Homework 3 Part 2 Instructions

September 17, 2018

1 Overview

We'll review and implement the oscillator problems again, to make sure that we can do it and extend this method to future problems.

Partial credit will be given when appropriate. If output is off but style is good and code is close or on the right track, I will give partial credit. At best, make sure your code doesn't crash when it runs. If you can't get code to not crash but still want it considered for partial credit then comment out the troublesome parts. **Test your submission to make sure it does not crash!**

2 Score and time limit Summary

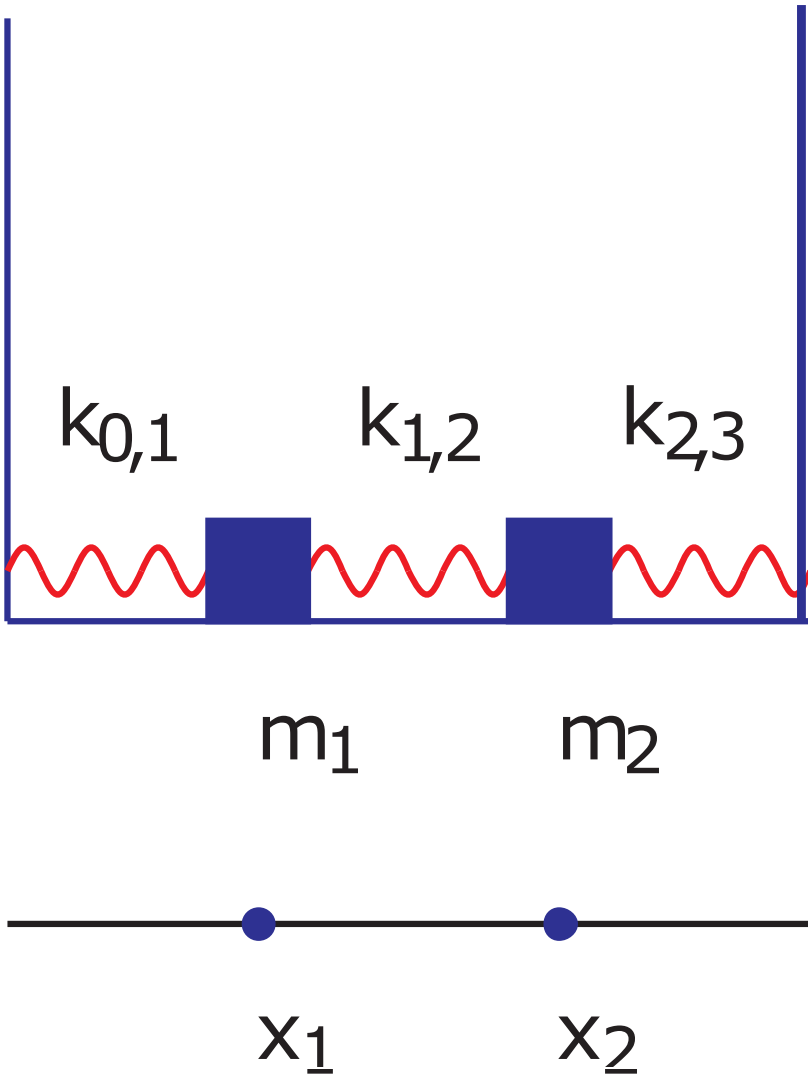
- Two Oscillators - 15 points
  - Implementation - 10 pts
  - Style - 5 pts
- Three Oscillators - 25 points
  - Implementation - 20 pts
  - Style - 5 pts
- N Oscillators - 40 pts
  - Implementation - 35 pts
  - Style - 5 pts
- Correct submission - 10 pts

Total score: 90 pts

3 Coupled Oscillators

3.1 Pair of Oscillating Masses

We'll start by reviewing the n=2 case for harmonic oscillators. That is two masses and three springs - one spring between the masses and one spring between each mass and the bounding wall. See Figure 3.1 below to get a visual.



As you continue to study physics you'll certainly find that doing force balances is not the easiest way to approach a problem. For this problem though it is done within reason and is a more intuitive way to understand the problem. We'll look at the force balance for each mass separately, using the spring force $F = kx$ for each spring.

