Dynamics Final Report

Sophie Li and Hannah Wilk

1 Abstract

We set out to develop a n-rigid body solver in MatLab. We wanted a user to define how many rigid bodies there are, and our system creates the correct equations of motion and animates the system. using basic equations of motion,$F = ma$ and $\sum M = \frac{d}{dt}(\omega * I)$ we where able to develop our n-pendulum solver. Using a GUI developed in python, our users can now animate a pendulum of any size.

2 Background

Over the course of the semester, we have derived the equations to pendulums multiple times. With each problem every week, we learned more about the pendulum system, and how to properly assess the physical situation. With each iteration of solving similar problems, we began to notice patterns. We came to the conclusion that these patterns could be transformed into an n case. No matter how many equations of rigid body pendulums we were asked to find we could find them, if we developed a MatLab script.

In parallel with noticing these mathematical patterns in the equations of motion, we discovered a physics based video game called Crayon Physics [1]. In this game the player was tasked with getting a ball to a goal. In their way are various obstacles, such as cliffs, water, or other physical barriers. Granted only a magic pencil, they have to draw various ramps and pendulum to get the ball to the goal. In this game, a lot of real time physics are used. Pendulums can be any size and need to work like pendulums. This inspired us to use our n-case equations to create a simplified GUI, so the user can control how large the n-case pendulums can be.

Newtonian Physics, and simplified Euler's equations will be applied to each link of an n=pendulum system to create our system. Since we will have an n-pendulum generator, we realized that there needs to be a user input option. Using python, we will integrate a GUI interface to make our n-pendulum more user-friendly.

3 Learning Objectives

After learning about the video game Crayon Physics, we decided we wanted to look more into allowing the user to define aspects of a dynamic system defined in MatLab. After our three years at Olin, we hadn't been asked to make a GUI in MatLab, this would allow us to try and tackle the complex feat. In addition to that, we wanted to work on fully understanding transformation matrices- how to set them up and how to properly use them. After taking the time to explore the possibilities in this area, we were able to develop even more learning goals. We wanted to understand how to generalize the pendulum's equations. After many weeks of working on p-sets involving pendulums we saw patterns start to emerge. Finally, we wanted to learn more about MatLab animation, and how a user can interact with MatLab's graphical interfaces. Will integrate a GUI interface to make our n-pendulum more user-friendly.

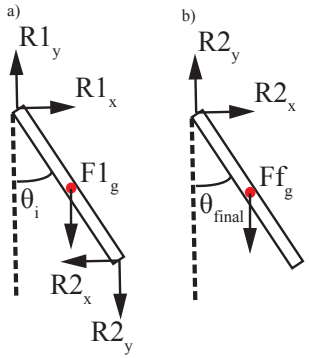


Figure 1: Free Body Diagram of System. a) of this figure shows a generic, non final, link in the pendulum. b) shows a generic final link in our n pendulum system. Note that R2 has an equal and opposite force for each pendulum

4 System Models

4.1 Free Body Diagram

Each link in the system has a momentum and forces associated with it. Each link has two reaction forces, one from the link before and one from the link after. In addition to the reaction forces, there is the force associated with gravity. The only exception is the final link of the pendulum. There is only one reaction force, created by the link before the final one. When the moment is taken with respect to the center of gravity, the two reaction forces create moments, acting in the same direction.

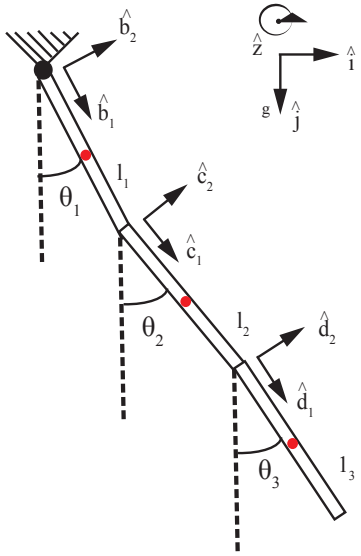


Figure 2: Shows how reference frames are defined in reference to each link. This and example of a three rod pendulum.

