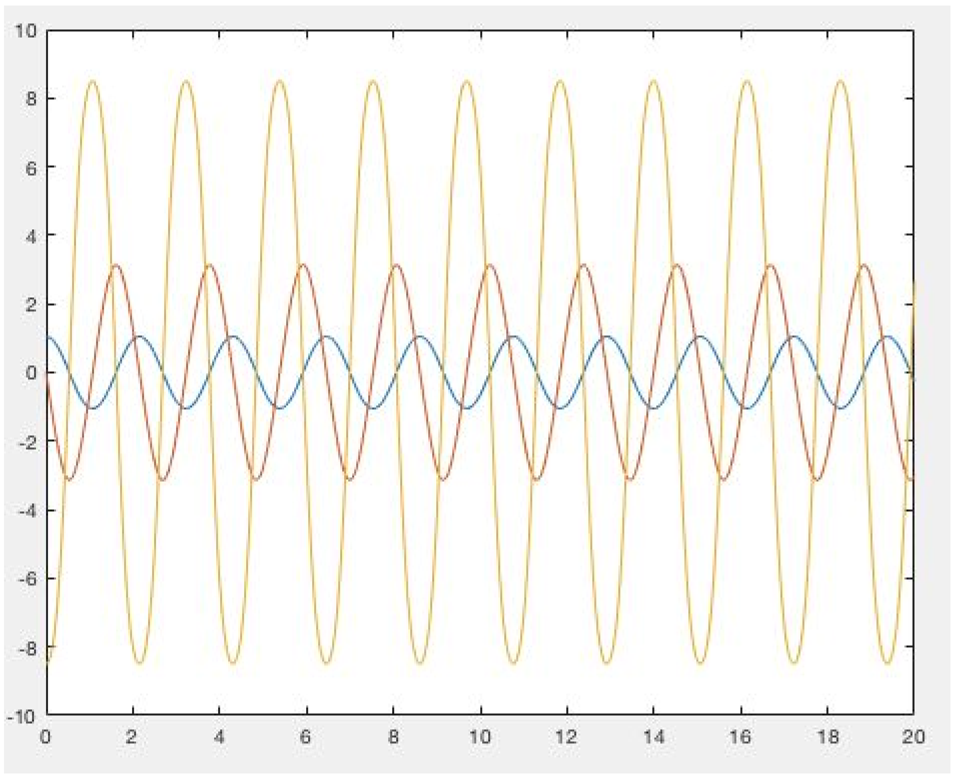
**1.4 Discussion**

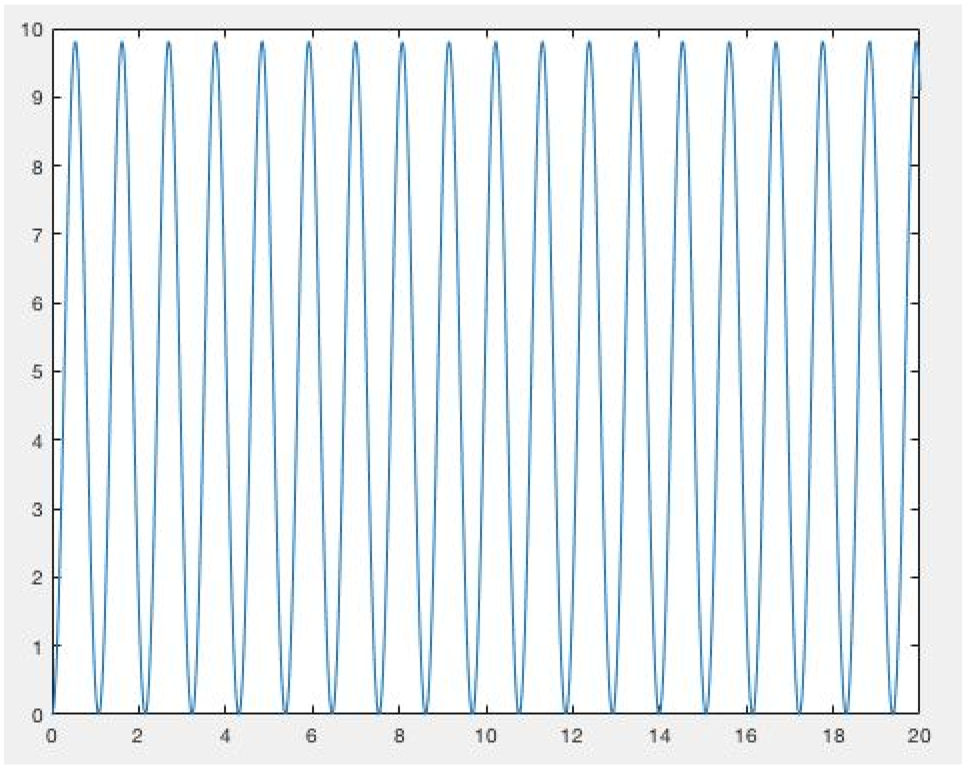
The output graphs show that in all variables, they oscillate as a function of time. Position, velocity, and acceleration simply oscillate between two values, showing that no energy leaves the systems. This is further supported by the energy graph, which also oscillates. This means that over time, the pendulum conserves all energy, which makes sense because we are neglecting the weight of the connecting rod, friction at the hinges, and air resistance.

