AE341 Modeling and Analysis Lab
Exercise 2

Modeling the Dragon Module as a Pendulum

Ayush Sharma (SC19B008), G. Ramana Bharathi (SC19B028)
Department of Aerospace Engineering
Indian Institute of Space Science and Technology

(Dated: October 11, 2021)

A system of mathematical equations that is used to describe a real-life physical system mathematically is called a mathematical model of the corresponding physical system. In this exercise, we have broken down the process of lifting the Dragon Module into three phases and tried to propose mathematical models for each of the three phases separately. We have also analysed simpler mathematical models for the three phases, and tried to explain the different assumptions involved in the mathematical models proposed, and how we can relax some of these assumptions. For writing the scripts for implementing the numerical methods, we have referred to [1], [2].

NOMENCLATURE

T	Kinetic Energy
V	Potential Energy
E	Total Energy
m	Mass of the Dragon Module
v	Speed of the module
r	Length of the pendulum string
$\mathcal{L}$	Lagrangian of the corresponding system

I. INTRODUCTION

In the first exercise, we tried to roughly determine the dimensions of the A-frame crane and the range of values for the angle the crane makes with the boat's horizontal surface, the angular speed of the frame, and the rate of change of length of the rope connected to the dragon module while in the second exercise we looked at the various commercially available ordinary differential equation solvers, and the different algorithms implemented by these solvers. In this exercise, we have proposed a mathematical model for estimating the position and other parameters of the Dragon module as it is lifted from the ocean surface on to the deck of the GO Navigator.

II. THEORY

II.1. Breaking the overall motion of the capsule down into separate steps

It can be seen from the video of the recovery of the Dragon capsule that the entire process can be broken down into three stages based on the different types of motion of the capsule. The three phases/stages are:

- **Part-1 :** lifting the module floating on water while the A-frame makes a fixed angle with the ship's horizontal surface
- **Part-2 :** rotating the A-frame structure about its base starting from a horizontal position (acute angle of around 30^o with the ship's horizontal surface) to a vertical position(obtuse angle of around 110^o with the ship's horizontal surface)
- **Part-3 :** lowering the crew module so that it rests on the dragon nest on the GO Navigator ship.

We are going to separately consider each of the above mentioned cases and try to look at the different possible mathematical models for the physical problem that needs to be modeled.

II.2. Mathematical Models

Following are the different mathematical models that could be used to simplify the physical system and obtain an approximate mathematical representation of the actual problem.

- **A Simple Pendulum :** The simplest possible situation would be assuming the Dragon module as a point mass and assuming it to oscillate like a bob of a pendulum with constant length in a single plane about the pivot (point on top of the A-frame where the string connected to the capsule is fixed). In such a case, the governing equation for the motion of the capsule with respect to the pivot point is given by:

$$\ddot{\theta} + \frac{g}{r}\sin\theta \;=\; 0$$

where, r is the length of the pendulum. The above equation is non-linear in θ, and it can be solved using any ODE solver on Python, MATLAB, and

other commercial packages. This equation can be further simplified and made analytical when it is assumed that the angle θ is close to zero[1]. For such a case, we have $\sin\theta \approx \theta$, The problem with this model is that it is going to be highly unrealistic and would not represent the actual scenario with great accuracy.

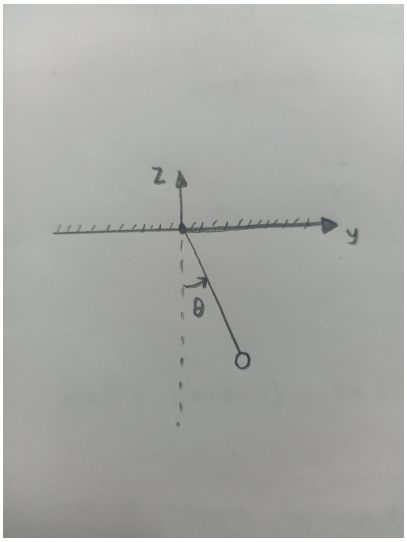


FIG. 1. Simple Pendulum Schematic with the considered reference frame

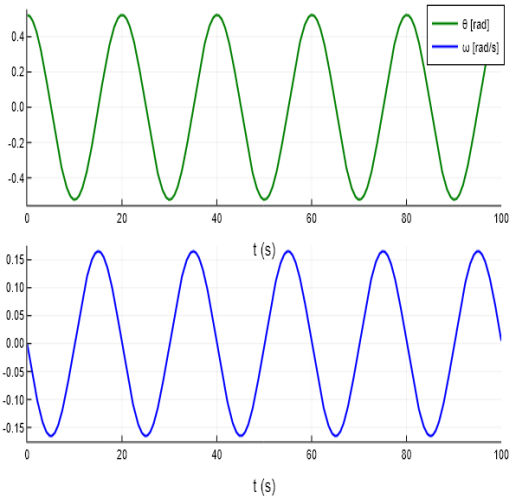


FIG. 2. Solution for θ and ω for the simple pendulum

- **A Simple Pendulum with damping :** This model is a slightly modified version of the first situation, in the sense that it also includes damping, which is a part of almost all real physical systems. This model is a better approximation of the actual scenario than the first case, but it is also not really great since it only considers planar motion. The governing equation for this case(here too we are looking at the motion of the capsule with respect to the fixed pivot point) is given by:

$$\ddot{\theta} + b\dot{\theta} + \frac{g}{r}\sin\theta \;=\; 0$$

where b represents is the damping coefficient. In the earlier situation(first model), the capsule would keep on swinging forever without stopping, whereas in this case, we are considering losses due to friction which will mean that the capsule would eventually come to a stop. Note that the convention for the angle and reference frame is the same for the damped simple pendulum as well.

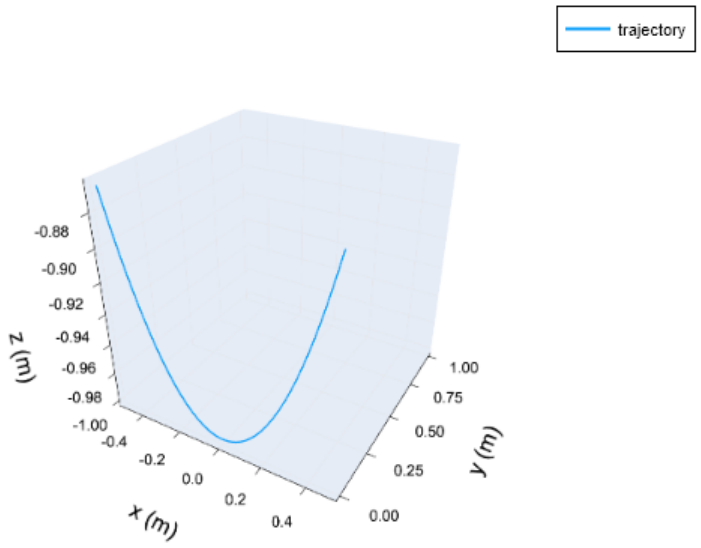


FIG. 3. Simple pendulum trajectory

- **A Simple Pendulum with changing length :** In the previous two cases we considered the length of the pendulum (distance between the bob and the pivot) to be a constant. When we closely look at the video of the capsule's recovery though, we see that in the cases where the pivot is fixed, the length of the rope keeps on changing (slowly and continuously - it is like a quasi-steady process). In this model, we are considering the dragon capsule to resemble the motion of a bob of a pendulum with changing length. The governing equations for the motion of the dragon capsule according to this

