Numerical simulation of the damped and undamped double pendulum systems using MATLAB

George Luther (13528904)

School of Physics and Advanced Materials, University of Technology, Sydney, 15 Broadway, Ultimo, New South Wales 2007, Australia

E-mail: George.Luther-1@student.uts.edu.au

Date: 10/06/2020

Abstract

This investigation aims to demonstrate the chaotic nature of the damped and undamped cases for a double pendulum system through numeric simulation. The parameters which were varied in this simulation are the pendula lengths, masses of the two bobs, the acceleration due to gravity and, the initial angles of the pendula. Damping is introduced in the form of air resistance on each rod; these constants lower the total energy of the system over time. In the computer program, each 2nd order ODE was broken up into two respective 1st order ODE's that were used in the algorithm. The 4th order Runge-Kutta algorithm was ultimately used as the numeric integrator for the two 2nd order ODE's introduced governing the motion of each pendulum arm. The simulation produced results that differed drastically with small initial angle deviations, which is to be expected in such chaotic systems. The numerical solution was validated based on the assumption of constant energy in the underdamped case.

Nomenclature

m_1	mass of pendulum 1 (kg)
m_2	mass of pendulum 2 (kg)
l_1	arm-length of pendulum 1 (meters)
l_2	arm-length of pendulum 2 (meters)
t	time (seconds)
ϕ_1	the angular displacement of pendulum 1 (radians)
ϕ_2	the angular displacement of pendulum 2 (radians)
$\dot{\phi}_1$	angular velocity of pendulum 1 (rad/s)
$\dot{\phi}_2$	angular velocity of pendulum 2 (rad/s)
$\ddot{\phi}_1$	angular acceleration of pendulum 1 (rad/s^2)

$\ddot{\phi}_2$	angular acceleration of pendulum 2 (rad/s^2)
k_1	damping constant of pendulum one due to air resistance (dimensionless constant)
k_2	damping constant of pendulum two due to air resistance (dimensionless constant)

1. Aim and Motivation

This project demonstrates chaotic motion due to the sensitivity of the initial conditions which govern its behavior. Many systems in the universe behave in this chaotic way, and we rely on numeric simulations to help visualize and interpret them as there may be no clear analytical solution for that problem currently. Numerical integrators such as the 4th order Runge-Kutta (RK4) algorithm is one such example of a numerical approximation to an analytical solution by inputting two uncoupled 2nd order ODEs that describe the motion of the system in the case for a double pendulum. The MATLAB code will be able to be modified to simulate the movement of other chaotic systems with 2nd order ODEs by replacing the differential equations used in this model with the new system's ODEs.

This project aims to simulate the chaotic nature of a double pendulum system with varying initial conditions to help visualize the problem further and its sensitivity to initial conditions. The effect of air resistance was demonstrated in this model through the damping constants, k_1 and k_2 that are multiplicative on the angles and are dimensionless for simplicity in the numerical application.

2. Introduction

Chaos theory is the unpredictability of a system based on initial conditions that were imposed upon it. It is difficult to visualize such nonlinear systems, such as the three-body planetary motion problem, as there are no clear analytical solutions to most of these problems. The double pendulum is one example of a chaotic motion system that does not have an analytical solution, which is why we rely on a numerical model [1-2].

One such method of producing a numerical solution is by using the MATLAB function, ODE45. This function uses the 4th order Runge-Kutta method and takes one single 2nd order ODE as an input, transforming it into a single 1st order differential equations. However, one way to use the ODE45 is to introduce a state-space vector taking both angular displacements and angular velocities as elements, being a 4x1 matrix [2]. However, this eliminates the versatility of the code, where it is applied to other chaotic systems such as the three-body problem by defining new variables and re-writing the equations of motion. This versatility is the justification for using the RK4 method directly in the MATLAB script apart from its simplicity, analogous to using state-space vector form.

Because this system does not have an analytical solution, much like other chaotic systems [2], it is difficult to conduct a validity check to test the accuracy of the behavior of the nonlinear system. Therefore, basic laws of physics must be applied to the system to verify the validity. The simulation is valid by our definition if the following rules apply:

1) If the mass of the first bob is larger than that of the second bob, i.e. $m_1 > m_2$: The double-pendulum system should have a massive transfer of momentum from m_1 to m_2, causing the second rod to complete multiple revolutions while the motion of the first rod is hindered due to its large mass. This behavior is due to the first rod's enormous angular momentum being imposed on the second rod, whereas the second rod will have little momentum to transfer due to its low bob mass.
2) When the arm of the second pendulum is larger than the length of the first rod, i.e. $l_2 > l_1$ then the system should behave like a simple pendulum with periodic oscillations.
3) Similar to step 1) when the mass of the second pendulum bob is larger than the mass of the first bob, i.e. $m_2 > m_1$ then the system should behave like a simple pendulum with periodic oscillations.
4) For the damped case, that is, where $0 < k_1, k_2 < 1$ then the system will eventually come to a stop. As this damping constant gets smaller, the rate of energy dissipation will be more significant. <u>*This will be tested twice.*</u>
5) The total energy of the system, $E = T + V$, should be a constant value if there are no damping factors present. That is, $k_1 = 1$ and $k_2 = 1$. Because of no energy dissipation in the system, it will continue its motion infinitely.
6) At initial angles of $\phi_1, \phi_2 = 0°$ the pendula should remain stationary with no motion present. If motion occurs, it is due to a term that does not rely on the angle array inside for the function, which then suggests that our equation is incorrect. ***This case provides no useful information about the result, and the visualization is trivial. Therefore, it is left for the reader to understand this themselves.***

These rules will apply to the double pendulum system and will be used in the **Discussion section** of this report to verify the validity of the simulation.

3. Background Theory

3.1. The Euler-Method

When the mathematics of a system is too complicated or when there is no real analytical solution, we must rely on numerical integrators such as the Euler-Method to simulate these problems. The two numeric methods that will be discussed in this section of the report are the Euler method and the Runge-Kutta family of numeric integrators. The Euler-Method takes the definition of the derivative as a from Newton's first principles as:

$$v(t) = \frac{dx}{dt} = \lim_{h\to 0}\left(\frac{x(t+h)-x(t)}{h}\right)$$

By getting rid of the limit, we introduce the error term $O(h^n)$, which means we ignore every order past the first order, and it is known as the truncation error. Thus, making the Euler Method a linear numerical integrator. Continuing with the derivation,

$$\frac{dx}{dt} \approx \frac{x(t+h)-x(t)}{h} + O(h^n)$$

Then, the difference equation is, therefore obtained through the use of finite approximation:

$$x_{n+1} = x_n + hf(x_n)$$

Where $f(x_n)$ is a 1st order ODE. The Euler-Method is a linear numeric integrator, which means that it is accurate to the order of the step-size, h that one specifies into the code. This method was not used here, and the RK4 algorithm was used instead as this method is accurate to four orders of magnitude (10^4) higher than h. The order of accuracy due to step-size is because of the Taylor-Series Expansion of the RK4 algorithm using the first four terms of the Taylor Series, analogous to the Euler-Method that uses the first term.

3.2. The Runge-Kutta Family of Numeric Integrators

The Runge Kutta method is a numeric integrator which calculates the slope of a given function between separate time intervals, t_i and t_{i+1} where $h = t_{i+1} - t_i$. Higher orders of the Runge-Kutta method, say RK-N calculate the slope of a function using a moving average N times between t_i and t_{i+1} and it is accurate based on h to the order of 10^N [5].

Figure. 1 demonstrates the first-order Runge-Kutta algorithm, which uses a single slope to approximate the numerical solution for this nonlinear function. As seen, the time-step h must be extremely small to achieve a numeric solution that is close to that of the analytical one where h shown in the Figure is not sufficient enough to estimate the analytical solution. The higher order of Runge-Kutta corresponds to an increase in linear slopes, which leads to an increase in accuracy due to these slopes compensating for the large h. The RK4 algorithm uses four linear gradients, minimizing the truncation error $O(h^n)$. The RK4 algorithm takes the general form…

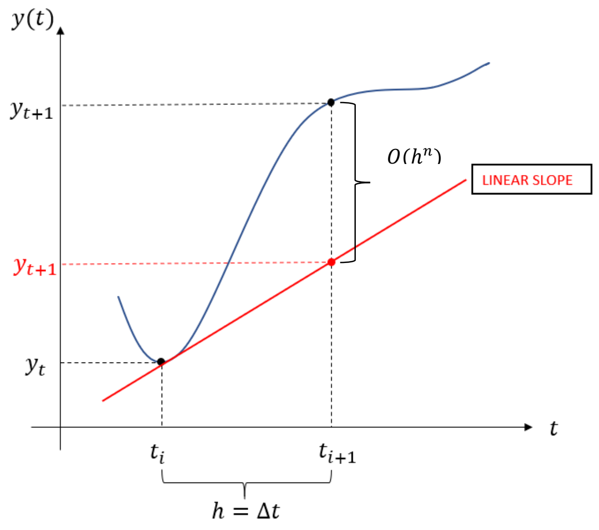


Figure. 1. The Runge-Kutta method of first-order as an example.

$$y_{t+1} = y_t + h(a_1K_1 + a_2K_2 + a_3K_3 + a_4K_4) + O(h^5)$$

Where a_i corresponds to the weights that the K_i slope value holds. Finally, because we are using the first four terms of the Taylor-Series expansion, our time-step is fourth-order accurate. Fourth-order means that if $h = 10^{-2}$, our accuracy will be to the order of 10^{-8}.

For the Classical Runge-Kutta 4 algorithm, these weights are as follows. These weights were obtained through tedious algebra:

$$y_{t+1} = y_t + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

Using $F(y, t)$ as the function,

$K_1 = F(y_t, t_i)$

$K_2 = F\left(y_t + \frac{h}{2}K_1, t_i + \frac{h}{2}\right)$

$K_3 = F\left(y_t + \frac{h}{2}K_2, t_i + \frac{h}{2}\right)$

$K_4 = F(y_t + hK_3, t_i + h)$

The Runge-Kutta method has its drawbacks when it comes to requiring uncoupled differential equations. Algebra is used to uncouple a set of differential equations, namely the process of simultaneous equations to eliminate relating terms between the two equations.

3.3. The Motion of the Double Pendulum

The basic model of the double pendulum appears in **Figure 2.** For the simulation conducted in this paper. The variables used are labeled here and defined (see Nomenclature). The assumptions made for this simulation are:

- Uniformly distributed point-masses, i.e., the volume of the bobs were not pre-defined and instead treated as points in space having mass.
- The mass of each pendulum is assumed to be massless; otherwise, if they were not, there will only be a minuscule change in the center of masses leaning towards the rod.
- The pendula are attached to a stationary block where they can move an entire revolution without causing a collision with the block.