Upon repeating this experiment using the implicit euler's method, I found that the values of all variables as a function of time oscillates towards zero. This is due to excessive damping, which slowly erodes the energy, which is probably more realistic because in real life, energy in a pendulum is transformed into other forms of energy.

Using the forward Euler's method does not conserve energy. In fact, it adds energy into the system over time, which is definitely not what we want. Using smaller timesteps doesn't alleviate the problem, it just makes the increase smaller. However, an semi-implicit method conserves energy fully because it uses the previous steps acceleration and velocity to calculate the current velocity.

Semi-implicit Euler's


10
8
6
4
2
0
-2
-4
-6
-8
-10
0
2
4
6
8
10
12
14
16
18
20



Note: the first way I solved this problem was semi-implicit euler's on Tuesday and I didn't know there was another way until Thursday, my bad for doing this problem wrong.

## 2. DNA Analysis

### 2.1 Introduction

The goal of this problem is to calculate the lengths of protein-coding segments in a DNA segment.

### 2.2 Model and Methods

The script begins by loading  the DNA sequence using the following code:

```
load('chr1_sect.mat');
```

Then, the script creates an array full of zeros that will hold the length of the DNA segments. In order to account for all possible outcomes, the length of the array is set to the total number of bases in the DNA. The following lines of code reflect this:

```
numBases = length(dna);
lengthArray = zeros(1, numBases);
```

Then, the script sets a lengthIndex variable that will indicate how many DNA segments found and a startPoint variable that stores the index of the start of a DNA segment.

Then, the script runs a for loop that checks, starting from the 1st index, every third base until it finds a start tag. This following code show this:

```
for k = 1:3:numBases-2
        if dna(k) == 1 && dna(k+1) == 4 && dna(k+2) == 3
        …
        end
end
```

When the script finds a start tag, the startPoint variable is set to the current index and it runs another for loop trying to find an end tag:

```
for i = startPoint+3:3:numBases-2
        if (dna(i) == 4 && dna(i+1) == 1 && dna(i+2) == 3) || (dna(i) == 4 && dna(i+1) ==
        1 && dna(i+2) == 1) || (dna(i) == 4 && dna(i+1) == 3 && dna(i+2) == 1)
        ...
        end
end
```

Within the loop, the script would add one to lengthIndex to represent one DNA segment found, then then putting a length value into the lengthArray. Afterwards, the script continues searching for the next DNA segment and resume the first for loop using this code:

```
lengthIndex = lengthIndex+1;
lengthArray(lengthIndex) = i - startPoint + 3;
k = i;
```

Then, the script creates a newLengthArray that holds all the DNA segment lengths in order to properly calculate the statistics desired. The following code does this:

```
newLengthArray = zeros(1, lengthIndex);
for x = 1:lengthIndex
   newLengthArray(x) = lengthArray(x);
end
```

Finally, the script prints out the data using the following print statements:

```
fprintf('Total Protein-Coding Segments: %.0f\n', length(newLengthArray));
fprintf('Average Length: %.2f\n', mean(newLengthArray));
fprintf('Maximum Length: %.0f\n', max(newLengthArray));
fprintf('Minimum Length: %i\n', min(newLengthArray));
```

**2.3 Calculations and Results**

When the program is executed the following output is printed to the screen:

```
Total Protein-Coding Segments: 6297
Average Length: 92.17
Maximum Length: 1149
Minimum Length: 6
```

The maximum and minimum lengths make sense because both values are divisible by 3.

**2.4 Discussion**

The output shows that they are 6297 DNA segments of interest. The average length of these segments is 92.17, the maximum length of a DNA segment was 1149 and the minimum length of one was 6. The maximum and minimum lengths make sense because both values are divisible by 3 and the minimum length is greater than 3.

By multiplying total segments and average length, there are approximately 580394.49 DNA bases in use. This is out of the original 1261563 number of bases, this means that approximately 46% of the DNA is directly used in the DNA coding process.

The most frequently used stop codon was TGA. The least frequently used stop codon was TAG.

If I were to modify my algorithm to allow the starting point of a protein-coding segment to be updated such sequences such as ATG-ATG-TAG would be treated as two strains, we would simply remove the :

```
k = i;
```

At the end of the for loop when an end tag is found so that the script starts searching for a new start tag immediately after the current start tag.