[1] i.e., when θ is smaller than around 15°

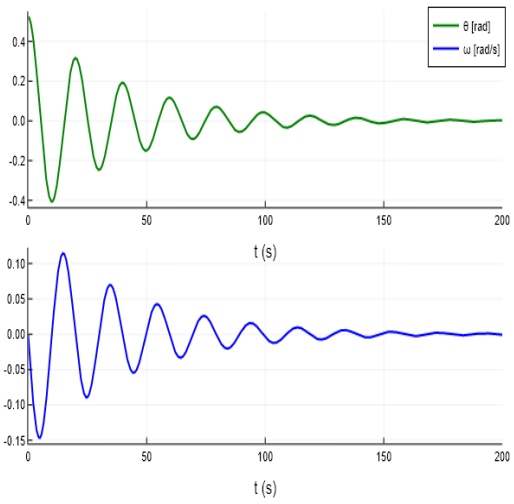


FIG. 4. Plotting the solutions for θ and ω for damped simple pendulum

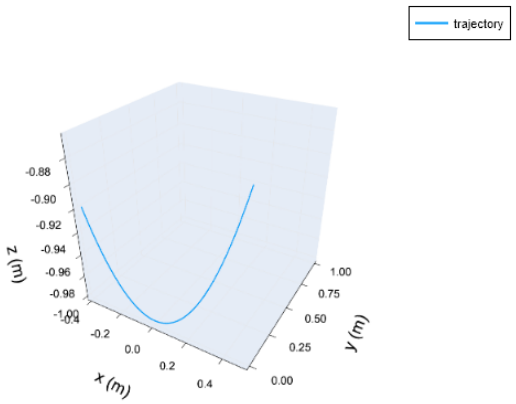


FIG. 5. Simple damped pendulum trajectory

model is given by:

$$\ddot{\theta} = \frac{g}{r}\sin\theta - 2\frac{\dot{\theta}\dot{r}}{r}$$
$$\ddot{r} = r\dot{\theta}^2 - g\cos\theta$$

There will be two equations in case we don't know the velocity in the radial equation, but when we know r(t), we will just have one second order non-linear ODE in θ, which can be converted into a system of 2 first order ODEs and then solved numerically. Just like its predecessors, this case also does not consider the motion of the capsule in three dimensions (it too considers planar motion only).

- **A Spherical Pendulum :** The simplest mathematical model for motion of the Dragon capsule in three dimensions would be represented by the spherical pendulum. In this model, we consider the Dragon capsule to be a point mass, which is going to be the bob of a spherical pendulum with the pivot attached to the top of the A-frame, as shown in the figure. The length of the pendulum is considered to be constant. The governing equations for the motion of the capsule according to this mathematical model are:

$$\ddot{\theta} = \dot{\phi}^2\sin\theta\cos\theta + \frac{g}{r}\sin\theta$$
$$\ddot{\phi} = -2\dot{\theta}\dot{\phi}\cot\theta$$

The above equations are both non-linear second order ODEs. We can solve the system of equations numerically, by converting the two ODEs into a system of 4 first-order ODEs. This mathematical model is going to be relatively more accurate in representing the physical problem when compared to its predecessors, but it does not deal with the fact that the length of the string changes in the first and third parts of the physical system.

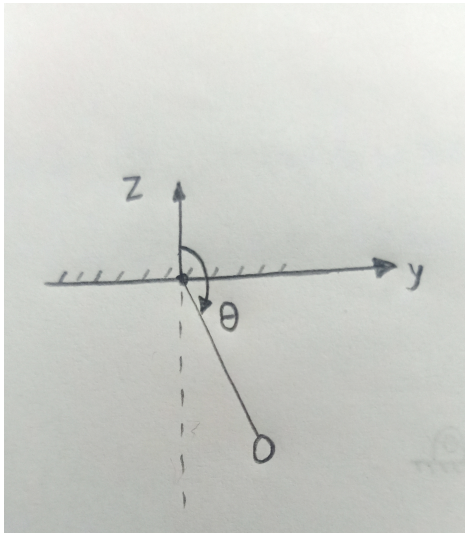


FIG. 6. Simple pendulum with changing length - reference axes and convention for θ

- **A Spherical Pendulum with changing length:** This model would be a very good approximation of the physical problem (for stages 1 and 3 of the recovery process) when considering the Dragon capsule as a point mass. This model has been discussed in greater detail(since this is the model we will be considering for stages 1 and 3) in the later sections of this report.

In the above list, we have mentioned only the models in which we are assuming the Dragon capsule to be a point mass. It is noted that in actuality the capsule cannot be considered as a point object, but that would

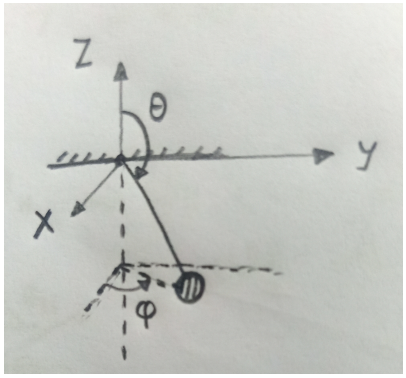


FIG. 7. Reference co-ordinate system and sign convention for the spherical pendulum

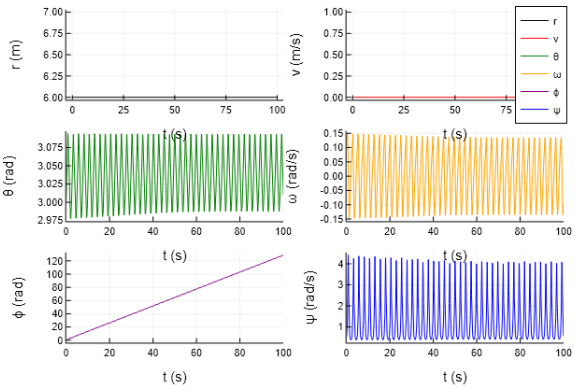


FIG. 8. Plots of r(t), v(t), $\theta\,(t)$, $\omega\,(t)$ for spherical pendulum

increase the complexity of the problem to a great extent and also require much more computation time for solving the governing equations numerically.