Damping constants, $k_1$and k_2 were introduced **(see Appendix B),** which lower the total energy of the system over time. These constants have a value between 0 and 1, where there is no energy loss due to damping when $k_1, k_2 = 1$

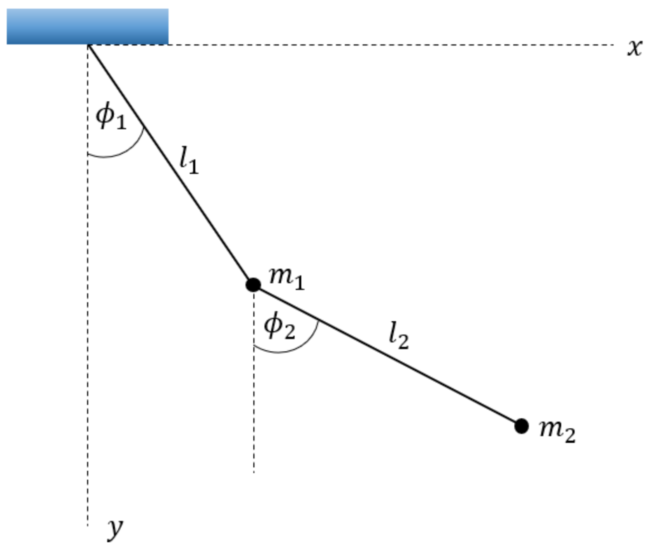


Figure 2. A diagram of the double pendulum system.

The equations of motion used to demonstrate the behavior in **Figure 2.** are:

$$\begin{cases} \ddot{\phi}_1 = \dfrac{(-g(2m_1+m_2)\sin(\phi_1) - m_2 g sin(\phi_1 - 2\phi_2) - 2\sin(\Delta\phi)\, m_2\big(\dot{\phi}_2^2 l_2 + \dot{\phi}_1^2 l_1 \cos(\Delta\phi)\big)}{l_1(2m_1+m_2-m_2\cos(2\Delta\phi))} \\ \ddot{\phi}_2 = \dfrac{(2\sin(\Delta\phi)\times\dot{\phi}_1^2 l_1(m_1+m_2) + g(m_1+m_2)\times\cos(\phi_1) + \dot{\phi}_2^2 l_2 m_2 \times \cos(\Delta\phi)}{l_2(2m_1+m_2-m_2\cos(2\Delta\phi))} \end{cases}$$

The process of obtaining these two equations is seen in Appendix A. The damping constants were not implemented in the equations of motion but instead were used in the RK4 for-loop where the integrated angular velocities, $\dot{\phi}_1$ and $\dot{\phi}_2$ were decreased by a factor of k_1 and k_2 at each time step, respectively.

4. Simulation Details and Validation:

The validation of this model, which was mentioned in the introduction, will be tested in this section. If the conclusions on validity are in-line with the outputs, then it is reasonable to suggest that the model works fine, and there are no clear errors. **The purpose of this validation section is to verify that the result is correct as there is no clear analytical solution, so such extreme plots are used with proper reasoning to justify that the code is working fine.** The cases for a validity check are tabulated below, which are the details on simulation from variables that will be adjusted (**Table. 1.**). Four cases will be examined here and discussed:

Table. 1. The parameters used for each case to test the validity and results in this process.

Parameter	Case 1	Case 2	Case 3	Case 4
Number of data points, n	6667	6667	6667	6667
Step size, h	0.009	0.009	0.009	0.009
length of rod 1, l_1	1	1	1	1
length of rod 2, l_2	1	5	1	5
mass of bob 1, m_1	5	1	3	10
mass of bob 2, m_2	1	1	1	1
gravitational constant, g	9.81	9.81	9.81	9.81
Initial angular velocity of rod 1, $\dot{\phi}_1$	0	0	0	0
Initial angular velocity of rod 2, $\dot{\phi}_2$	0	0	0	0
The initial angle of rod 1, ϕ_1	$\frac{\pi}{4}$	π	$\frac{\pi}{4}$	$\frac{\pi}{4}$
The initial angle of rod 2, ϕ_2	$\frac{\pi}{2}$	$\frac{\pi}{4}$	$\frac{\pi}{2}$	$\frac{\pi}{2}$
Damping constant of rod 1, k_1	1	1	0.999	0.999
Damping constant of rod 2, k_2	1	1	1	0.999

5. Results and Discussion:

5.1. Case 1: The mass of the first bob is bigger than that of the second, $m_1 > m_2$:

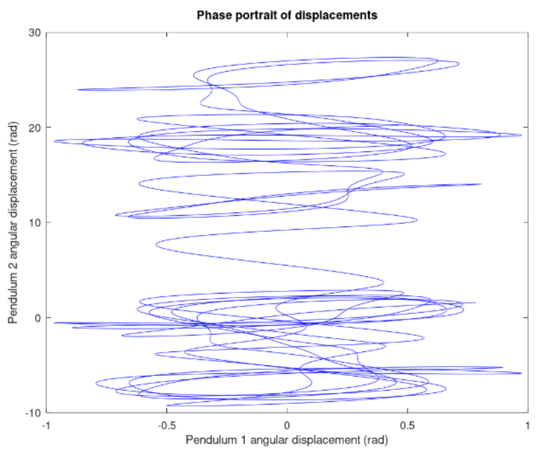


Figure. 3. The phase portrait of the angular displacement for Case 1.

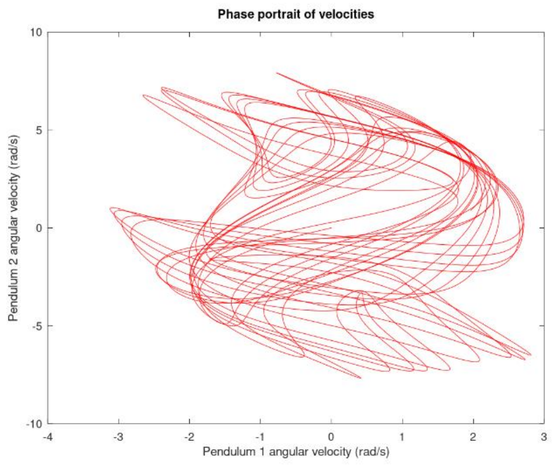


Figure. 4. The phase portrait of the angular velocity for Case 1.

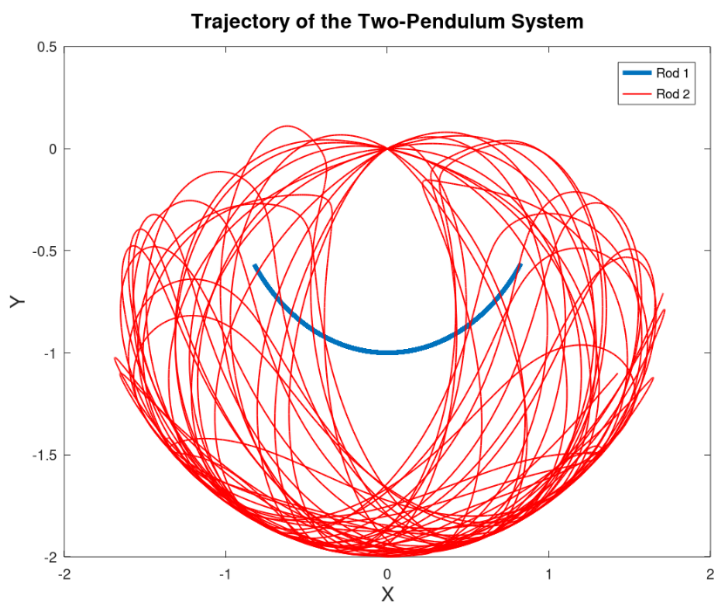


Figure. 5. The trajectory plot for the double pendulum for Case 1.

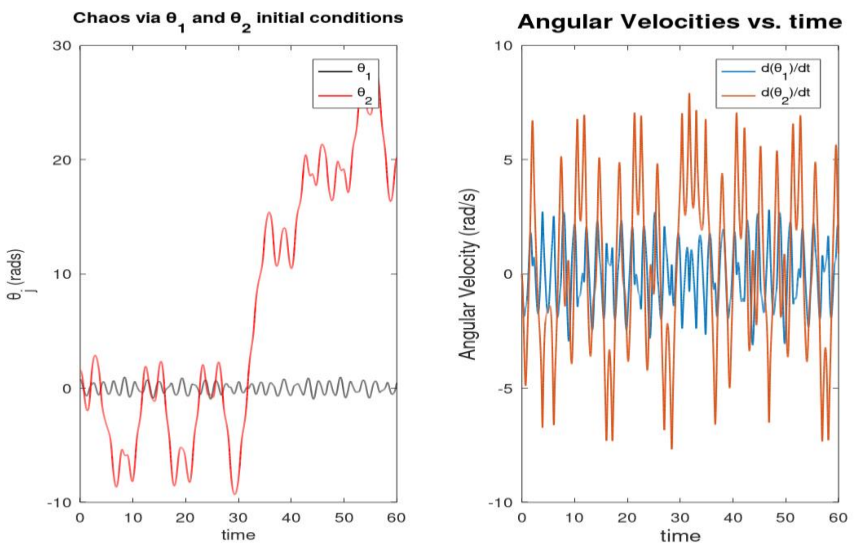


Figure. 6. The changing angles and angular velocities w.r.t time for Case 1.

For Case 1 ($m_1 \gg m_2$), **Figures 3 and 4** demonstrate the phase portraits of each pendulum. Because the contours clumped in the plot, these phase portraits thereby indicate that the system is unstable and follows a wide range of paths, i.e., there is no clear path that the system takes. **Figure. 5.** Takes a look at the trajectory plot of the double pendulum due to $m_1 = 5kg$. As seen, the system behaves chaotically and because of the mass variable that is $m_1 = 5 \times m_2$, there is chaotic motion present as the angular momentum exerted onto rod 2 from rod 1 is larger than the angular momentum exerted from rod 1 onto rod 2. Because of this, rod 1 will keep oscillating with an approximate periodicity while the energy that rod 2 experiences will allow for a noticeable deviation in angles. **Figure. 6** illustrates this precisely in the angular displacements plot where rod 1 is oscillating periodically while rod 2 is chaotic.

5.2. Case 2: The increase in l_2 as $l_2 = 5 \times l_1$ with initial angles $\phi_1 = \pi$ and $\phi_2 = \frac{\pi}{4}$:

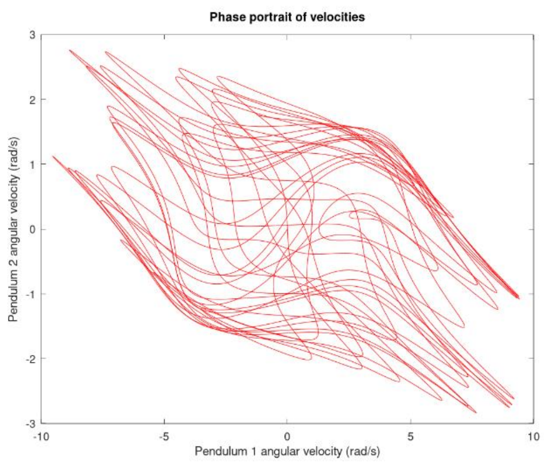


Figure. 7. The phase portrait of the angular velocity for Case 2.

