Mauricio Deguchi
UID: 405582562
CEE/MAE 20
October 26, 2021

Homework 4

1 Resonance problem: Mass-Spring System

1.1 Introduction

This problem asks us to model an oscillating spring-mass system. Through this script, we want to plot the position of the cart versus time as an external force acts on the system in a sinusoidal fashion. This models a real world situation with a simple mass-spring system and will help lay a foundation for other similar differential equation problems in the coming years.

1.2 Models and Methods

The script first declares all of the constants which were defined in the problem statement. These constants can then be used to define our semi-implicit Euler method. The fundamental equations for this method are:

$$v_{k+1} = v_k + \frac{f_c \cos(\omega * k) - k_{spring} x_k}{m} dt$$

$$x_{k+1} = x_k + v_{k+1} dt$$

This method allows us to approximate our next position using our next velocity and current position. This approximation ensures that we are not arbitrarily adding energy into our system in the way that the explicit Euler method would. These equations are implemented into the script within a for loop, which runs this sequence for the specified time in the problem statement (t = 0s to t = 80s in time steps of 0.01s). The for loop runs the calculations for v_{k+1}, and then stores x_k into an array with it's corresponding time position 'k' through the following line:

```
x(k+1) = x_k;
```

the value x_k is the horizontal position of the mass one time step behind our current time step. This is because our current time step will calculate a new value for x in line following this one. This loop will continue to run for our given time interval, storing all of the values of x into an array. This array can then be used to create our plot with a corresponding time array using the `linspace` function.

1.3 Calculations and Results

The for loop is run in two separate occasions. Both for loops are identical with only a single difference between them. In between the two for loops, we plot our first set of data and then change our value of ω as stated in the problem statement. We then run the loop again and using the `hold on` function we plot our second set of data on top of the existing plot creating the following

graph: The premise of our calculations lies in using our next velocity to predict where our next position will lie. This is done by first calculating our current velocity and then using the slope formula with this velocity to find our current position from our old position

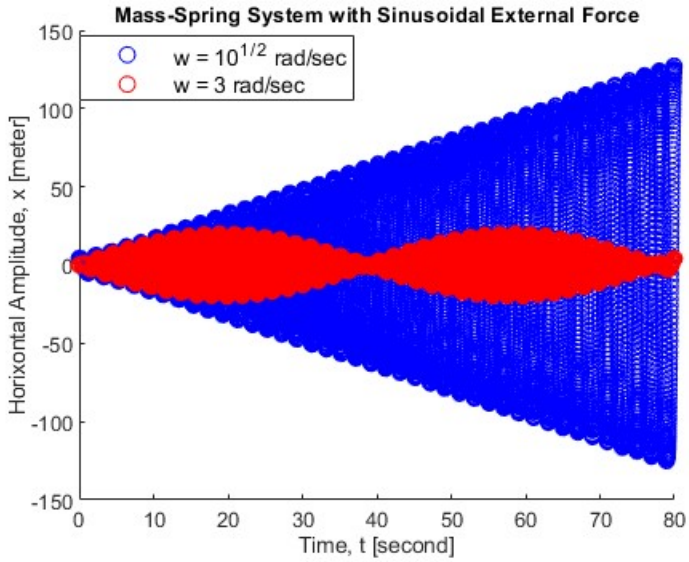

Figure 1. Euler's Semi-Implicit Forward step method for a mass on a spring without friction

1.4 Discussion

The position versus time graph shown above displays the effect of varying the angular velocity of the external force. My assumption for what is happening in the second scenario is that the external force acts in resonance with the spring-mass system, leading it to increase in energy over the course of each cycle. This increase in energy can be seen by an ever-increasing amplitude for our second plot. One interesting challenge for this problem was plotting. At first, I simply plotted the data for each trial one point at a time (i.e. we calculate the position and plot it immediately so the plot command was within the for loop) and this was extremely time consuming. Which makes sense because we are plotting 8000 points individually. The use of an array greatly increased the speed and efficiency of the program which will be a useful tool in future endeavors.

2 Resonance problem: Mass-Spring-Damper System

2.1 Introduction

This problem uses a near identical model to the previous situation with the same semi-implicit Euler method to approximate the position of a mass-spring system. The key difference in this problem is the introduction of a damper which dissipates energy from the system. This will be introduced into our equations with a dependence on velocity.

2.2 Models and Methods

Here, we will use the same array and for loop methods as used in problem 1. However, we will have to make a slight modification to our original equation, leading to the following:

$$v_{k+1} = v_k + \frac{f_c \cos(\omega * k) - k_{spring} x_k - c v_k}{m} dt$$

$$x_{k+1} = x_k + v_{k+1} dt$$

Again, these equations are ran through a for loop which first calculates our next velocity and uses it to predict our next position. These values are stored into an array similar to the previous one which is plotted immediately after the for loop ends. We then change our value of ω and run the for loop again, plotting our data together on a single graph.