Also, for the second stage of the recovery process, we have considered the mathematical model as a spherical pendulum with moving pivot point (to be more specific, the pivot point is accelerating). This case has also been discussed at length later in the report.

II.3. Assumptions

While considering simple mathematical models to represent the actual physical system, we have considered certain assumptions to hold true. Some of these assumptions may or may not be realistic depending upon the physical system considered. We are now going to mention all the assumptions that can possibly be considered while proposing a mathematical model to represent the given physical problem (i.e., we will write the assumptions considered in the simplest possible mathematical model for the problem - the simple pendulum).

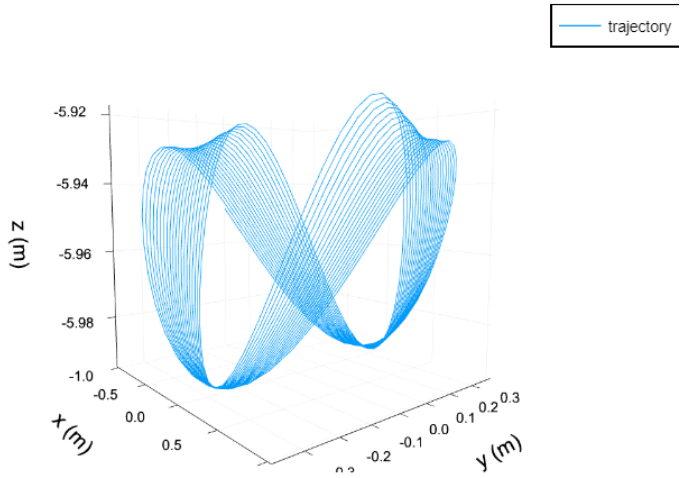


FIG. 9. Trajectory of spherical pendulum with constant length

- **Inextensible and massless string**
- **Point mass assumption for the capsule**
- **Planar motion assumption**
- **Constant length of the string**[2]
- **Taut string assumption**
- **Air resistance and other energy dissipative sources have not been considered**
- **Smooth motion of the capsule**

II.4. Relaxing the Assumptions

Now we are going to relax some of the above assumptions, and consider the other assumptions, and try to explain why we have done so.

- **Relaxed assumptions :** We have not assumed the capsule's motion to be confined in a single plane. We are considering 3-dimensional oscillations of the capsule, which means that it has a spherical pendulum-like motion. We have also not considered the length of the string to be constant the whole time. The mathematical models proposed for the first and third stages of the recovery process have taken the length of the string to be variable.

[2] this means that the length of the string is not a function of time

- **Considered assumptions :** We are considering the Dragon capsule to be a point object. It is not a good assumption considering the fact that it has dimensions comparable to the A-frame and that it is also rotating about the point , but this consideration would make the problem extremely complicated, whereas the assumption would lead to a much simpler problem by comparison. Another assumption we are considering is we are neglecting the effect of air resistance and other such forces that cause damping (torsional damping, friction between different parts, etc.). We have assumed the motion of the capsule to be smooth, which is a reasonable assumption as it can be seen from the video that the motion of the capsule is really smooth for the larger part of the recovery process. We are also considering the string to remain taut for the entirety of the process, which is also true since the gravitational force on the capsule ensures that the string always experiences tensile forces. Lastly, we are considering the string to be massless and inextensible. This is because the extension in the string would be negligible compared to its original length since it is a tough, high-strength string. As for the massless assumption, we can say that the expression containing the mass of the string would be accompanied by similar expressions containing the mass of the capsule in the numerator, and since the mass of the capsule is much higher than that of the string, we can neglect the terms containing the string's mass in the numerator.

II.5. Considered Mathematical Model

In this section, we are going to discuss the mathematical models that we have considered for the different stages of the recovery process. In all the three stages, we are going to consider the motion of the capsule to be like that of the bob of a spherical pendulum with respect to the point of suspension. Another thing to be noted is that the nature of the motion of the capsule(according to the considered mathematical model) would be similar for stages 1 and 3.

For finding out the governing equations of motion in each of the three stages, we have taken the help of Lagrangian Mechanics. The basic gist of the steps included in this method are:

- The first step is to define an inertial co-ordinate reference frame.
- The next step is to write the magnitude of the velocity vector of the Dragon capsule. We will first write the components of the position along the three reference axes, and then differentiate those to find the velocity component along the respective direction and use these components to obtain the velocity vector magnitude
- Once we obtain the magnitude of velocity, we can write the kinetic energy of the dragon capsule. We can obtain the gravitational potential energy using the z-coordinate of the of the capsule. For example, following are the values of the different parameters required in case of the model for stage 1.

$$
\begin{aligned}
x &= r\sin\theta\cos\phi \\
y &= r\sin\theta\sin\phi \\
z &= r\cos\theta \\
v^2 &= \dot{x}^2 + \dot{y}^2 + \dot{z}^2 \\
T &= \frac{1}{2}mv^2 \\
V &= mgz \\
\mathcal{L} &= T - V
\end{aligned}
$$

- The next step would include obtaining two second order ODEs with the help of the *Euler-Lagrange Equations.*

$$
\begin{aligned}
\frac{d}{dt}\left(\frac{\partial\mathcal{L}}{\partial\dot{\theta}}\right) - \frac{\partial\mathcal{L}}{\partial\theta} &= 0 \\
\frac{d}{dt}\left(\frac{\partial\mathcal{L}}{\partial\dot{\phi}}\right) - \frac{\partial\mathcal{L}}{\partial\phi} &= 0
\end{aligned}
$$

- The next step would be to convert these two second order ODEs into a system of four first order ODEs, so that we can use some ODE solver from a commercial package in order to obtain a numerical solution for the system of equations.