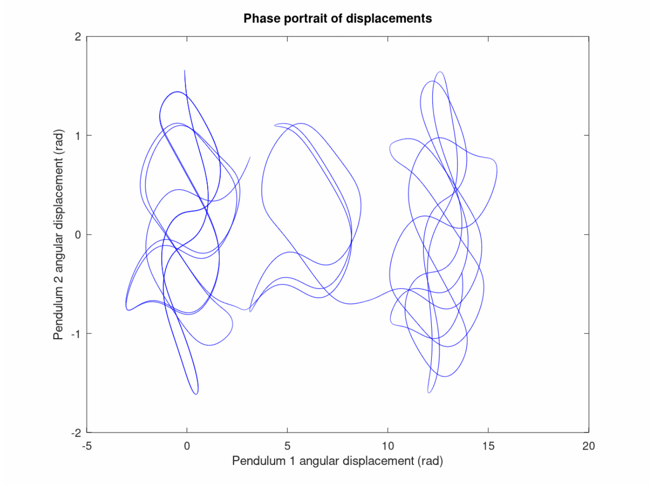


Figure. 8. The phase portrait of the angular displacement for Case 2.

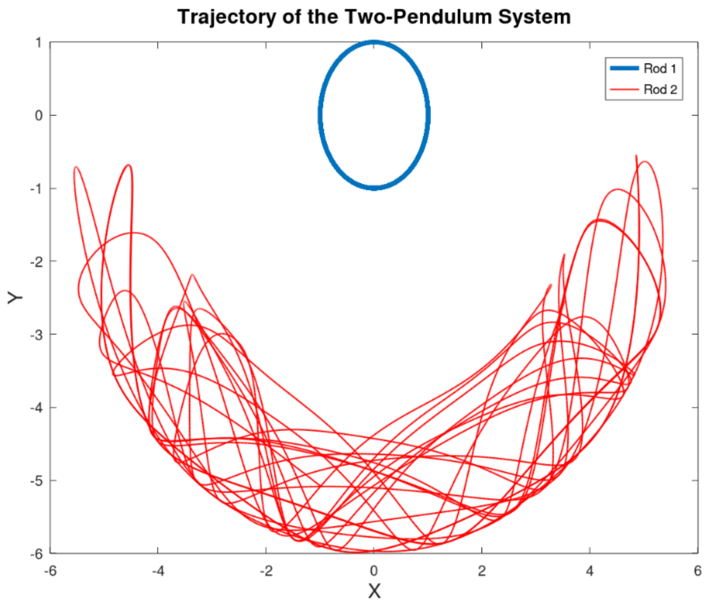


Figure. 9. The trajectory plot for the double pendulum for Case 2.

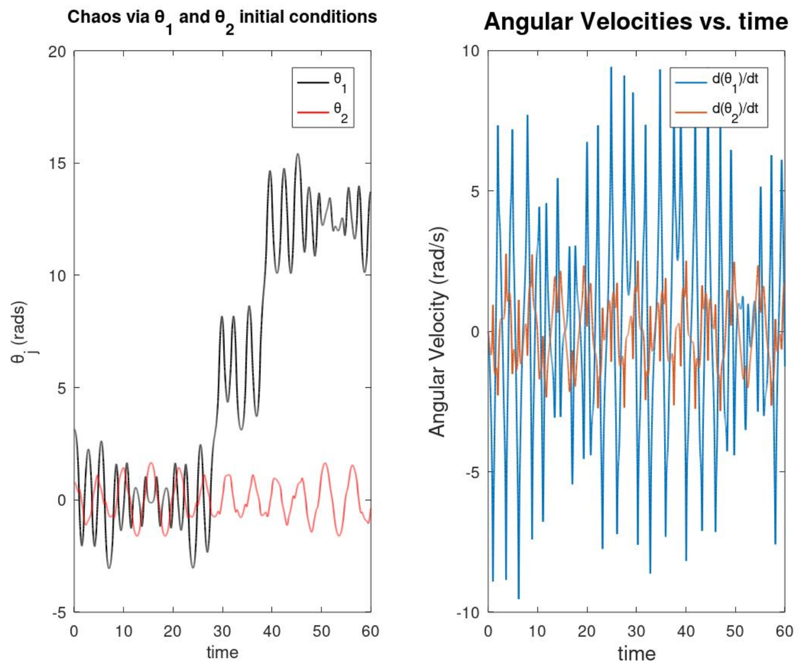


Figure. 10. The changing angles and angular velocities w.r.t time for Case 2.

Figures 7 and 8 show phase portraits for angular displacement and velocity for Case 2, respectively. Because of both figures appearing to be scattered, and there seems to be no correlation between $\dot{\phi}_1$ and $\dot{\phi}_2$ or the angles. We conclude that there is no clear path that the phase portraits follow. Therefore, the system is deemed to be not stable. This is interpreted as the system not desiring these conditions imposed on it due to the unstable phase plots.

When $l_2 > l_1$, theoretically, the second rod would behave like a simple pendulum (see Appendix C for the derivation of a simple pendulum) with no damping where rod 2 and rod 1 have minuscule effects on each other. In **Figure 9,** Case 2, the smaller length l_1 will necessarily behave as an extended arm to the second pendulum, creating a simple pendulum case. The longer l_2 is compared to l_1, the more this simple pendulum will form, and fewer fluctuations will be seen in the trajectory plot. Rod 2 was behaving periodically between the initial condition of $\phi_{initial} = \frac{\pi}{2} rads$. **Figure. 10.** shows the behavior of angular velocities over time. For the second rod, the similarities to the simple undamped pendulum are apparent. In contrast, the first rod shows slight variations due to the second rods swing, having a noticeable effect on the swing of the first rod. This swing was also due to the initial condition imposed initially of $\phi_1 = \pi$ that started motion of the first rod upright.

5.3. Case 3: The increase in m_2 as $m_1 = 3kg$ with damping constants $k_1 = 0.999$ and $k_2 = 1$ at play.

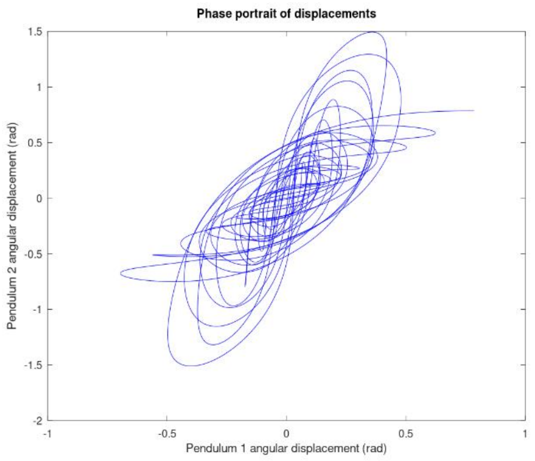


Figure. 11. The phase portrait of the angular displacement for Case 3.

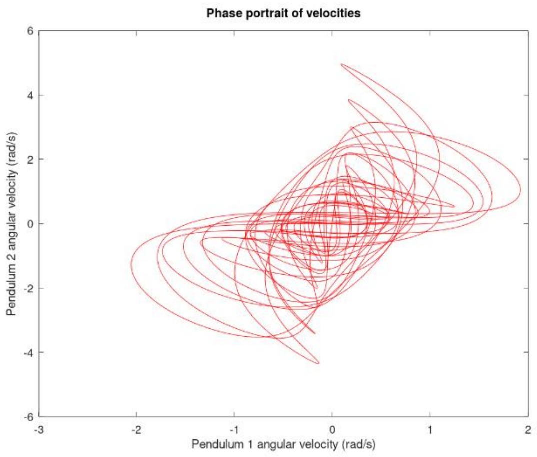


Figure. 12. The phase portrait of the angular velocity for Case 3.

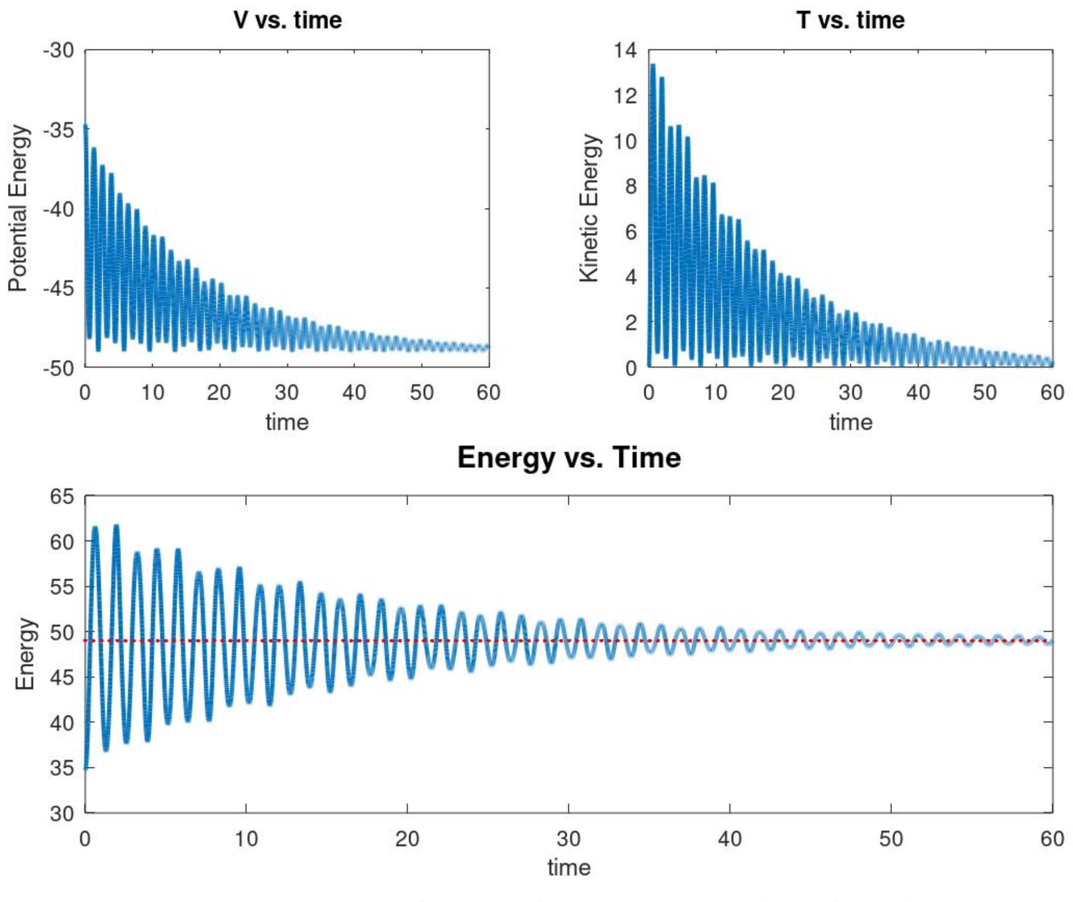


Figure. 13. The Kinetic, Potential and total Energy vs. time plots where the energy is decreasing with time due to $k_1 = 0.999$ for Case 3.