An incredibly important step to solving an n case pendulum is to look at cases to develop patterns. One thing we did was set up a triple pendulum to create a starting point. With this pendulum we were able to

clearly set up how we were going to use reference frames. Each link of the pendulum has its own reference frame. These were used, along with the vertical angles to develop transformation matrices. It is important to note that the $\hat{b_3}$ direction is parallel to all of the other Z directions produced by the other reference frames. This means for all of our moment equations, whether it was made in the $\hat{b}$ or $\hat{c}$ direction, the final $_3$ direction were equivalent. $\hat{b_3} = \hat{c_3}$

4.2 Equations

It is important to note that n is the total number of rigid bodies in the pendulum system. i is the index for the rigid bodies. For example: i=1, is the first body, or the top pendulum, i=2 is the second, i=n would be the final pendulum in the system. After going through math that was similar to math found in Problem set 5, we decided to make an n case model.

Our first step was to analyze our absolute acceleration

$$A_n = \sum_{i=1}^{n}(L_i\ddot{\theta}_i^2(-cos(\theta_i)\hat{i}+sin(\theta_i)\hat{j})-L_i\dot{\theta_i}^2(sin(\theta_i)\hat{i}+cos(\theta_i)\hat{j})-\frac{L_n}{2}\ddot{\theta_n}(-cos(\theta_i)\hat{i}+sin(\theta_i)\hat{j})+\frac{L_n}{2}\dot{\theta_n}(sin(\theta_i)\hat{i}+cos(\theta_i)\hat{j}) \tag{1}$$

We then looked into the sum of forces, or more simply:$F = ma$. From this we analyzed the FBD and came up with the equations for every pendulum.

$$\sum F_n = -R_{nx}\hat{i} - R_{ny}\hat{j} + m_n g\hat{j} + R_{(n+1)x}\hat{i} + R_{(n+1)y}\hat{j} \tag{2}$$

Equation 2 represents our generalized sum of forces equation. Each force has an equal and opposite effect on the next consecutive link. We needed to develop an equation that took into account what link the index was on, and how many total n-pendulums there were. In order to account for the edge case- there only being two forces on the last pendulum instead of three- we needed to build a for loop into our solving logic, that removed the final force. After solving for the sum of moments, we set the equation to Euler's equations. Because of the geometry of the rods, we were able to drop many terms in euler's equation. We were able to use $\sum M = \frac{d}{dt}(\omega * I)$. The sum of moments where the calculated equations we found from the center of gravity of each rod, using the FBD to choose the correct forces. The right side of the sum of moment equation became $\dot{\omega}\dot{I}$ Where $I = \frac{mL^2}{12}$, the mass moment inertia of the rod at its center of gravity.

$$\frac{m_n {L_n}^2}{12}\ddot{\theta_n}^2 = \frac{-R_{nx}L_n}{2}sin(\theta_n) + \frac{R_{ny}L_n}{2}cos(\theta_n) - \frac{R_{(n+1)x}L_n}{2}sin(\theta_n) + \frac{R_{(n+1)y}L_n}{2}cos(\theta_n) \tag{3}$$

Equation 3 is the generalization of our findings of the equation$\sum M = \frac{d}{dt}(\omega * I)$. Since we are setting up a matrix in order to generalize this problem, we needed to get the correct forces onto the correct sides. We needed to see all terms with $\ddot{\theta}$ and reaction forces on one side. On the other, terms with no unknown forces and $\dot{\theta}$. What we got in part where generalized equations split into the $\hat{i}$ and $\hat{j}$ directions.

$$\sum_{i=0}^{n}(-m_n L_i cos(\theta_i)\ddot{\theta_i}) - \frac{m_n L_n}{2}cos(\theta_n)\ddot{\theta_n} - R_{nx} + R_{(n+1)x} = \sum_{i=0}^{n} -m_n L_i \dot{\theta_i}^2 sin(\theta_i) + \frac{m_n L_n}{2}\dot{\theta_n}^2 sin(\theta_n) \tag{4}$$

$$\sum_{i=0}^{n}(m_n L_i sin(\theta_i)\ddot{\theta_i})-\frac{m_n L_n}{2}sin(\theta_n)\ddot{\theta_n}-R_{ny}+R_{(n+1)y} = \sum_{i=0}^{n} -m_n L_i \dot{\theta_i}^2 cos(\theta_i)+\frac{m_n L_n}{2}\dot{\theta_n}^2 cos(\theta_n)-m_n g \tag{5}$$

These final two equations where the basis of our logic in creating n-case pendulums. These were input into loops to create the repetitive nature of the final pendulum equations.