Let us now move on to the mathematical models considered for the different stages

II.5.1. Part-1: Lifting the module

1. **Mathematical Model -** Here we have considered a spherical pendulum with changing length. The change in length of the pendulum is considered to be a quasi-steady process and hence the rate of change of length is assumed to be a constant and a known quantity, which means the length of the string varies linearly with time and is known at any time instant t.
2. **Duration of this stage -** From the video, we found the overall duration of this stage to be around *8 seconds*
3. **Description -** The coordinate frame origin is located at the point of suspension of the pendulum. While writing the Euler-Lagrange equations, we are not going to consider an E-L equation for the length of the string r since r is a known function of time. Figure 7 shows how the axes and the different angles are being considered

4. **Deriving the Governing Equations:**

$$
\begin{aligned}
v^2 &= \dot{r}^2 + (r\,\dot{\theta})^2 + (r\,\dot{\phi}\,sin(\theta))^2 \\
T &= \frac{1}{2}mv^2 = \frac{1}{2}m(\,\dot{r}^2 + (r\,\dot{\theta})^2 + (r\,sin(\theta)\,\dot{\phi})^2) \\
V &= m\,g\,z = m\,g\,r\,cos(\theta) \\
\mathcal{L} &= T - V \\
&= \frac{1}{2}m(\,\dot{r}^2 + (r\,\dot{\theta})^2 + (r\,sin(\theta)\,\dot{\phi})^2) \\
&\quad - m\,g\,r\,cos(\theta)
\end{aligned}
$$

Now, writing the E-L equations for the Lagrangian obtained above, and simplifying the equations, we obtain the following second order non-linear ODEs

$$
\begin{aligned}
\ddot{\theta} &= \dot{\phi}^2 \sin\theta\cos\theta + \frac{g}{r}\sin\theta - 2\frac{\dot{\theta}\dot{r}}{r} \\
\ddot{\phi} &= -2\dot{\theta}\dot{\phi}\cot\theta - 2\frac{\dot{\phi}\dot{r}}{r}
\end{aligned}
$$

Now, in the above equations, we will replace $\dot{\theta}$ by ω and $\dot{\phi}$ by ψ, and obtain the following system of first order ODEs.

$$
\begin{aligned}
\dot{\theta} &= \omega \\
\dot{\omega} &= \psi^2 \sin\theta\cos\theta + \frac{g}{r}\sin\theta - 2\frac{\omega\dot{r}}{r} \\
\dot{\phi} &= \psi \\
\dot{\psi} &= -2\omega\psi\cot\theta - 2\frac{\psi\dot{r}}{r}
\end{aligned}
$$

5. **Initial Conditions:**

$$[\theta_o,\, \omega_o,\, \phi_o,\, \psi_o] = [17\pi/18,\, 0.1,\, 0.1,\, 0.1]$$

6. **Other Parameters:**

$$r \;=\; 6 + vt$$

where $v \;=\; -0.25$ m/s

All the angles are mentioned in radians and all the angular velocities have been mentioned in radians/seconds. We have solved the above system of equations on Python and Julia. The code for the program has been provided in the Appendix of the report. The different plots have been provided in section III of this report.

II.5.2. Part-2: Rotating the A-frame

1. **Mathematical Model -** Here we have considered a spherical pendulum with constant length but moving point of suspension. The change in angle made by the frame and the horizontal surface of the ship is considered to be a quasi-steady process and hence the rate of change of length is assumed to be a constant and a known quantity, which means the angle between the ship and the A-frame varies linearly with time and is known at any time instant t.

2. **Duration of this stage -** From the video, we found the overall duration of this stage to be around *45 seconds*

3. **Description -** The coordinate frame origin is located at the centre of the base of the A-frame. We cannot consider the origin to be situated at the point of suspension because in this case, any reference frame with origin as the point of suspension would be a non-inertial frame of reference. While writing the Euler-Lagrange equations, we are not going to consider an E-L equation for the angle between the ship's surface and the A-frame even though it is varying since it is a known function of time. Figure 10 clearly shows how the different axes and the different angles have been considered in this case.

4. **Deriving the Governing Equations:**

$$
\begin{aligned}
x &= r\sin\theta\cos\phi - L\cos\zeta \\
y &= r\sin\theta\sin\phi \\
z &= r\cos\theta + L\sin\zeta \\
v^2 &= (L\dot{\zeta})^2 + (r\,\dot{\theta})^2 + (r\,\dot{\phi}\,sin(\theta))^2 \\
&\quad + 2Lr\dot{\zeta}\left(\dot{\theta}\sin\zeta\cos\theta\cos\phi - \dot{\phi}\sin\zeta\sin\theta\sin\phi - \dot{\theta}\cos\zeta\sin\theta\right) \\
T &= \frac{1}{2}mv^2 \\
V &= m\,g\,z = m\,g\,(r\cos\theta + L\sin\zeta) \\
\mathcal{L} &= T - V \\
&= \frac{1}{2}mv^2 - m\,g\,(r\cos\theta + L\sin\zeta)
\end{aligned}
$$

Now, writing the E-L equations for the Lagrangian obtained above, and simplifying the equations, we obtain the following second order non-linear ODEs

$$
\begin{aligned}
\ddot{\theta} &= \dot{\phi}^2\sin\theta\cos\theta + \frac{g}{r}\sin\theta \\
&\quad - \frac{L\dot{\zeta}^2}{r}(\cos\zeta\cos\theta\cos\phi + \sin\zeta\sin\theta) \\
\ddot{\phi} &= -2\dot{\theta}\dot{\phi}\cot\theta + \frac{L\dot{\zeta}^2}{r}\cos\zeta\sin\theta\sin\phi
\end{aligned}
$$

Now, in the above equations, we will replace $\dot{\theta}$ by ω and $\dot{\phi}$ by ψ, and obtain the following system of

first order ODEs.

$$\dot{\theta} = \omega$$
$$\dot{\omega} = \psi^2 \sin\theta\cos\theta + \frac{g}{r}\sin\theta - \frac{L\beta^2}{r}(\cos\zeta\cos\theta\cos\phi + \sin\zeta\sin\theta)$$
$$\dot{\phi} = \psi$$
$$\dot{\psi} = -2\omega\psi\cot\theta + \frac{L\beta^2}{r}\cos\zeta\sin\theta\sin\phi$$

5. **Initial Conditions:**

$$[\theta_o,\ \omega_o,\ \phi_o,\ \psi_o] = [17\pi/18,\ 0.1,\ 0.1,\ 0.1]$$

6. **Other Parameters:**

$$\zeta = \frac{\pi}{6} + \beta t$$

where $\beta = +0.078$ radians/seconds

All the angles are mentioned in radians and all the angular velocities have been mentioned in radians/seconds. We have solved the above system of equations on Julia. The code for the program has been provided in the Appendix of the report. The different plots have been provided in section III of this report.

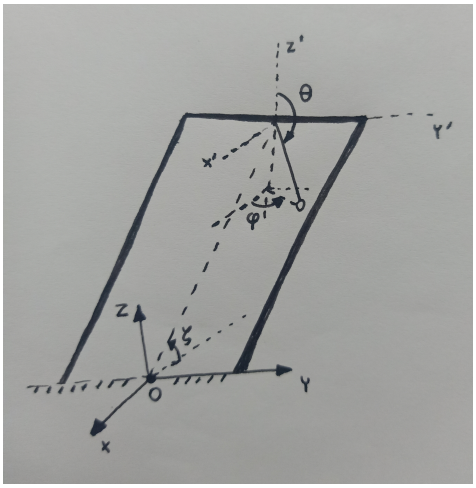


FIG. 10. Reference co-ordinates and angles considered for stage 2 of the recovery process

II.5.3. Part-3: Placing the module in the Dragon's nest

1. **Mathematical Model -** Here we have considered a spherical pendulum with changing length. The change in length of the pendulum is considered to be a quasi-steady process and hence the rate of change of length is assumed to be a constant and a known quantity, which means the length of the string varies linearly with time and is known at any time instant t.