First to clarify notation, we will define each mass as m_i where i is the "number" of the mass. Think of this like a unique ID number for each mass. Each

of these masses will have a displacement x_i numbered in the same way. The key here is to define this displacement x as the distance from the equilibrium position of the spring, starting from the leftmost spring. This will determine whether the previous spring is compressed, stretched, or at equilibrium (no force). This will require that the system is modeled from left to right so that the displacement in one spring is accounted for in the spring on the other side of the same mass. For example, if $x_1 > 0$ and $x_2 = 0$ then the first spring will be stretched (wants to compress, dragging m_1 to the left in the negative x direction. Since x_2 was zero then the second spring is in equilibrium and has no force on m_2 from itself. This would however require the third spring to be compressed which would then push mass 2 to the left (negative direction). Consider several cases and think about which springs are compressed and which are extended and what that means for which way the masses will be pushed or pulled. Remember that a negative position for a mass is to the left of its equilibrium position, and will compress the left spring and extend the right spring.

Once you feel comfortable with how the mass positions and spring compressions are related, we can start to model the system. We'll do a force sum for each mass. For any springs between masses we will have to get the difference of the position of both masses. For instance if m_1 is far to the left and m_2 is far to the right the spring between them is very stretched, and will pull m_1 strongly to the right and m_2 strongly to the left. The tricky part is getting the signs right on each term. Think about what a positive position will mean for the direction the spring will push or pull. Try to write the force equations on your own first then check if they're correct below:

$$\Sigma F_1 = m_1 a_1 = -k_{0,1} x_1 + k_{1,2} (x_2 - x_1) \tag{1}$$

$$\Sigma F_2 = m_2 a_2 = -k_{1,2} (x_2 - x_1) - k_{2,3} x_2 \tag{2}$$

With the equations, you can see more clearly what will happen in the system. For instance if x_1 and x_2 are equal and positive (meaning mass 1 and mass 2 are pushed to the right by the same amount), then the spring between them is neither stretched nor compressed. This is shown in the equations because

the $x_2 - x_1$ is zero in this case, but mass 2 is pushed back to the left by the $k_{2,3}$ spring and mass 1 is pulled to the left by the $k_{0,1}$ spring.

Now that we have our equations we need a way to simulate its motion given some initial conditions. What is the acceleration for each?

The acceleration is actually not a great place to start. This is where discretization is advantageous because it is easier to understand what is happening, and it is easy to work with discrete data points on a computer. Discrete just means that we are working with a finite set of data points. These data points model a continuous function in this case but we only worry about the change between points based on our time step. As the timestep approaches zero, it gets closer and closer to a continuous representation. Depending on conditions a timestep around the order of 0.0001 should be more than sufficient. To get to this discrete representation, consider the acceleration and a different way to represent it:

$$a = \frac{dv}{dt} \tag{3}$$

This represents a change in velocity over a time step. We can define a constant time step, and since we are looking at discrete points in time, we can break dv into the difference between the velocity at the next time step and the velocity at the current timestep. Consider the equation below, where n is the timestep number.

$$a = \frac{v_{n+1} - v_n}{dt} \tag{4}$$

If $a_i = v_{i,n+1} - v_{i,n}/dt$ is the acceleration at a timestep $n+1$ for mass i, then Equation 1 can be rewritten as

$$m_1 \frac{v_{1,n+1} - v_{1,n}}{dt} = -k_{0,1}x_1 + k_{1,2}\left(x_2 - x_1\right) \tag{5}$$

The only way to model this system, as well as other systems, is with some set of initial conditions. So we can start by setting the velocity at timestep 0

$v_{i,0}$ and the initital position $x_{i,0}$ to some values. In doing this we will have a place to start our discretization and will need to continue along at the next step. The key takeaway from this is that we will have to solve Equation 5 for the *next* time step $v_{i,n+1}$:

$$v_{1,n+1} = \left(-\frac{k_{0,1}}{m_1} x_{1,n} + \frac{k_{1,2}}{m_1} \left(x_{2,n} - x_{1,n} \right) \right) dt + v_{1,n} \tag{6}$$

Now you can do this for every mass in this type of system. Notice that I added a timestep identifier n to the x positions as well, as these will be tracked through the time we model the system for. We will also have to discretize the position of each mass, but this is much simpler as it is based on the velocity and time step which is already taken care of by our work with the force equations.

$$x_{i,n+1} = v_{i,n} dt + x_{i,n} \tag{7}$$

This is all we need to model the system. Now how to put this into Python?

First lets consider what data we need, its type, and the best way to store it for access. For each mass, we will have a set of positions (one at each time step) and a set of velocities (also at each time step). These both sound like lists or arrays right? And we have a set of masses, which perhaps could also be represented as a list/array. For simplicity we will assume that all spring constants are equal and all masses are equal. Note that there are many ways to go about doing this representation, and the one I use here is an attempt at an intuitive representation. Feel free to use any representation you like.