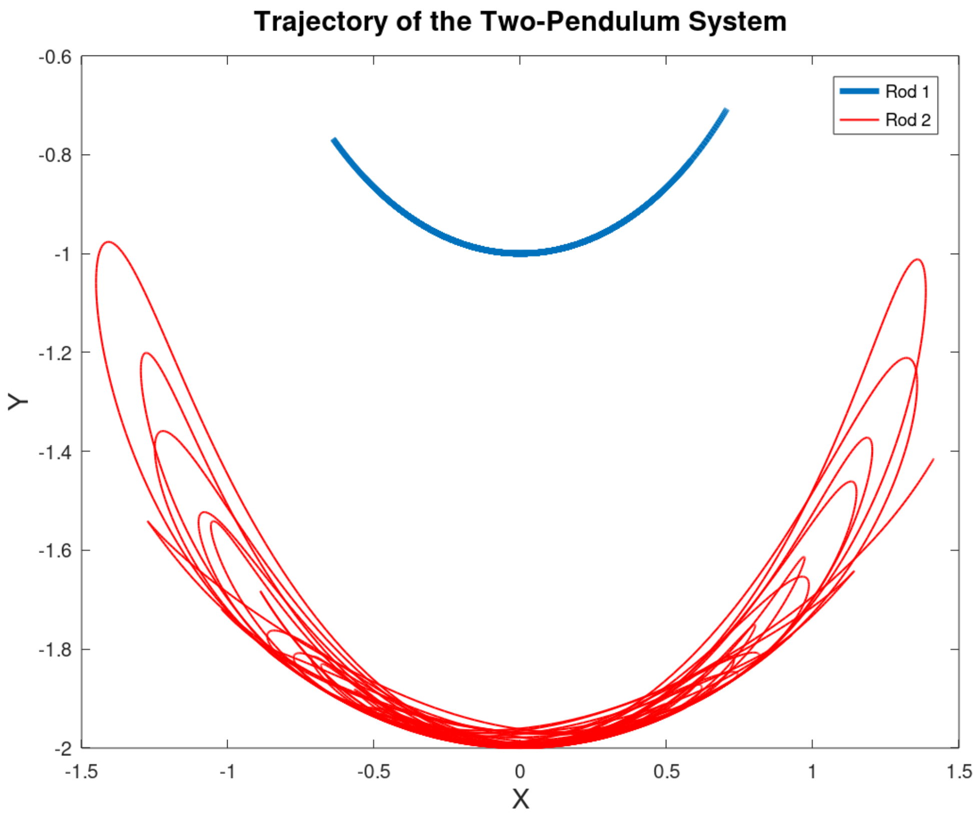


Figure. 14. The trajectory plot for the double pendulum for Case 3.

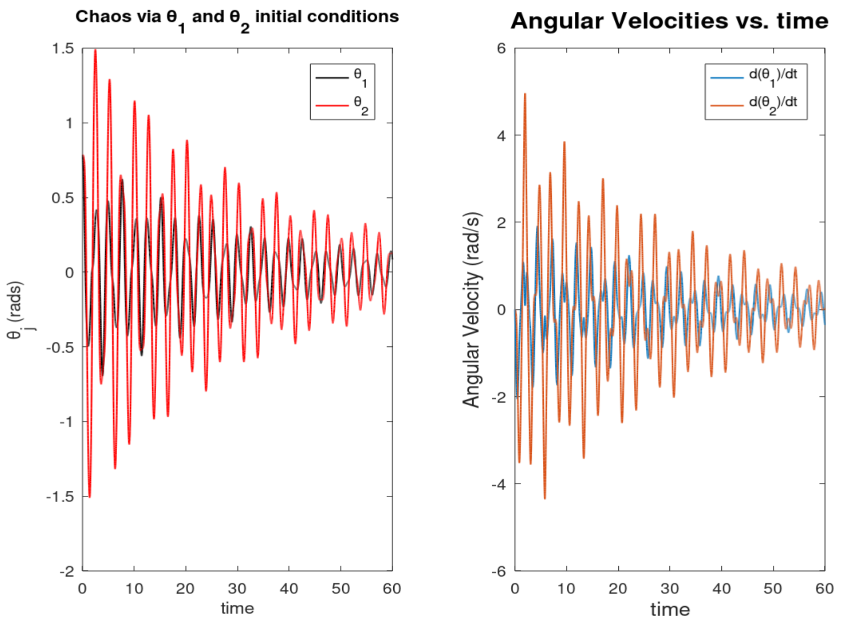


Figure. 15. The changing angles and angular velocities w.r.t time for Case 3, damping k_1.

Both phase portraits in **Figures 11 and 12.** Are centered about the origin and the concentric curves that are present get thicker and denser as $(x, y) \rightarrow (0, 0)$. Therefore, these types of phase-portraits for angular velocity and displacement gives information on a damping system. **Figure. 13** contains the total energy over time plot for the damped system (bottom of **Figure. 13.**). This plot is reducing with an increase in time due to the damping factors at play here, $k_1 = 0.999$. **Figure. 14** shows the trajectory of pendulum 1 and 2, where rod 2 is damped, and rod 1 is undamped. Because of this damping, the trajectory of rod 2 becomes less spread despite the mass of rod 1, forcing rod 2 to be propelled, creating wide curves in the sides of the trajectory plot. Ultimately, the system is damped, and this is mainly seen further in **Figure. 15,** as the angular velocity and displacements are drastically being reduced over time.

3.4 Case 4: The damping constants, k_1 *and* $k_2 = 0.999$ was introduced to demonstrate the damping of the whole system.

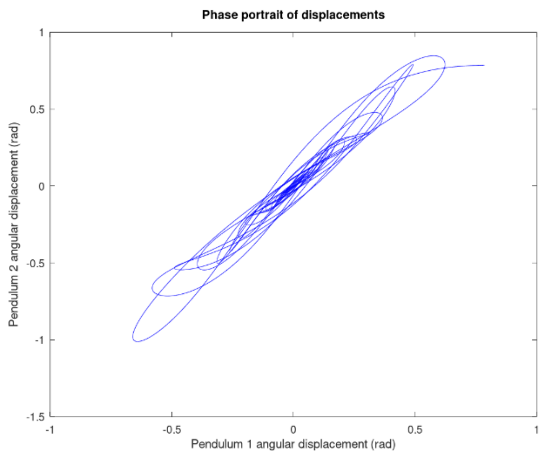


Figure. 16. The phase portrait of the angular displacement for Case 4.

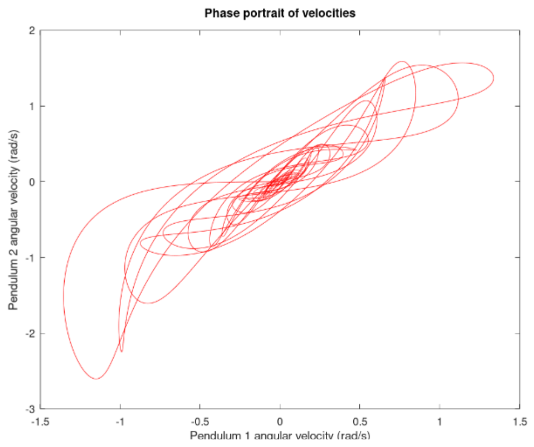


Figure. 17. The phase portrait of the angular velocity for Case 4.

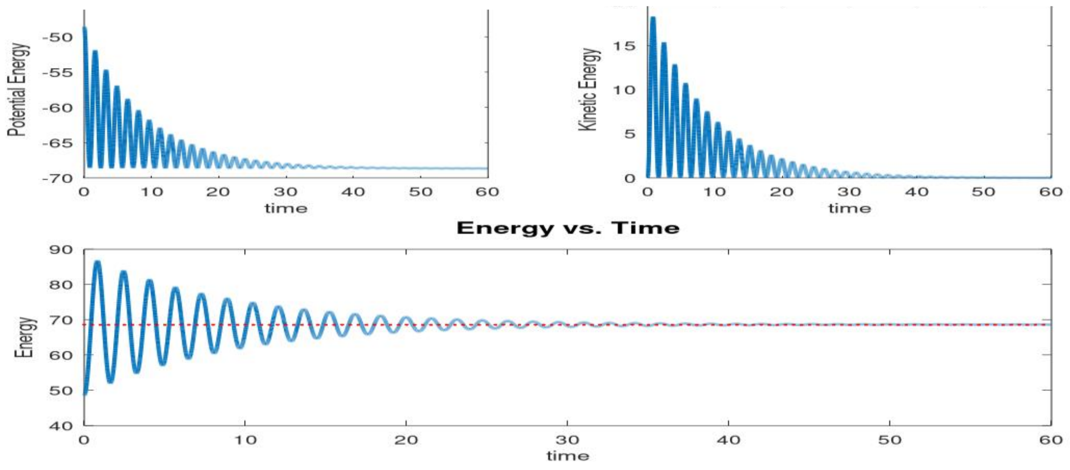


Figure. 18. The Kinetic, Potential and total Energy vs. time plots where the energy is decreasing with time due to $k_{1,2} = 0.999$ for Case 4.

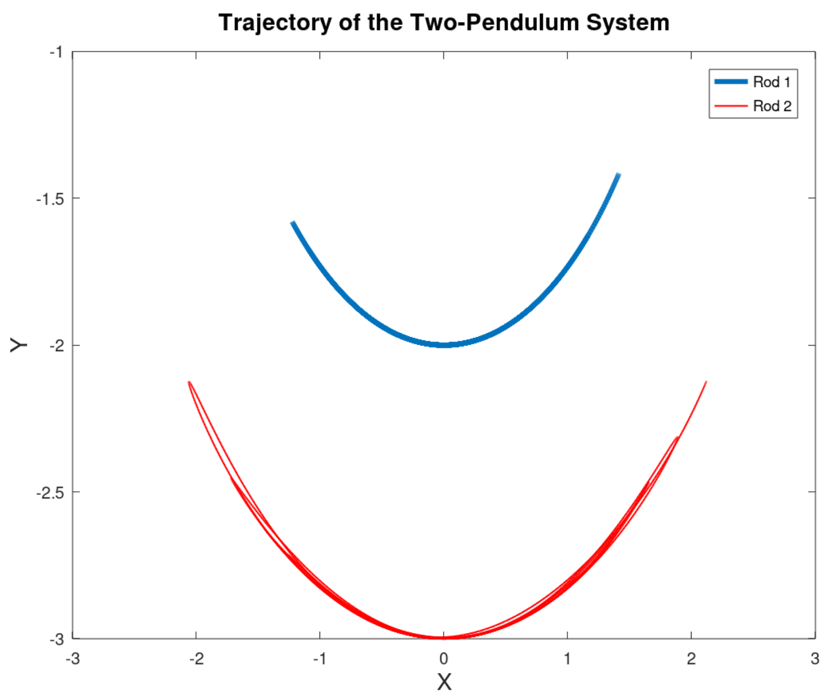


Figure. 19. The Trajectory of the two-pendulum system while damping constants, $k_{1,2} = 0.999$ are at play in Case 4.