2. **Duration of this stage -** From the video, we found the overall duration of this stage to be around *4 seconds*

3. **Description -** The coordinate frame origin is located at the point of suspension of the pendulum. While writing the Euler-Lagrange equations, we are not going to consider an E-L equation for the length of the string r, since r is a known function of time. Figure 7 shows how the axes and the different angles are being considered

4. **Deriving the Governing Equations:**

$$v^2 = \dot{r}^2 + (r\,\dot{\theta})^2 + (r\,\dot{\phi}\,sin(\theta))^2$$
$$T = \frac{1}{2}mv^2 = \frac{1}{2}m(\,\dot{r}^2 + (r\,\dot{\theta})^2 + (r\,sin(\theta)\,\dot{\phi})^2)$$
$$V = m\,g\,z = m\,g\,r\,cos(\theta)$$
$$\mathcal{L} = T - V$$
$$= \frac{1}{2}m(\,\dot{r}^2 + (r\,\dot{\theta})^2 + (r\,sin(\theta)\,\dot{\phi})^2) - m\,g\,r\,cos(\theta)$$

Now, writing the E-L equations for the Lagrangian obtained above, and simplifying the equations, we obtain the following second order non-linear ODEs

$$\ddot{\theta} = \dot{\phi}^2\sin\theta\cos\theta + \frac{g}{r}\sin\theta - 2\frac{\dot{\theta}\dot{r}}{r}$$
$$\ddot{\phi} = -2\dot{\theta}\dot{\phi}\cot\theta - 2\frac{\dot{\phi}\dot{r}}{r}$$

Now, in the above equations, we will replace $\dot{\theta}$ by ω and $\dot{\phi}$ by ψ, and obtain the following system of first order ODEs.

$$\dot{\theta} = \omega$$
$$\dot{\omega} = \psi^2\sin\theta\cos\theta + \frac{g}{r}\sin\theta - 2\frac{\omega\dot{r}}{r}$$
$$\dot{\phi} = \psi$$
$$\dot{\psi} = -2\omega\psi\cot\theta - 2\frac{\psi\dot{r}}{r}$$

5. **Initial Conditions:**

$$[\theta_o,\ \omega_o,\ \phi_o,\ \psi_o] = [17\pi/18,\ 0.1,\ 0.1,\ 0.1]$$

6. **Other Parameters:**

$$r = 4 + vt$$

where $v = +0.25$ m/s

All the angles are mentioned in radians and all the angular velocities have been mentioned in radians/seconds. We have solved the above system of equations on Python and Julia. The code for the program has been provided in the Appendix of the report. The different plots have been provided in section III of this report.

II.6. Relative and Absolute Tolerances

In this section, we are going to explain what relative and absolute tolerances are, how they affect the solution of the system of differential equations and what happens to the solutions when these tolerances are changed.

Every numerical algorithm will have errors and it is important to quantify them to understand the convergence of the solution and to analyze the performance of the algorithm on the given differential equation and to decide an appropriate algorithm. Converge tests are performed on the solution at every step and the current state is compared with the previous state to arrive at the error. There are basically two types of errors- absolute error (atol) and relative error (rtol)- which are self explanatory from their names.

Excerpt from the documentation of the ODE solver odeint() from scipy.integrate: [3]
" *The input parameters 'rtol' and 'atol' determine the error control performed by the solver. The solver will control the vector, e, of estimated local errors in y, according to an inequality of the form "max-norm of* $(e/ewt) <= 1$*", where ewt is a vector of positive error weights computed as "ewt = rtol * abs(y) + atol". rtol and atol can be either vectors the same length as y or scalars. Defaults to 1.49012e-8.* "

e is the vector of estimated local errors while ewt is a vector of positive error weights.

The inequality:

$$\left\| \frac{e}{ewt} \right\| \leq 1$$

where ewt is defined as:

$$ewt = rtol * \mid y \mid + atol$$

Since relative tolerance gives the relative change in the solution since the last iteration, it is really useful for checking convergence in general as the relative change dictates where the solution is going to stagnate at one consistent value and the change is insignificant compared to the solution itself. But using relative tolerance has a roadblock when the solution lies at zero, because now the difference divided by a solution that tends to zero has a large value and is undefined on zero and the solution will never converge. Here using absolute tolerance becomes useful as it doesn't have the singularity at zero. Since most solvers need either of the criteria to be satisfied to stop iterating, most of the times rtol converges into its limit earlier than atol except when solution is at zero wherein atol converges before rtol.

III. DATA PRESENTATION

The plots for the numerically obtained parameters, the trajectories and the total energies for each of the three stages have been presented in this section. We have also varied the relative and absolute tolerance parameters for the ODE solvers used and compared the results corresponding to these variations in the form of plots. The plots begin from the next page of this report.

III.1 Plots corresponding to the first stage of recovery:

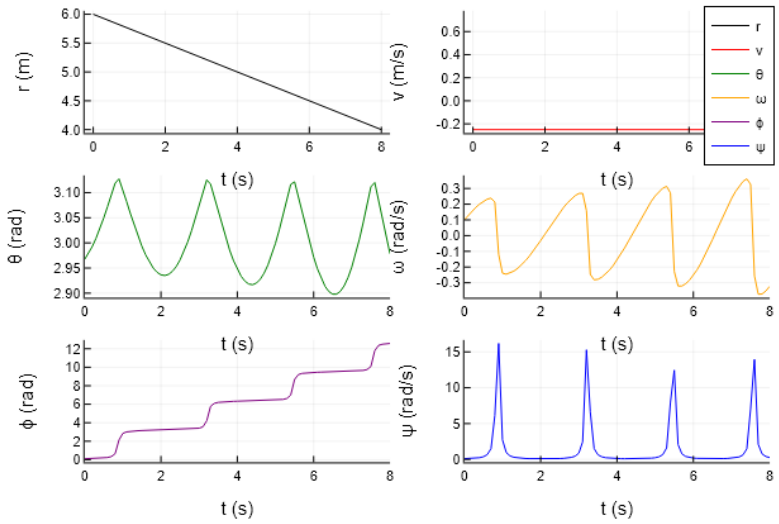


Figure 11. Plots for all the parameters which have been numerically computed, corresponding to stage 1