Lists are very basic types, but for many internal computer science reasons that we don't need to get into right now, a numpy array is faster, and more efficient with its memory usage. We will use this now, as our data sets may eventually reach a very large size. If we can combine all of this data into one superset, then everything can be accessed via indexing that represents which mass, which set is being used for that mass (position or velocity) and that value at a specific data point. In a pseudo code sense I would have:

```
oscillators[mass number][position (0) or velocity (1)][time]
```

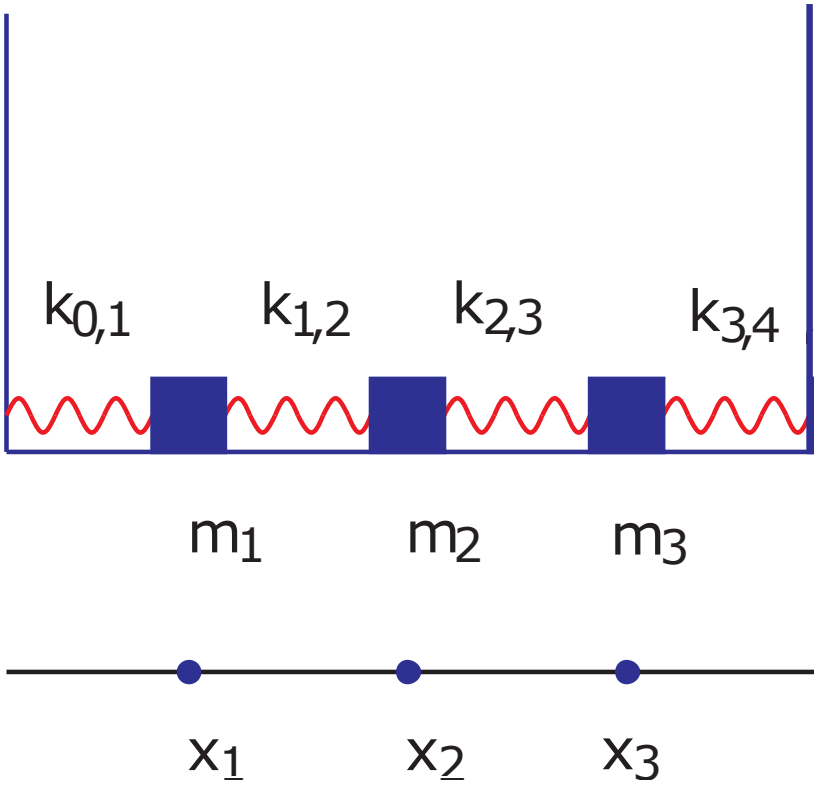
If we have a determined number of steps for the time index, then we can create a numpy array of arrays to act as a container for all the data we will soon model. Assuming you have imported numpy as np

```
oscillators = np.zeros((2, 2, steps + 1))
```

The first 2 represents the number of masses, the second represents the number of data sets for each (one of position and one for velocity), and the number of data points for those. We use steps + 1 because we will be starting with initial conditions for t = 0. At this point, go over to the two_oscillators.py file and fill in the section that sets up the initial conditions.

You'll see that we just adjusted the 0th index of the velocities and positions for each mass. From there the rest of the data is determined by our discretization. We need to do the discretization for as many steps as we defined, so this will be done in a for loop. In the python file, there is a guide for how you will implement Equations 7 and 6. Once you do that, read through the rest of the code to understand how the plot is made. Note that this code won't run until you fill in some of the blanks, such as determining the time step dt.

3.2 Three Coupled Oscillators



Looking at Figure 3.2, the problem is very similar to the pair of coupled oscillators, but the middle mass will have a slightly more complex force equation. Consider how the spring between the pair of masses was dependent on the difference between the two positions. This holds for any spring between two masses, so for the set of three this needs to be applied for both springs attached to the middle mass. Write out the force equations on your own, Then change to the form with dv/dt and solve for $v_{i,n+1}$. Check your equations against the ones below:

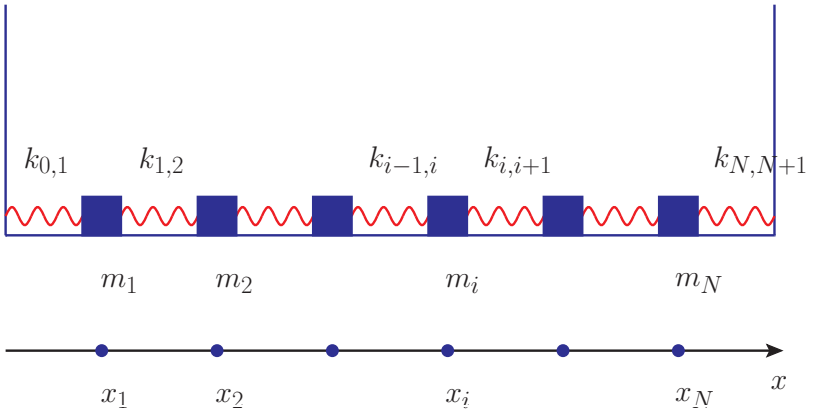
$$v_{1,n+1} = \left(\frac{-k_{0,1}}{m_1} x_{1,n} + \frac{k_{1,2}}{m_1} \left(x_{2,n} - x_{1,n} \right) \right) dt + v_{1,n} \tag{8}$$