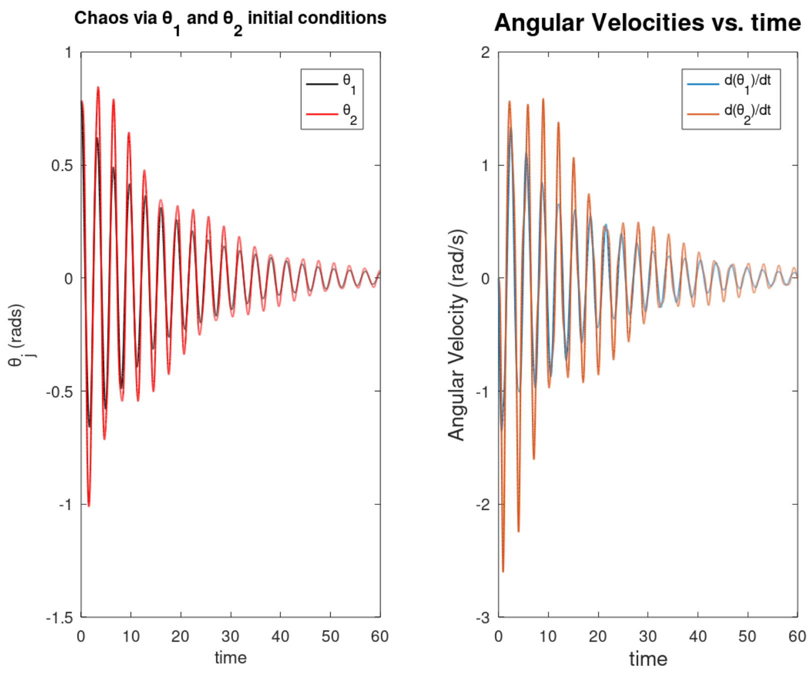


Figure. 20. The changing angles and angular velocities w.r.t time for Case 4, with damping constants, $k_{1,2} = 0.999$.

In Case 4, **Figures 16 and 17** show the phase portraits of the damped system and the behavior is to be as expected, where the phase portraits oscillate around the $(x, y) = (0,0)$ point, much like Case 3. However, the effect of damping is more dramatic here as another damping constant was introduced. There are fewer curves outside of the localized area that is at the origin, effectively making this solution converge to zero and not wandering around. From the phase portrait figures, it is gathered that the preferred state of this system with these conditions is $(\dot{\phi}_1, \dot{\phi}_2, \phi_1, \phi_2) = 0$. Further, **Figure. 18** is a demonstration of the effects of k_1 and k_2 as the total energy of the system converges to the mean value. The trajectories of the pendula, as seen in **Figure. 19,** are hindered due to the damping constants, and it is far more like a simple pendulum analogous to Case 3. The angular velocities and displacements over time converge to zero. The whole system is brought to a complete stop. This stop is to be expected and verifies the case as damping the angular velocity means to reduce it as defined in the script.

There are limitations in this numerical simulation, one such restriction being the step-size. By using the Runge-Kutta of the fourth-order, the solution gets more accurate for a decrease in step-size. That is if $h = 10^{-2}$, an RK4 algorithm will provide a reasonable solution to an order of 10^{-8}. This algorithm can be improved to that of an RK8 algorithm or even higher. However, it is far more challenging to implement into the code.

On another note, the processing power of the computer used to run simulations is limited, such that small step-sizes will take quite a while to implement into the code. If the processor of the computer that had run the script was improved such that it would work with extremely low step-sizes for h, the accuracy of the simulation would be improved drastically.

Conclusions:

In conclusion, the RK4 algorithm was implemented into the double pendulum simulation used in this paper. The equations of motion were derived using the Lagrangian Formalism, and the differential equations were then uncoupled (see Appendix A for derivation). This analysis displayed the simulation of the double pendulum system along with respective plots that further helped describe the initial conditions and variables imposed upon it. This code was then validated through the use of the analysis of results to make sure they match up logically, as there is no clear analytical solution for this system. Finally, it is in hopes of the author that the simulation may help develop an understanding of the double pendulum system and chaotic motion for the reader.

Acknowledgements:

I would like to acknowledge my tutors: Martin Bell, Fabio Henriques, and Oscar Nieves, for continuous support and guidance with the MATLAB programming language by providing me with YouTube videos and Lecture information on the matter and help with debugging code when an issue arose.

References:

[1] Fractalfoundation.org. 2020. *What Is Chaos Theory? – Fractal Foundation*. [online] Available at: <https://fractalfoundation.org/resources/what-is-chaos-theory/> [Accessed 2 June 2020].

[2] Svirin, A., 2020. *Double Pendulum*. [online] Math24. Available at: <https://www.math24.net/double-pendulum/> [Accessed 2 June 2020].

[3] Mathstools.com. 2020. *Runge-Kutta Methods - Solving ODE Problems - Mathstools*. [online] Available at: <https://www.mathstools.com/section/main/Runge_Kutta_Methods> [Accessed 1 June 2020].

[4] Neumann, E., 2016. *Myphysicslab Double Pendulum*. [online] Myphysicslab.com. Available at: <https://www.myphysicslab.com/pendulum/double-pendulum-en.html> [Accessed 1 June 2020].

[5] Bourne, M., 2018. *12. Runge-Kutta (RK4) Numerical Solution For Differential Equations*. [online] Intmath.com. Available at: <https://www.intmath.com/differential-equations/12-runge-kutta-rk4-des.php> [Accessed 2 June 2020].

[6] Maths.surrey.ac.uk. 2020. *THE SIMPLE PENDULUM*. [online] Available at: <http://www.maths.surrey.ac.uk/explore/michaelspages/documentation/Simple> [Accessed 1 June 2020].

Appendix A – Dynamics of the Double Pendulum System:

Writing the coordinates for x and y in terms of the variables for the pendulum's motion:

$$x_1 = l_1 \sin(\phi_1)$$

$$y_1 = -l_1 \cos(\phi_1)$$

$$x_2 = l_1 \sin(\phi_1) + l_2 \sin(\phi_2)$$

$$y_2 = -l_1 \cos(\phi_1) - l_2 \cos(\phi_2)$$

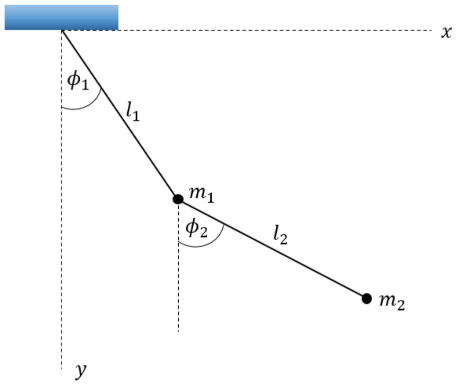


Figure 2. A diagram of the double pendulum system.

We obtain the relative velocities by differentiating x, y positions with respect to time (ϕ = ϕ(t)):

$$\dot{x}_1 = \dot{\phi}_1 l_1 \cos(\phi_1)$$

$$\dot{y}_1 = \dot{\phi}_1 l_1 \sin(\phi_1)$$

$$\dot{x}_2 = \dot{\phi}_1 l_1 \cos(\phi_1) + \dot{\phi}_2 l_2 \sin(\phi_2)$$

$$\dot{y}_2 = \dot{\phi}_1 l_1 \sin(\phi_1) + \dot{\phi}_2 l_2 \sin(\phi_2)$$

Then the kinetic energy, T, is:

$$T = \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2 = \frac{1}{2} m_1 (\dot{x}_1^2 + \dot{y}_1^2)^2 + \frac{1}{2} m_2 (\dot{x}_2^2 + \dot{y}_2^2)$$

Substituting,

$$T = \frac{1}{2} m_1 [\dot{\phi}_1^2 l_1^2 (\cos^2(\phi_1) + \sin^2(\phi_2))] + \frac{1}{2} m_2 [\left(\dot{\phi}_1 l_1 \cos(\phi_1) + \dot{\phi}_2 l_2 \sin(\phi_2)\right)^2$$
$$+ \left(\dot{\phi}_1 l_1 \sin(\phi_1) + \dot{\phi}_2 l_2 \sin(\phi_2)\right)^2]$$

Then, using trig. Identity of $(\cos^2(\phi_1) + \sin^2(\phi_2)) = 1$;

$$T = \frac{1}{2} m_1 \dot{\phi}_1^2 l_1^2 + \frac{1}{2} m_2 [\dot{\phi}_1^2 l_1^2 (\cos^2 \phi_1 + \sin^2 \phi_2) + \dot{\phi}_2^2 l_2^2 (\cos^2 \phi_2 + \sin^2 \phi_2)$$
$$+ 2 l_1 l_2 \dot{\phi}_1 \dot{\phi}_2 (cos\phi_1 cos\phi_2 + \sin\phi_1 sin\phi_2)]$$

We may further simplify our total kinetic energy to:

$$T = \frac{1}{2} m_1 l_1^2 \dot{\phi}_1^2 + \frac{1}{2} m_2 (l_1^2 \dot{\phi}_1^2 + l_2^2 \dot{\phi}_2^2 + 2 l_1 l_2 \dot{\phi}_1 \dot{\phi}_2 \cos(\phi_1 - \phi_2)$$

The total potential energy is the more trivial case

$$V = gm_1y_1 + gm_2y_2$$

Substituting,

$$V = -(m_1 + m_2)gl_1\cos\phi_1 - m_2gl_2cos\phi_2$$

Using the Lagrangian,

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0$$

We have already stated that we will use ϕ_1 and ϕ_2 as generalized coordinates, q_i. Then, we define $\Delta\phi = \phi_1 - \phi_2$ for compactness,

For equation (1):

$$\phi_1: (m_1 + m_2)l_1\ddot{\phi}_1 + m_2l_2\ddot{\phi}_2\cos(\Delta\phi) + m_2l_2\dot{\phi}_2^2\sin(\Delta\phi) + g(m_1 + m_2)sin\phi_1 = 0$$

Rearranging in terms of $\ddot{\phi}_1$,

$$\ddot{\phi}_1 = \left[-m_2l_2\ddot{\phi}_2\cos(\Delta\phi) - m_2l_2\dot{\phi}_2^2\sin(\Delta\phi) - g(m_1 + m_2)sin\phi_1\right] \times \frac{1}{l_1(m_1+m_2)} \qquad \textbf{(1)}$$

For equation (2):

$$\phi_2: m_2l_2\ddot{\phi}_2 + m_2l_1\ddot{\phi}_1\cos(\Delta\phi) - m_2l_1\dot{\phi}_1^2\sin(\Delta\phi) + m_2gsin\phi_2 = 0.$$

Rearranging in terms of $\ddot{\phi}_2$,

$$\ddot{\phi}_2 = \left[-m_2l_1\ddot{\phi}_1\cos(\Delta\phi) + m_2l_1\dot{\phi}_1^2\sin(\Delta\phi) - m_2gsin\phi_2\right] \times \frac{1}{l_2m_2} \qquad \textbf{(2)}$$