4.3 Assumptions

To simplify our system, we assumed that there were no dampening forces acting on the system. This includes the exclusion of drag and other effects due to external forces. We also assumed the force of gravity was a

constant 9.81 $\frac{m}{s^2}$, seeing as this system is being analyzed close to the ground on Earth. While looking at the pendulum links, we assumed that each link was a uniform rod with length L_i and mass M_i. Each rod had an angle, θ_i, with respect to the vertical. While defining the equations of motion, we took them with respect to the center of mass-the center of the rod.

While we were solving our rod problems, in order to validate our work, we were looking at position, velocity, acceleration, and most importantly energy. If our total energy ended up being a straight line, we said that the simulation was working correctly. This is physically a strong case, however there could be errors related to energy. However we ignored that potential and accepted the energy graphs as validation.

5 Results

To make the experience more user friendly, we wrote a GUI that would allow users to define how many pendulums and its starting conditions.

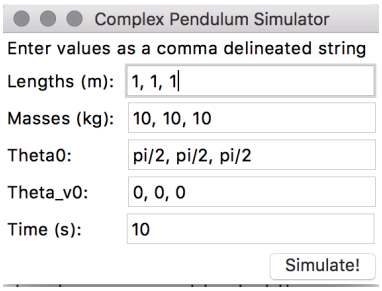
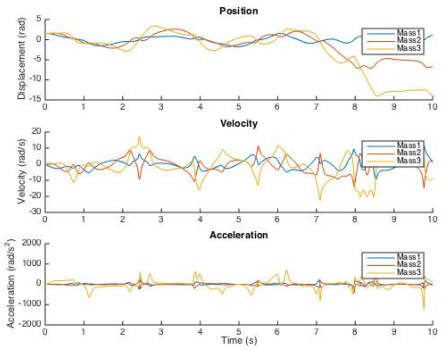
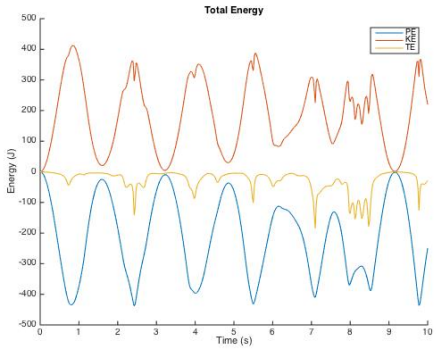


Figure 3: GUI with 3 pendulums



(a) Position, Velocity, Acceleration



(b) Energy Graphs for 3 pendulums

One can see that the energy graphs aren't quite flat, though we attribute to calculation errors due to rounding as previously discussed. Theta, theta-dot, and theta-ddot behave as expected, with each additional mass having greater displacement than the previous. The acceleration graph shows sharp peaks where the pendulum would have hit the peak of it's motion and began moving in the opposite direction.

Since our code successfully generates Matlab scripts for n-case pendulums, we added additional rods to see what would happen.

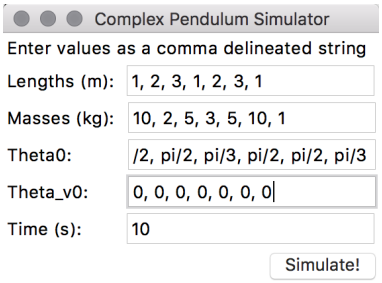
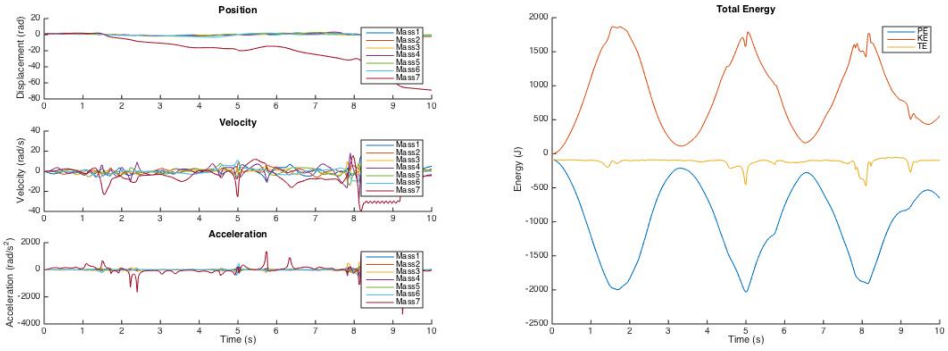


Figure 5: GUI with 7 pendulums



(a) Position, Velocity, Acceleration

(b) Energy Graphs for 7 pendulums

With 7 pendulums and randomized starting conditions, we still see that energy is flat, which indicates that our model is working.