$$v_{2,n+1} = \left(\frac{-k_{1,2}}{m_2} (x_{2,n} - x_{1,n}) + \frac{k_{2,3}}{m_2} (x_{3,n} - x_{2,n}) \right) dt + v_{2,n} \tag{9}$$

$$v_{3,n+1} = \left(\frac{-k_{2,3}}{m_3} (x_{3,n} - x_{2,n}) - \frac{k_{3,4}}{m_3} x_{3,n} \right) dt + v_{2,n} \tag{10}$$

With these equations, go into three_oscillators.py and implement them in the code, and check the initial conditions that have been set for you and how the oscillators variable was initialized slightly differently. The implementation is very similar as in the last part but you now have an extra equation with a slightly more complex form to account for the extra coupled spring. Note that I left the subscripts on the masses and spring constants for clarity, but in the implementation we use the same value for all of them.

3.3 N coupled oscillators



As we expand to handle N coupled oscillators as in Figure 3.3, the key is to look for a pattern. First consider the equations for if we had 4 coupled oscillators:

$$v_{1,n+1} = \left(\frac{-k_{0,1}}{m_1} x_{1,n} + \frac{k_{1,2}}{m_1} (x_{2,n} - x_{1,n}) \right) dt + v_{1,n} \tag{11}$$

$$v_{2,n+1} = \left(\frac{-k_{1,2}}{m_2} \left(x_{2,n} - x_{1,n} \right) + \frac{k_{2,3}}{m_2} \left(x_{3,n} - x_{2,n} \right) \right) dt + v_{2,n} \tag{12}$$

$$v_{3,n+1} = \left(\frac{-k_{2,3}}{m_3} \left(x_{3,n} - x_{2,n} \right) + \frac{k_{3,4}}{m_3} \left(x_{4,n} - x_{3,n} \right) \right) dt + v_{3,n} \tag{13}$$

$$v_{4,n+1} = \left(\frac{-k_{3,4}}{m_4} \left(x_{4,n} - x_{3,n} \right) - \frac{k_{4,5}}{m_4} x_{4,n} \right) dt + v_{4,n} \tag{14}$$

The trick is the middle masses not next to the walls all have the same form. This can therefore be handled by a loop that adjusts the index regarding which mass is being referenced. Then the first and last masses have to be handled separately, but just as they were in the three oscillator problem. Go into the n_oscillators code and check out how the variables are initialized, how n is used, and then fill in the code to handle the discretization for each mass. Read the comments, they'll help you out a bunch.

4 Submission

It is EXTREMELY IMPORTANT to me that you do this correctly. Plz.

You should have a file structure as follows:

```
working_directory
        username_hw03_part2
                two_oscillators.py
                three_oscillators.py
                n_oscillators.py
```

The `working_directory` doesn't matter, work wherever fits your organization style. Everything from `username_hw03_part2` matters. **Match this name exactly** but with your username. **NOT** `username_HW03_part2`, **NOT** `username_hw03part12`, **NOT** `username_hw3_part2`...

This attention to specifics is very important in computing, it can help avoid bugs in code, but also prevent accidental damage to systems or sensitive data. This will become clearer the farther you go into computing. Pay attention to these details and you will not have any penalties!

To submit, zip up `username_hw03_part2`. **DO NOT** select all the python files at once and zip those, zip the whole folder. **DO NOT** have extra levels of folders. For example the following structure would result in a penalty:

```
working_directory
        username_hw03_part2.zip
                username_hw03_part2
                        username_hw03_part2
                                two_oscillators.py
                                three_oscillators.py
                                n_oscillators.py
```

When you zip up your folder for submission, you can verify that its right. Double click on the zip folder and you can still see what's in it without re extracting the folder. On the first level, you should see a folder with the name following `username_hw03_part1`. If you look inside that, you should see your python files. If for example you see python files immediately after looking inside the first level of the zip folder, it's not correct. If you have to look into the zip folder then a folder then *another* folder, it's wrong. Double check this! Ask your classmates for help, send me an email, or check in office hours. Me checking that your submission structure is right now saves a lot of time later. I don't want to penalize anyone for this, but we have to be able to follow specifications!

Have fun and keep up the great work!