The above equations, **(1)** and **(2)** are describing the motion of the chaotic system for ϕ_1 and ϕ_2 as generalized coordinates. These equations must then be uncoupled such that $\ddot{\phi}_1$ does not depend on $\ddot{\phi}_2$, this is done through the process of substitution:

Now, we must uncouple the equations by substituting **(1)** into **(2)**:

For $\ddot{\phi}_1$,

$$\ddot{\phi}_1 = \frac{1}{l_1(m_1 + m_2)}\Big[(-m_2l_2\cos(\Delta\phi)\big(-m_2l_1\ddot{\phi}_1\cos(\Delta\phi) + m_2l_1\dot{\phi}_1^2\sin(\Delta\phi) - m_2gsin\phi_2\big) \times \frac{1}{l_2m_2} - m_2l_2\dot{\phi}_2^2\sin(\Delta\phi) - g(m_1 + m_2)sin\phi_1\Big]$$

Expanding and simplifying,

$$\ddot{\phi}_1 = \frac{\ddot{\phi}_1 m_2 \cos^2(\Delta\phi)}{m_1+m_2} - \frac{m_2 l_1 \dot{\phi}_1^2 \sin(\Delta\phi)\cos(\Delta\phi)}{m_1+m_2} + \frac{g m_2 \sin(\phi_2)\cos(\Delta\phi)}{l_1(m_1+m_2)} - \frac{m_2 l_2 \dot{\phi}_2^2 \sin(\Delta\phi)}{l_1(m_1+m_2)} - \frac{g}{l_1} sin\phi_1$$

Gathering $\ddot{\phi}_1$ terms and simplifying further, we get:

$$\ddot{\phi}_1 = \left[1 - \frac{m_2\cos^2(\Delta\phi)}{m_1+m_2}\right]^{-1}\left(-\frac{m_2 l_1 \dot{\phi}_1^2 \sin(\Delta\phi)\cos(\Delta\phi)}{m_1+m_2} + \frac{g m_2 \sin(\phi_2)\cos(\Delta\phi)}{l_1(m_1+m_2)} - \frac{m_2 l_2 \dot{\phi}_2^2 \sin(\Delta\phi)}{l_1(m_1+m_2)} - \frac{g}{l_1} sin\phi_1\right) \qquad \textbf{(4)}$$

As for $\ddot{\phi}_2$:

$$\ddot{\phi}_2 = \left[-\ddot{\phi}_1\cos(\Delta\phi) + \dot{\phi}_1^2\sin(\Delta\phi) - \frac{g}{l_1} sin\phi_2\right] \times \frac{l_1}{l_2}$$

Where $\ddot{\phi}_1$ is defined above for $\ddot{\phi}_2$. Therefore, these two equations have been coupled as $\ddot{\phi}_1$ and $\ddot{\phi}_2$ are not dependent on each other. These two differential equations may now be implemented into the Runge-Kutta algorithm.

These two uncoupled equations were for $\ddot{\phi}_1$ and $\ddot{\phi}_2$ were further simplified after a lot of algebra using trigonometric properties and **computational algebra** [4]. The below differential equations were used in the MATLAB code for the RK4 algorithm. However, using equations (3) and (4) would suffice but be less compact:

$$\ddot{\phi}_1 = \frac{(-g(2m_1+m_2)\sin(\phi_1) - m_2 g sin(\phi_1 - 2\phi_2) - 2\sin(\Delta\phi) m_2(\dot{\phi}_2^2 l_2 + \dot{\phi}_1^2 l_1 \cos(\Delta\phi))}{l_1(2m_1+m_2-m_2\cos(2\Delta\phi))} \qquad \textbf{(5)}$$

And,

$$\ddot{\phi}_2 = \frac{(2\sin(\Delta\phi)\times\dot{\phi}_1^2 l_1(m_1+m_2) + g(m_1+m_2)\times\cos(\phi_1) + \dot{\phi}_2^2 l_2 m_2 \times \cos(\Delta\phi)}{l_2(2m_1+m_2-m_2\cos(2\Delta\phi))} \qquad \textbf{(6)}$$

Appendix B – Introducing Damping Constants

The damping constants, k_1 and k_2 were introduced in the code through implementation inside the Runge-Kutta for-loop. k_1 and k_2 are damping constants, holding a value between 0 and 1, where 1 suggests no damping and 0 suggests maximum damping. With iteration inside the for-loop, angular velocity is equal to itself multiplied by its respective damping constant. Damping constants of $k_1, k_2 = 0.999$ were used here as they helped visualize the effect of an underdamped case without critically damping the system at an early stage.

The damping constants incorporated in the MATLAB script:

Below is the implementation of the damping constants in the MATLAB script:

```
alp(i + 1) = alp(i) + (1/6)*(K1alp + 2*K2alp + 2*K3alp + K4alp);
a2p(i + 1) = a2p(i) + (1/6)*(K1a2p + 2*K2a2p + 2*K3a2p + K4a2p);
a1(i + 1) = a1(i) + (1/6)*(K1a1 + 2*K2a1 + 2*K3a1 + K4a1);
a2(i + 1) = a2(i) + (1/6)*(K1a2 + 2*K2a2 + 2*K3a2 + K4a2);
alp(i + 1) = k1*alp(i + 1);
a2p(i + 1) = k2*a2p(i + 1); % Damping the angular velocities.
```

Appendix C – The Simple Pendulum

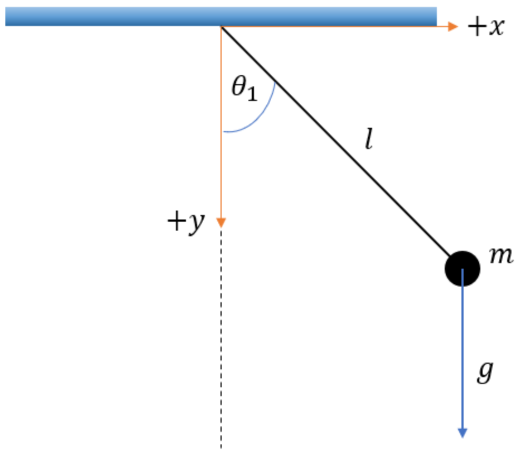
For this simple pendulum case [6] **(Figure. 21.)**, it is evident that the force due to gravity that drives the pendulum's motion is:

$$F = -mgsin(\theta)$$

Then, using Newton's 2nd Law:

$$-mgsin(\theta) = m\ddot{r}$$

$$-gsin(\theta) = \ddot{r}$$



Where $\ddot{r}$ is the acceleration of the pendulum's bob over its radial trajectory. The rod makes an angle with the vertical during its motion that is given by:

Figure. 21. The Simple Pendulum sketch.

$$r = l\theta$$

At any time during its motion. Therefore, differentiating this expression yields:

$$\ddot{r} = l\ddot{\theta}$$

Substituting this radial acceleration into the force equation,

$$-gsin(\theta) = l\ddot{\theta}$$

Therefore,

$$\ddot{\theta} = -\frac{g}{l} sin(\theta)$$

Because this is a second-order ODE that describes the motion of this system, we must separate it using the finite differences on the definition of the derivative:

$$v = \dot{\theta} \tag{1}$$

Therefore,

$$\dot{v} = -\frac{g}{l} \sin(\theta) \tag{2}$$

Then, the Euler-method is introduced here:

$$\frac{v(t + \Delta t) - v(t)}{\Delta t} = -\frac{g}{l} \sin(\theta)$$

Here, we convert Newton's first principle definition of the derivative into a form that is suitable for numerical integration with the Euler-Method after rearranging and making $\Delta t \to 0$ as our time-step.

$$Euler\ for - loop\ implementation \in \begin{cases} v(n+1) = v(n) - \Delta t \frac{g}{l} \sin(\theta) \\ \theta(n+1) = \theta(n) \end{cases}$$

These two arrays may then be input into the for-loop for the Euler-method **(Figure. 22.)**:

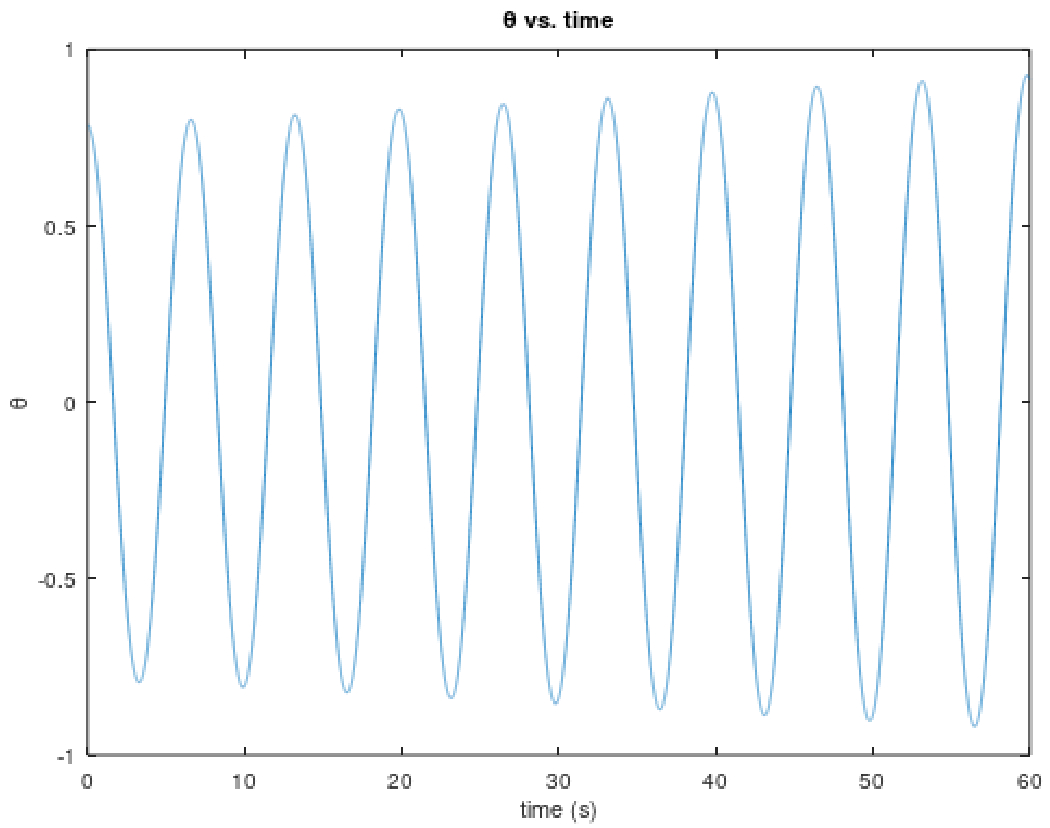


Figure. 22. The angle deviations with respect to time of the simple pendulum. Here, the simple pendulum exhibits an increase in the angle over time and this is not desired.