For fun we simulated 20 pendulums:

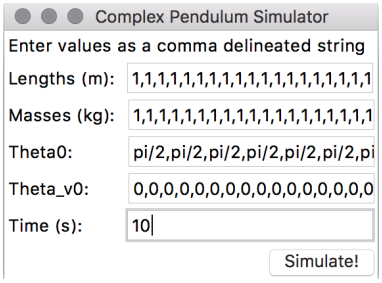
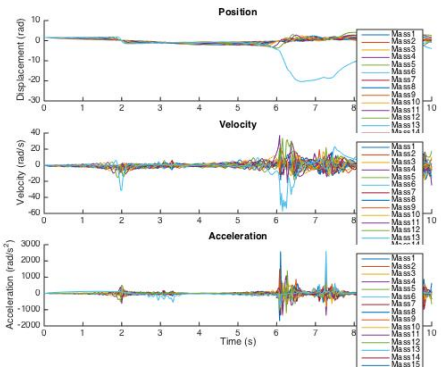
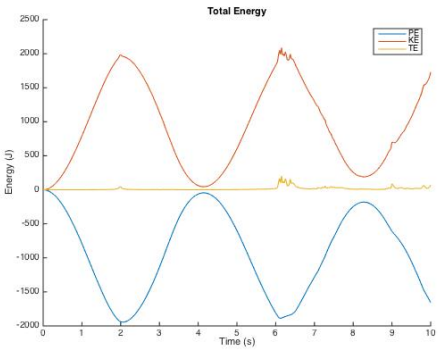


Figure 7: GUI with 20 pendulums



(a) Position, Velocity, Acceleration



(b) Energy Graphs for 20 pendulums

And energy is still flat! The auto-generated code for 20 pendulums was around 750 lines of MatLab.

We observe that as additional rods are added, the motion of the pendulums becomes slower, as each rod adds both length and mass. As one can see in the energy graphs for each iteration of the test below, the 3-link pendulum had quite a few oscillations, while the 20-link pendulum only had three.

6 Using the Code-Animation

To run the entire project, do the following:

1. Use the Python GUI to input the initial conditions. Click "Simulate" to generate the MatLab script
2. After Python creates the MatLab script with the proper matrices and equations, run the script "n_pen.m" inside MatLab. Validate your solution by plotting energy for each test case

After MatLab is done running the script, the user will be presented with three graphs. The first is the position, velocity, and acceleration of each theta in the pendulum. The second is the energe of the system. The third is a animation of the pendulum motion. An .avi file of the animation will be saved to the local folder after the animation is finished playing.

7 Diagnosis

7.1 Reference Frame Troubles

Initially our pendulum did not behave properly. Our first pendulum was not being affected by the other links in the pendulum. After hours of debugging we realized we had switched the orientation of one of our forces. This meant that our transformation matrix was incorrect. While we understood the mathematical steps that had to be addressed in order to solve the pendulum problem, we did not realize we had made a simple FBD error. Instead of the 1 component of our reference frame going along the length of pendulum, it was going perpendicular to the base. This cause our transformation matrix to change from: $\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \text{-}\sin(\theta) & \cos(\theta) \end{bmatrix}$ *to* $\begin{bmatrix} \sin(\theta) & \cos(\theta) \\ \text{-}\cos(\theta) & \sin(\theta) \end{bmatrix}$. They are not equal transformation matrices. This was causing our issues and was breaking our pendulum. While we caught the mistake, it was frustrating. The mathematical steps were straight forward but a small accounting error had caused us many hours of work. One should always make sure they understand how their reference frames are set up.

7.2 Matlab's Errors, Not Ours

Initially, we accomplished an n-body pendulum solver by using for loops inside our Matlab ODE to generate the A matrix and r vector. (Where A matrix is composed of our unknown variables while the r vector is the

known.) However, we were unable to get flat energy graphs due to cumulative errors that occurred during our matrix evaluation phase.