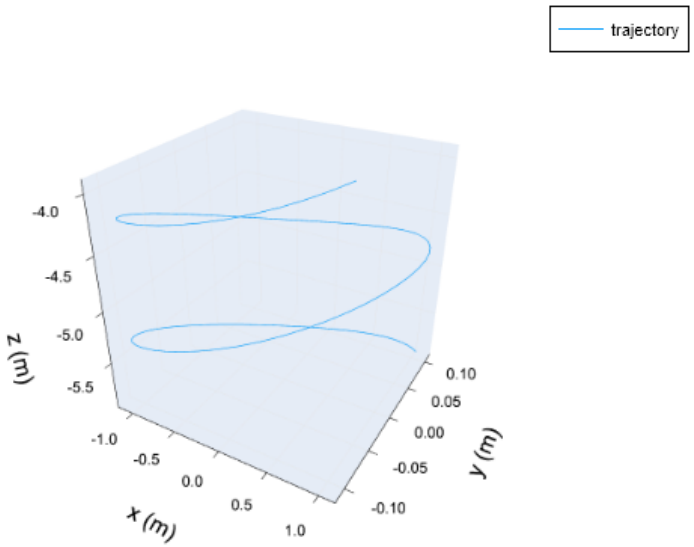


Figure 12. Trajectory of the capsule during stage 1 of the recovery

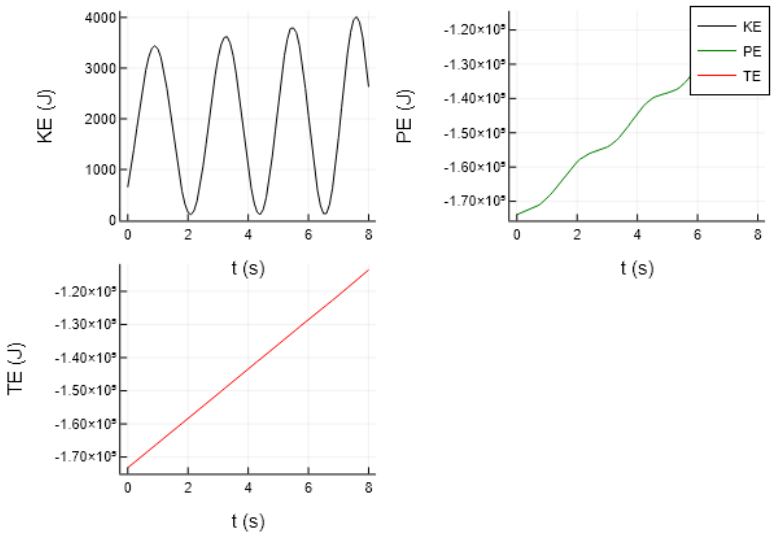


Figure 13. Variation of energy as the capsule moves during stage 1 of recovery

III.2 Plots corresponding to the second stage of recovery:

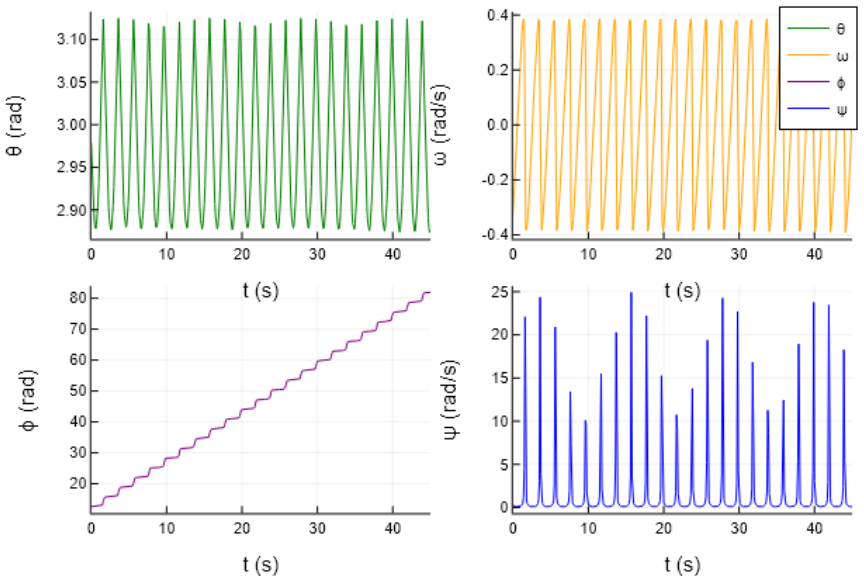


Figure 14. Plots for all the parameters which have been numerically computed, corresponding to stage 2