The Euler-method was then implemented; however, the angle increased with respect to the time when no damping factors had been present. This is due to the order of accuracy that this method provides. Then, an RK4 algorithm was implemented, and the accuracy had increased dramatically with the same step-size, as shown below… **(Figure. 23.)**

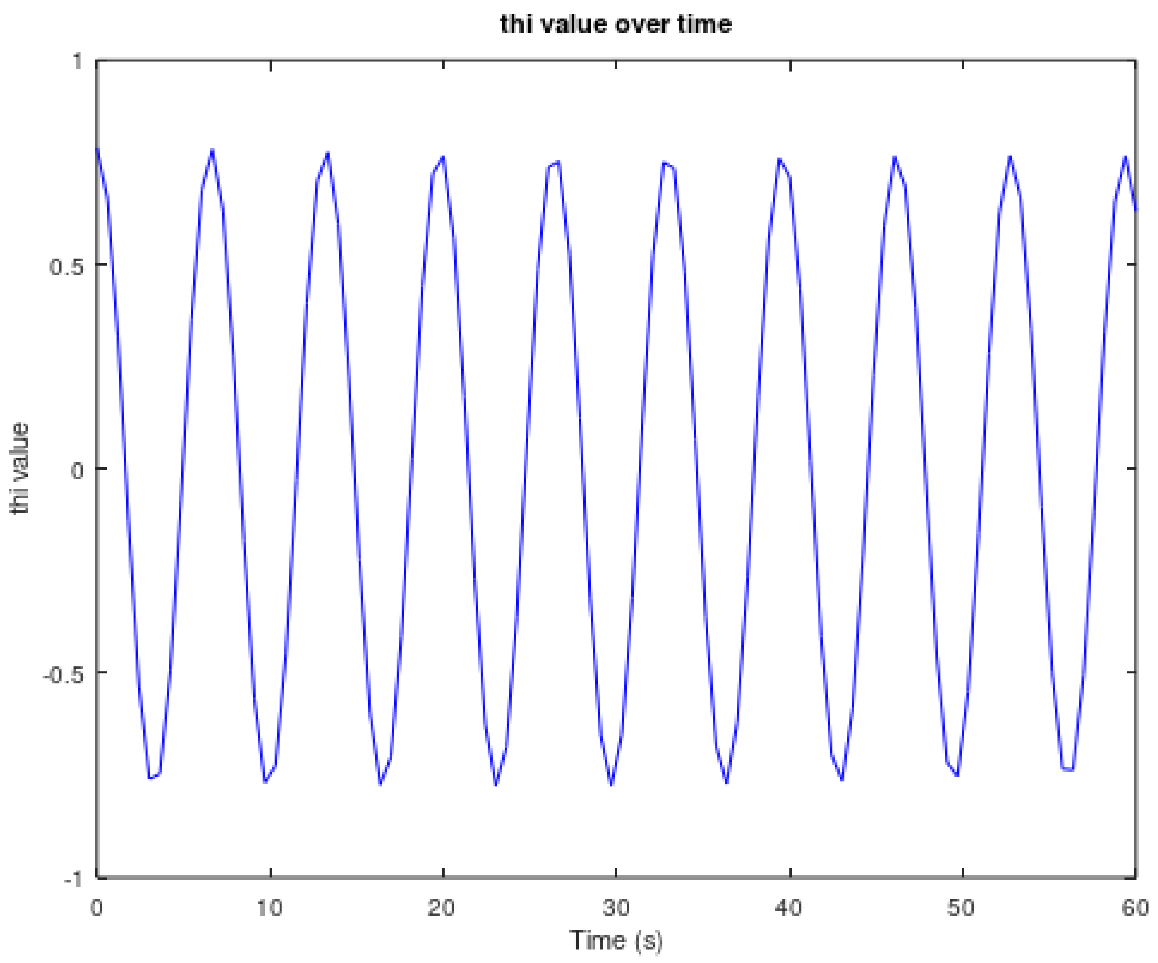


Figure. 23. The Runge-Kutta 4th order algorithm used for the simple pendulum case. This numerical integrator proved to be far more accurate than the Euler-method as it is not a linear integrator like the Euler-method but accurate to orders of 4 magnitudes (10^4).

The damping constant, in the form of k was implemented into the script the same way as it was added for the double pendulum inside the Runge-Kutta for-loop (see Appendix A). The simple pendulum case also had an analytical solution which takes the form:

$$\ddot{\theta} = -\theta_0 \frac{g}{l} \times \cos\left(\sqrt{\frac{g}{l}}t\right)$$

This way, the numerical solution may be validated, and it had been verified with the code provided in Appendix E. However, it may have to be uncommented to see a full comparison.

Appendix D – Activity Log and Project Management

The project spanned over six weeks, as below.

Week	Description
1	This week I tried deriving the equations of motion for a double pendulum, the result was verified with the book, Landau and Lifshitz Classical Mechanics [1] where they did a similar derivation. Knowledge of Classical Mechanics that I had taken last semester was beneficial here as we did have a homework problem relating to the equations of motion of a double pendulum. This problem from last semester helped to justify the derivation as they followed the same procedure. Simultaneously, I was working on the simple pendulum simulation.
2	The simulation of the simple pendulum was commenced using the Euler method. I had run into a problem earlier, however, using the sind function instead of the sin function while inputting the initial angle as $\phi(rads)$ instead of degrees. Also, I figured out that I could choose not to use the movie function, which is a built-in MATLAB animation function, and instead stuck to indexing the results in a for-loop with the plot function and then using the drawnow feature to command MATLAB to make an animation.
3	This week, the simple pendulum works fine with the Euler-method, the RK2 and RK4 methods are implemented this week, and the implementation was quite trivial, it is a numerical integrator that takes moving averages. The damping constant, k was also introduced, and this damping constant was inserted into the code as a number between 0 and 1, this way if the damping constant is 0.99 for instance, the $\phi(i+1)$ value will decrease by a factor of 0.99 after each iteration in the for-loop. Eventually, causing the pendulum to stop at a halt. Furthermore, the num2str function was discovered, and it was inserted into the plot to show the time at each frame in the animation, I made this the title by using "time =" + num2str(t). There was a problem with the angle plot as it was damping with a low time-step. However, this was later resolved as I had made a typo in the code, and this was fixed to have constant angles.
4	The double pendulum was started this week. The simple pendulum code was modified to satisfy the double pendulum. I added two angles in here instead of the single angle for the simple pendulum. I had to verify the equations of motion, and it turned out that they were correct, but I had implemented them incorrectly using the RK4 algorithm, so I started with the RK2 method for the double pendulum simulation in this case. It turns out that you cannot separate functions into multiple lines of code. Otherwise, the system will not behave the way it is supposed to. I had written the function in multiple lines to make the code more compact. However, this was fixed in a later step when I was debugging the code and had tried writing it in a single line. This had got the animation to start working, and I was pleased with the result.
5	After the animation was working for the double pendulum, associated plots were included. Firstly, the angle vs. time plot from the simple pendulum case was included and implementing it for the double pendulum system was trivial because all I had to do was add a hold on command to plot the second angle over time in the same plot to show the angular displacement of the double pendulum system over time. Also, the angular velocities with respect to time plot was added in for more useful information about the system. After all these outputs, the damping constants for the two-pendulum system were added in a similar way to that of the damped simple pendulum case but this time reducing the angular velocities of ϕ_1 and ϕ_2 separately, this essentially makes one of the rods damp more or damp equally. I did this mainly for the second rod as it will be traveling faster if it has a longer arm length.

6	This week, I started writing up the report, including validity checks for the simulation. One such validity check was that for total Energy vs. Time, the total energy should be constant across the entire time array as the kinetic and potential energies. Of course, there were small deviations from the actual constant Energy value of E = T + V due to background noise and the limitations of the time-step due to the computational processing power. Comments were added onto the code this week also to help a new user run the code and overall make it look more presentable.

Appendix E – Full Code for the Pendula Systems that were studied:

E1 – The Euler Method for the Simple Pendulum:

```
##%% Computational Physics: Simple Pendulum using pause() in a for-loop.

clc; close all; clear all;


t = linspace(0, 60, 10000); % This is time unit with 3600 values from 0 to 60.

dt = t(2) - t(1)

x = zeros(length(t), 1);

y = zeros(length(t), 1);

l = 10; % Length of stick (mass holders).

g = 9.81; % Gravitational acceleration on Earth.

m = 1; % The mass is a point mass and it is 1 unit.

initial_thi = 45*(pi/180); % Specify an initial thi value in degrees.

thi = linspace(-45, 45, length(t))

thi(1) = initial_thi;

alpha(1) = 0; % Angular velocity.

omega(1) = 0; % Angular acceleration.


for i = 1:length(t) - 1

  alpha(i + 1) = alpha(i) - (g/l)*sin(thi(i))*dt;

  thi(i + 1) = thi(i) + dt*alpha(i);

  x = l*sin(thi);

  y = -l*cos(thi);

end


plot(t, thi) % This method was implemented. However, it did not produce a desired result.

xlabel('time (s)')

ylabel('\theta')

title('\theta vs. time')
```

E2 – The RK4 Algorithm for the Damped and Undamped SIMPLE Pendulum:

```
%% RK4 Method (alpha, theta in single loop) seperate loop for plotting after


clear all; clc; close all;


t = linspace(0, 60, 100); % This is time term, may increase or decrease speed.
dt = t(2) - t(1)
x = zeros(length(t), 1);
y = zeros(length(t), 1);
l = 10; % Length of stick (mass holders).
g = 9.81; % Gravitational acceleration on Earth.
m = 1; % The mass is a point mass and it is 1 unit.
initial_thi = 45*(pi/180); % Specify an initial thi value in degrees.
thi(1) = initial_thi;
v = zeros(length(t), 1); % The alpha value which is d(thi)/dt.
v(1) = 0 % alpha is the angular velocity.


% RK4 functions
f1 = @(t, thi, v) v; % This is the saying d(thi)/dt = v
f2 = @(t, thi, v) -g/l * sin(thi); % v = secondDerivative of thi...
% f2 = @(t, thi, v) -initial_thi*g/l * cos(sqrt(g/l)*t); IS THE ANALYTICAL SOLUTION.
h = dt; % dt = t(2) - t(1) from previous line of code.


k = 1; % Damping constant (between 0 and 1), aka. \mu.
% RK4 loop and animation
for i = 1:length(t) - 1
  K1thi = f1(t(i), thi(i), v(i));
  K1v = f2(t(i), thi(i), v(i));
  K2thi = f1(t(i) + h/2, thi(i) + h*K1thi/2, v(i) + h*K1v/2);
  K2v = f2(t(i) + h/2, thi(i) + h*K1thi/2, v(i) + h*K1v/2);
  K3thi = f1(t(i) + h/2, thi(i) + h*K2thi/2, v(i) + h*K2v/2);
  K3v = f2(t(i) + h/2, thi(i) + h*K2thi/2, v(i) + h*K2v/2);
  K4thi = f1(t(i) + h, thi(i) + h*K3thi, v(i) + h*K3v);
  K4v = f2(t(i) + h, thi(i) + h*K3thi, v(i) + h*K3v);
  thi(i + 1) = thi(i) + h/6 * (K1thi + 2*K2thi + 2*K3thi + K4thi);
  v(i + 1) = v(i) + h/6 * (K1v + 2*K2v + 2*K3v + K4v);
  v(i + 1) = k*v(i + 1); % Damped simple Pendulum, set k = 1 if undamped is wanted.
end

%% Figure. 2. shows the varying thi value w.r.t time.
figure(1)
plot(t, thi, 'b');
xlabel('Time (s)')
ylabel('thi value')
title('thi value over time')

for i = 1:length(t) - 1
  % The animation of the simplePendulum is as follows...
  x = l*sin(thi);
  y = -l*cos(thi);
  figure(2)
  plot(l*sin(thi(i)), -l*cos(thi(i)), 'ko', 'MarkerSize', 15); % It is a uniform mass with size 15.
  hold on
  line([0 l*sin(thi(i))], [0 -l*cos(thi(i))]) % Generates a line, gave it the i-th value of...
  % ... the array to make a new line at every point for the animation.
  plot(0, 0, 'ro') % Demonstrating it is in fact a fixed origin.
  hold off
  grid on
  xlim([-l-1 l+1])
  ylim([-l-1 l+1])
  title(['t = ' num2str(t(i)) ', \mu = ' num2str(k)]);
  drawnow;
end
```