2.3 Calculations and Results

Aside from the introduction of the damping force, we also changed our time scale, so our new time scale runs the program through t = 0s to t = 50s in time intervals of 0.01s. In order to achieve this, our for loop runs from 0 to 4999 and we apply a division of 100 to the inside of the cosine function to accurately acquire our time. We also use the linspace function to create our time array through the following line:

```
t = linspace(0,50,5000);
```

This array will have the same size as our number of points, allowing us to plot our various data sets into the following graph:

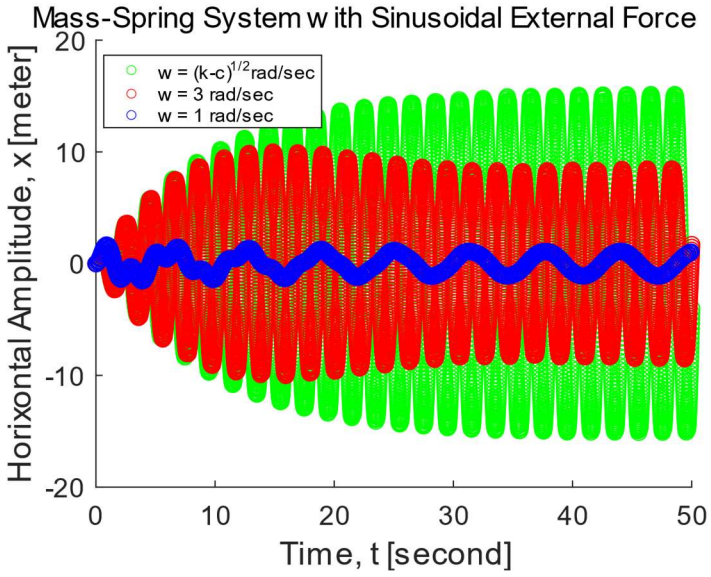

Figure 2. Euler's Semi-Implicit Forward step method for a mass on a spring with friction

2.4 Discussion

This plot created three very distinct data trends. First off, for the red one with $\omega = 3\ rad/sec$, we have small oscillations to begin with which grow quickly and then stabilize to a constant pattern after some time. This appears to be a scenario where the external force begins driving the spring mechanism which overshoots the amplitude to begin with; this is then corrected with the damping force and remains in constant, steady oscillation. In the second scenario where $\omega = 1\ rad/sec$, the blue data points appear to overshoot excessively at first and this is gradually corrected over time into a steady oscillation where towards the end we can see an nearly harmonic pattern. Lastly, in our green data set, we can see that the angular frequency of the external force is minimal, leading to small oscillations which cannot resonate with one another leading to gradually dying waves as time goes on.

3 Resonance Problem: Implicit Euler

3.1 Introduction

Unlike the prior problems, where algebraic manipulation allowed us to perform calculations based off our next speed and current position, here the Implicit Euler method is used. Through Implicit Euler, we can calculate our next velocity and next position simultaneously using the Jacobian of the discretized functions. Again, here we include the damping due to friction as well as the sinusoidal external force.

3.2 Models and Methods

Through most of this problem, we are able to follow the pseudocode provided in the problem statement. For starters, we establish our constants and dedicate an array for time, position, and velocity so that we have already preoccupied the amount of storage necessary. We then move into a for loop, where we follow a similar method to the prior two problems by establishing our time loop for our start time to our final time divided by the step size. In order to counteract this division, we then also divide by the same factor whenever the constant comes into our discretized equations. Unlike our prior methods, following the for loop, we introduce a while loop, where we conduct our calculations as well as an error for those calculations at every step in the for loop. If our error is not small enough, we continue to run through the while loop within the same step in the for loop. This ensures that our values at every step in the for loop are calculated to a preset tolerance. The cycle is then repeated for every step in the for loop and we can then plot our position and time arrays.

3.3 Calculations and Results

As previously stated, most of the calculations follow directly from the problem statement pseudocode. We first establish our velocity and time arrays into a single concatenated array. This array 'q' can then by indexed for any velocity and position value, and reassign them. The discretized functions are provided and are directly implemented as a 2x1 array. This array is then pushed through the \ calculation alongside the Jacobian to find our change in both position and velocity. The changes are implement and we calculate our error to see if we can move on to the

next step in the for loop. Once all of our steps have been completed, we plot our time and position arrays, producing the graph in Figure 3.

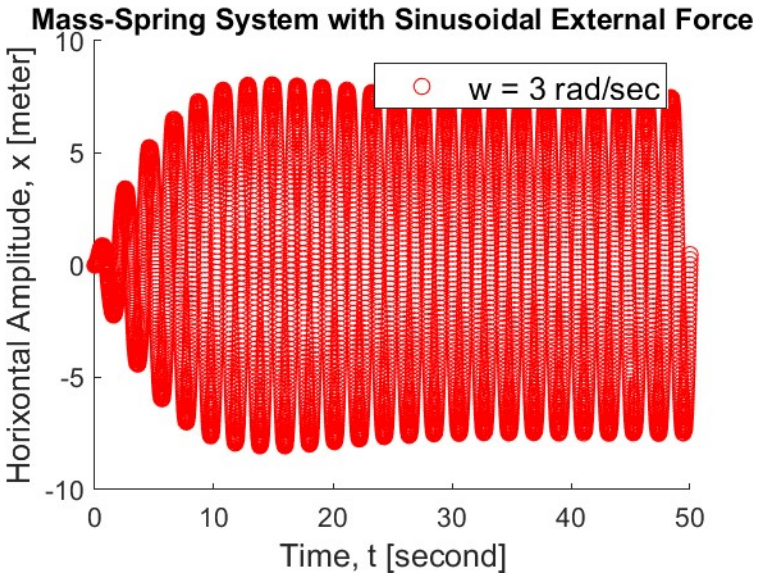

Figure 3. Euler's Implicit Forward step method for a mass on a spring with friction

3.4 Discussion

As we can see in both Figure 2 and Figure 3, we obtain very similar graphs for the two methods while other aspects remain constant. The general shape of the curve is very similar, though the one in Figure 3 is slightly more consistent once it reaches its peak value with only a small drop off. This again appears as though the object is gradually moved by the external force, slightly overshooting its maximum, and then returning to a more consistent, stabilized cycle, where the mass and external force resonate. Overall, this problem was highly complex; at various times, the math was difficult, and I still struggle to understand some of the calculations which Matlab conducted, though it was interesting to see the various methods and the differences between them.