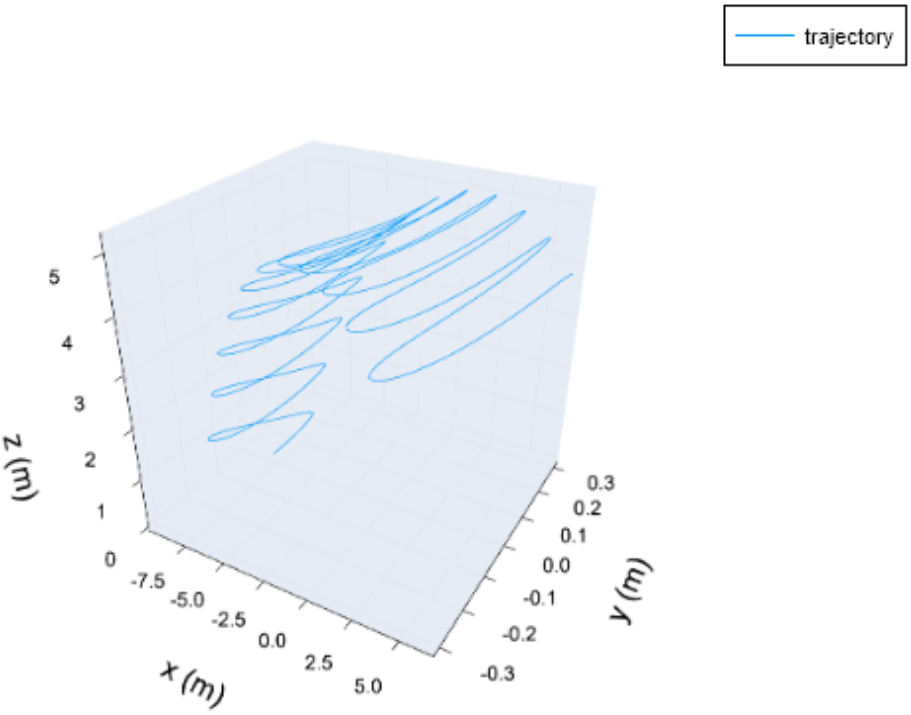


Figure 15. Trajectory of the capsule during stage 2 of the recovery

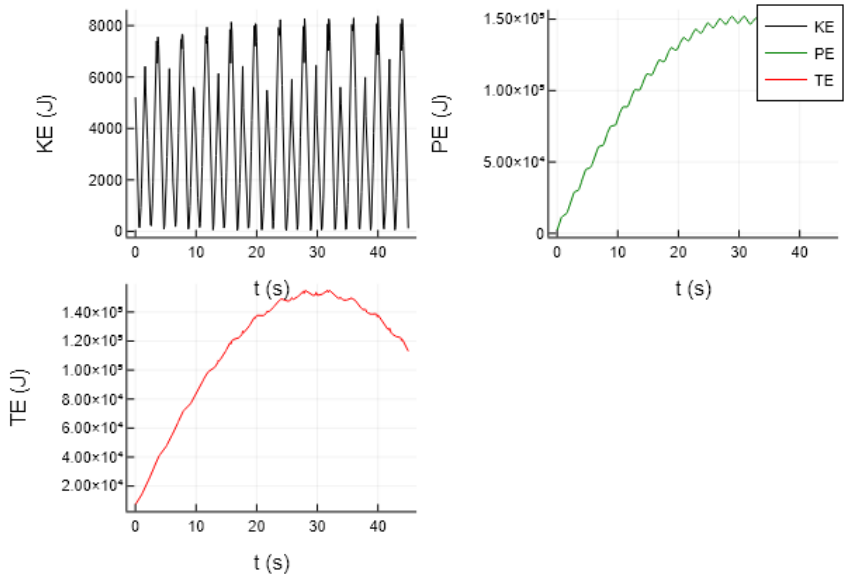


Figure 16. Variation of energy as the capsule moves during stage 2 of recovery

III.3 Plots corresponding to the third stage of recovery:

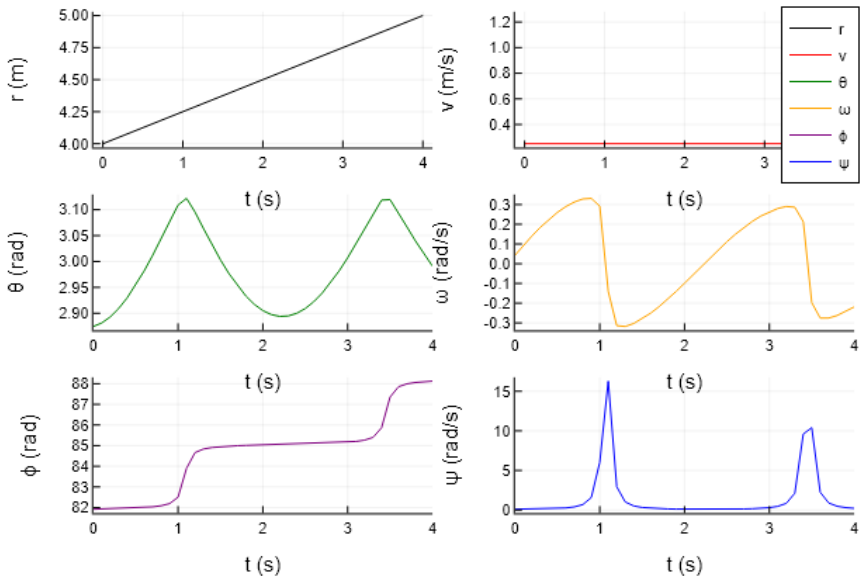


Figure 17. Plots for all the parameters which have been numerically computed, corresponding to stage 3

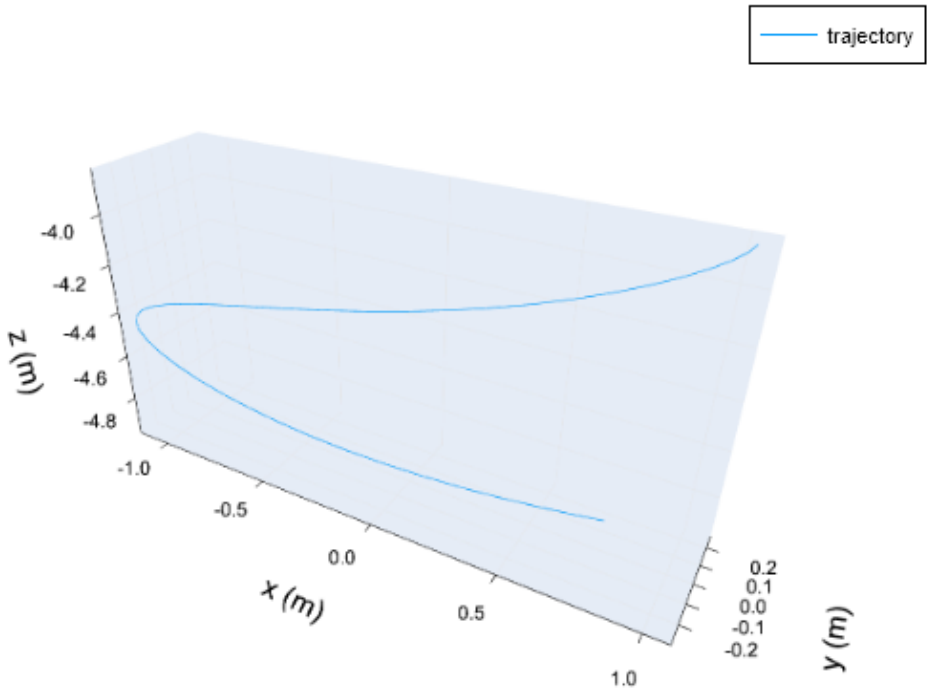
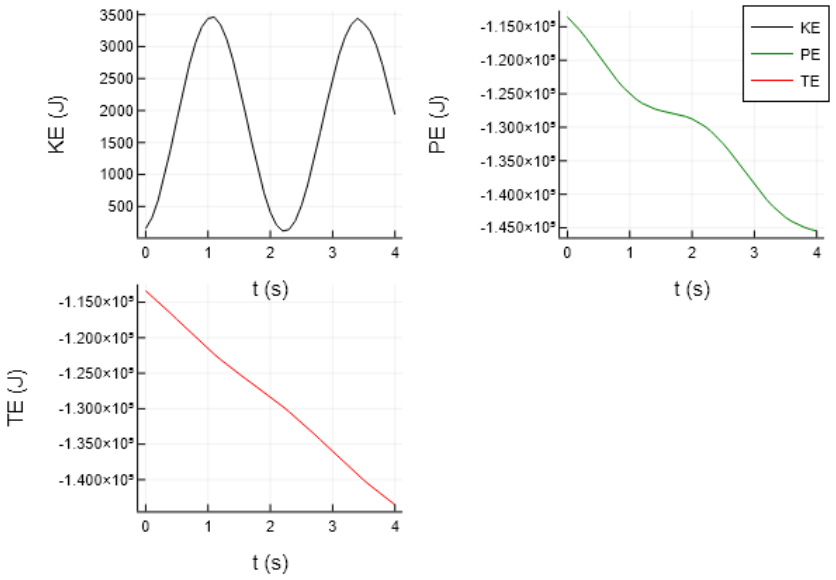