E3 – The RK4 Algorithm for the Damped and Undamped Double Pendulum:

```
clc; clear all; close all;


% Variables:

r1 = 1; % Length of pendulum 1

r2 = 1; % Length of pendulum 2

m1 = 1; % Mass on pendulum 1

m2 = 1; % Mass on pendulum 2

g = 9.81;


dt = 0.009;

t = [0:dt:60];

a1 = zeros(1, length(t));

a2 = zeros(1, length(t));


a1(1) = 45*(pi/180);

a2(1) = 45*(pi/180); % {a1, a2} = pi rads (upside down) for good example of E.

a1p = zeros(1, length(t));

a2p = zeros(1, length(t));

a1p(1) = 0;

a2p(1) = 0;

% The two equations for angular acceleration; fun_1 and fun_2:

fun_1 = @(t, a1p, a2p, a1, a2) (-g*(2*m1 + m2) * sin(a1) - m2*g*sin(a1 - 2*a2) -2*sin(a1-a2)*m2*(a2p*a2p * r2 + a1p*a1p*r1*cos(a1-
a2)))/(r1*(2*m1 + m2 - m2 * cos(2*a1-2*a2)));

fun_2 = @(t, a1p, a2p, a1, a2) (2*sin(a1-a2)*(a1p.*a1p * r1*(m1+m2) + g*(m1 + m2)*cos(a1)+a2p.*a2p * r2*m2*cos(a1-a2)))/(r2 * (2*m1 +
m2-m2*cos(2*a1-2*a2)));

F1 = @(t, a1p, a2p, a1, a2) a1p;

F2 = fun_1;

G1 = @(t, a1p, a2p, a1, a2) a2p;

G2 = fun_2;


h = dt; % For the RK2 loop:

k1 = 1; % Damping constant, k_1 for angular velocity 1. ki -> 0 is more damping.

k2 = 1; % Damping constant, k_2 for angular velocity 2.

for i = 1:(length(t)-1)

   K1a1 = h*F1(t(i), a1p(i), a2p(i), a1(i), a2(i));
```

```
    K1a1p = h*F2(t(i), a1p(i), a2p(i), a1(i), a2(i));
    K1a2 = h*G1(t(i), a1p(i), a2p(i), a1(i), a2(i));
    K1a2p = h*G2(t(i), a1p(i), a2p(i), a1(i), a2(i));

    K2a1 = h*F1(t(i) + h/2, a1p(i) + K1a1p/2, a2p(i) + K1a2p/2, a1(i) + K1a1/2, a2(i) + K1a2/2);
    K2a1p = h*F2(t(i) + h/2, a1p(i) + K1a1p/2, a2p(i) + K1a2p/2, a1(i) + K1a1/2, a2(i) + K1a2/2);
    K2a2 = h*G1(t(i) + h/2, a1p(i) + K1a1p/2, a2p(i) + K1a2p/2, a1(i) + K1a1/2, a2(i) + K1a2/2);
    K2a2p = h*G2(t(i) + h/2, a1p(i) + K1a1p/2, a2p(i) + K1a2p/2, a1(i) + K1a1/2, a2(i) + K1a2/2);

    K3a1 = h*F1(t(i) + h/2, a1p(i) + K2a1p/2, a2p(i) + K2a2p/2, a1(i) + K2a1/2, a2(i) + K2a2/2);
    K3a1p = h*F2(t(i) + h/2, a1p(i) + K2a1p/2, a2p(i) + K2a2p/2, a1(i) + K2a1/2, a2(i) + K2a2/2);
    K3a2 = h*G1(t(i) + h/2, a1p(i) + K2a1p/2, a2p(i) + K2a2p/2, a1(i) + K2a1/2, a2(i) + K2a2/2);
    K3a2p = h*G2(t(i) + h/2, a1p(i) + K2a1p/2, a2p(i) + K2a2p/2, a1(i) + K2a1/2, a2(i) + K2a2/2);

    K4a1 = h*F1(t(i) + h, a1p(i) + K3a1p, a2p(i) + K3a2p, a1(i) + K3a1, a2(i) + K3a2);
    K4a1p = h*F2(t(i) + h, a1p(i) + K3a1p, a2p(i) + K3a2p, a1(i) + K3a1, a2(i) + K3a2);
    K4a2 = h*G1(t(i) + h, a1p(i) + K3a1p, a2p(i) + K3a2p, a1(i) + K3a1, a2(i) + K3a2);
    K4a2p = h*G2(t(i) + h, a1p(i) + K3a1p, a2p(i) + K3a2p, a1(i) + K3a1, a2(i) + K3a2);

    a1p(i + 1) = a1p(i) + (1/6)*(K1a1p + 2*K2a1p + 2*K3a1p + K4a1p);
    a2p(i + 1) = a2p(i) + (1/6)*(K1a2p + 2*K2a2p + 2*K3a2p + K4a2p);
    a1(i + 1) = a1(i) + (1/6)*(K1a1 + 2*K2a1 + 2*K3a1 + K4a1);
    a2(i + 1) = a2(i) + (1/6)*(K1a2 + 2*K2a2 + 2*K3a2 + K4a2);
    a1p(i + 1) = k1*a1p(i + 1);
    a2p(i + 1) = k2*a2p(i + 1); % Damping the angular velocities.
end

% Define {X1, Y1, X2, Y2} for the plots below as the whole array of {a1, a2}:
X1 = r1 * sin(a1);
Y1 = -r1 * cos(a1);
X2 = r1 * sin(a1) + r2 * sin(a2);
Y2 = -r1 *cos(a1) - r2 * cos(a2);

% Important plots:
figure(1) % Displays the trajectory over the tspan.
plot(X1, Y1, 'LineWidth', 3) % The whole of x1, y1.
```

```
hold on
plot(X2, Y2, 'r', 'LineWidth',1) % The whole of x2, y2.
hold off
xlabel('X', 'FontSize', 14);
ylabel('Y', 'FontSize', 14);
title('Trajectory of the Two-Pendulum System', 'FontSize',14)
legend('Rod 1', 'Rod 2')

figure(2)
subplot(1, 2, 1)
plot(t, a1(:), 'k', 'LineWidth', 1)
hold on
plot(t, a2(:), 'r', 'LineWidth', 1) % Plot ang. displacement arrays over time
hold off
title('Chaos via \theta_1 and \theta_2 initial conditions');
legend('\theta_1', '\theta_2')
xlabel('time');
ylabel('\theta_j (rads)');

% Angular velocity test over the entire time-span:
subplot(1, 2, 2)
plot(t, a1p, 'LineWidth', 1)
hold on
plot(t, a2p, 'LineWidth', 1)
hold off
xlabel('time', 'FontSize', 13)
ylabel('Angular Velocity (rad/s)', 'FontSize', 13)
legend('d(\theta_1)/dt', 'd(\theta_2)/dt')
title('Angular Velocities vs. time', 'FontSize', 15)
% Energy Plot to test accuracy (should be constant Energy w/no damping case):
T_1 = (1/2)*m1*r1*r1*a1p.*a1p;
T_2 = (1/2)*m2*(r1*r1*a1p.*a1p + r2*r2*a2p.*a2p + 2*r1*r2*a1p.*a2p.*cos(a1-a2));
T = T_1 + T_2; % Kinetic Energy
V = -(m1 + m2)*g*r1*cos(a1) - m2*g*r2*cos(a2); % Potential Energy
E = abs(T) + abs(V);
avg_E = mean(E)
```

```
figure(4)
subplot(2, 2, [3, 4])
plot(t, E, 'LineWidth', 2)
xx = [0:0.5:t(end)]; % Because the yline function does not work on Octave...
hold on
plot(xx, avg_E, 'r', 'LineWidth', 1) % Should average out to approx. 0, minimizing noise.
hold off
xlabel('time')
ylabel('Energy')
title('Energy vs. Time', 'FontSize', 13.5)
subplot(2, 2, 1)
plot(t, V, 'LineWidth', 2)
xlabel('time')
ylabel('Potential Energy')
title('V vs. time')
subplot(2, 2, 2)
plot(t, T, 'LineWidth', 2)
xlabel('time')
ylabel('Kinetic Energy')
title('T vs. time')
figure(6)
plot(a1, a2, 'b')
title('Phase portrait of displacements')
xlabel('Pendulum 1 angular displacement (rad)')
ylabel('Pendulum 2 angular displacement (rad)')
figure(7)
plot(a1p, a2p, 'r')
title('Phase portrait of velocities')
xlabel('Pendulum 1 angular velocity (rad/s)')
ylabel('Pendulum 2 angular velocity (rad/s)')
 % ctrl + shift + R to uncomment code.
for i = 1:length(t) - 1
  % The co-ordinate system:
  x1 = r1 * sin(a1(i));
  y1 = -r1 * cos(a1(i));
  x2 = r1 * sin(a1(i)) + r2 * sin(a2(i));
  y2 = -r1 *cos(a1(i)) - r2 * cos(a2(i));


  % Begins the figure animation...
  figure(8)
  plot(x1, y1, 'ko', 'MarkerSize', m1*10); % First bob marker swinging.
  hold on
  line([0 x1], [0 y1])
  plot(x2, y2, 'ko', 'MarkerSize', m2*10); % Second bob marker swinging.
  line([x1 x2], [y1 y2])
  plot(0, 0, 'ro') % Demonstrating it is in fact a fixed origin.
  hold off
  grid on
  xlim([-r1-r2-1 r1+r2+1]) % Axis limits
  ylim([-r1-r2-1 r1+r2+1]) % Axis limits
  title(['t = ' num2str(t(i))]);
  drawnow;
end
```