To validate that this was indeed a Matlab issue and not an implementation issue on our end, we compared the results from our n-body solver for a 3-link pendulum with another script that simulated the motion of the same 3-link pendulum using an A matrix and r vector that we defined like in previous problem sets instead of generating them on the fly like in our solver.

The results between the two were identical for the first few time steps, but our solver produced values that increased faster than the second script. We concluded that this was a Matlab issue in matrix evaluation, so we set out to do n-body solving in a different manner.

By writing a python program to generate a Matlab script, we could define the A and r matrices instead of generating them inside the ODE. Using this implementation, we were able to reduce both computational complexity and increase the accuracy of our results.

8 Improvement

We currently have a rudimentary GUI for a user to interact with, but it is used in Python. As of right now the user can enter lengths, masses, θ initial, and velocity initial. Each is entered in its own field on the GUI. To indicate how many pendulum needed, one adds terms to each of these inputs, indicating different links, using commas. While the GUI is functional, it is lacking in design. What we mean by that it is not incredibly intuitive for the user to use. It is not as elegant as Crayon Physics, where the user can easily draw however many pendulums they want. Currently, it becomes difficult to input the conditions for a long pendulum. If one wants to make a 20 rod pendulum, one has to type out 20 different elements, 4 times, while making sure the input length is constant. If we were to continue to work on this project, one more iteration of GUI design could go a long way to making our experience function more like a video game. While the GUI was not written in MatLab, we still were able to learn about how GUI's are implemented into larger code.

In addition to adding more of a GUI, we would also look more into the MatLab triggered errors. When we did our first iteration we had the correct equations and respective matrices, but MatLab was plotting incorrect graphs, that acted unstable. As time progressed, the instability got worse. This led us to believe that it was an ODE problem. We would need to look into different ODE solves to fix the problem to remove the python/MatLab integration. This would allow us to stop calling a python script to make a MatLab script.

9 Reflection

While working on this assignment, we did a lot of good work developing our understanding of reference frames, moments, and forces. While going through this project, we derived equations of motion for1, 2, 3, 4, and 5 bar pendulums. This meant that we were pushing our understanding of how to set up complicated reference frame based problems. We had to constantly be on the lookout for sign errors and had to make sure correct transformation matrices were correlated to the proper angles. We became master debuggers. With the errors that were made with transformation matrices, we can now solve them in our sleep (I guess we will see how we do on our final exam.). Overall we developed a lot of good skills that will help us solve more complicated problems in the future. It was a lot of fun even though the work was frustrating at times.

10 Conclusion

When we began to work on this project we set out to make a complex video game that could produce user input pendulums of any length. We explored complex dynamic principals and learned more about animation in MatLab, in order to develop our n-case pendulum. We used the sum of forces and Euler's equations on rigid rods in order to develop equations of motion which allowed us to fully animate any n order pendulum. While there were issues with MatLab, we were able to work past it using python in order to create a GUI centered pendulum simulator, which acts as code validation for the Dynamics classes to come. Our final

iteration allows for a user to individually define inputs for the pendulum, while displaying an animation, along with plotting the all-important energy validation graph.

11 Future Use

While we learned how to solve complicated equations of motion for this problem, we are not sure how much more added value a problem like this would add to the class. Seeing as we were looking for a generalized solution to solve and size pendulum problem, we found a lot of repetitious solutions. After the third pendulum, the equations become easy to create by hand. This would be more of a problem to intrinsically teach debugging techniques, which has its own value, but isn't necessarily the best use of the class's limited time. We created a set of scripts that can validate pendulum code. People in the future classes can use our code to check to see if their rigid pendulum is correct.

11.1 Problem Statement

Pendulum Case Study:

(a) Find the equations of motion for one rigid body pendulum, place equations into a matrix, plot energy

(b) Find the equations of motion for a two mass rigid body pendulum

(c) Analyze patterns found in the first two pendulum cases, and develop a generalized matrix

(d) Show multiple examples of your code working, i.e. use different number of total rods. Validate your solution by plotting energy for each test case

References

[1] Crayon Physics, Petri Purho, Kloonigames ,2009