Figure 18. Trajectory of the capsule during stage 3 of the recovery



8

Figure 19. Variation of energy as the capsule moves during stage 3 of recovery

III. 4 Plots for the two angles θ and ϕ while varying the relative and absolute tolerances corresponding to stage 1

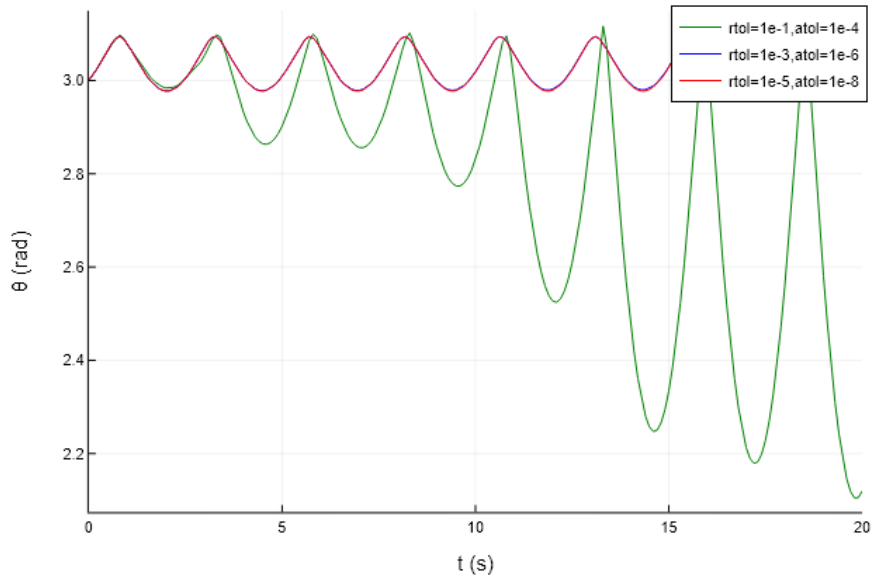


Figure 20. Plot depicting how the numerical solution for θ varies as we vary the rtol and atol parameters

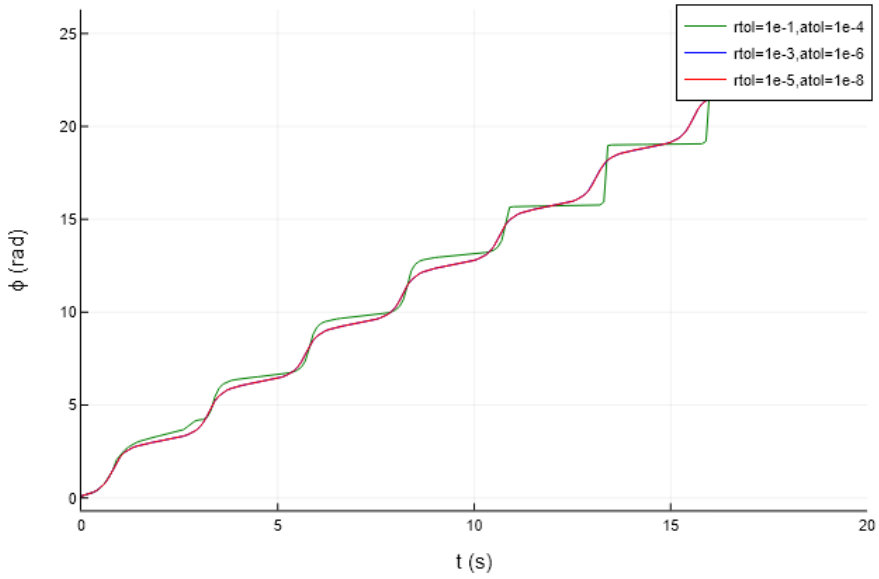


Figure 21. Plot depicting how the numerical solution for ϕ varies as we vary the rtol and atol parameters

III.5 Plots for checking the convergence of the numerically obtained solutions

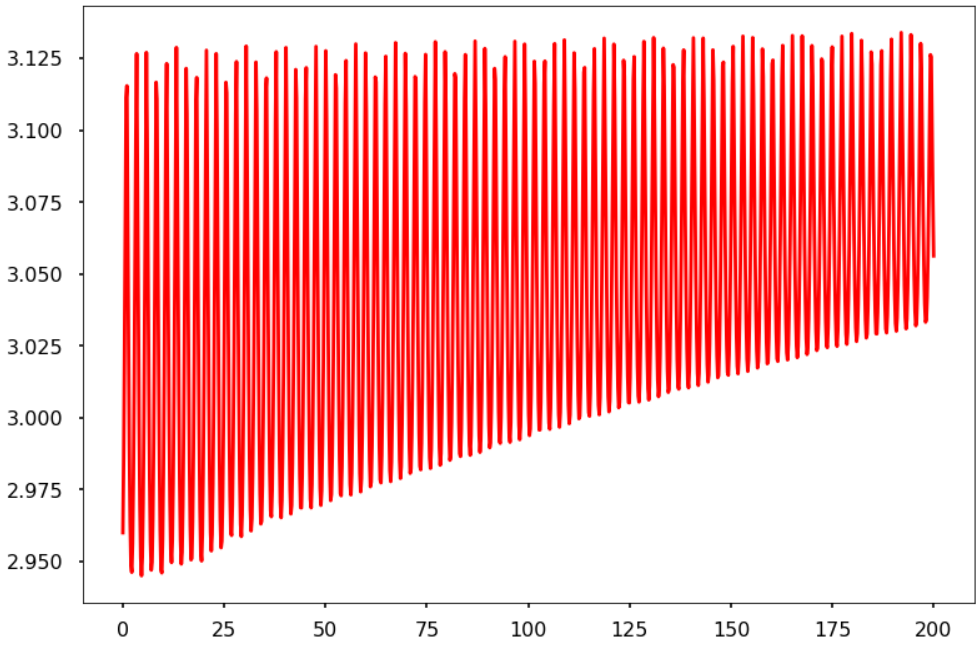


Figure 22. Plot depicting how the numerical solution for θ varies with time - stage 1

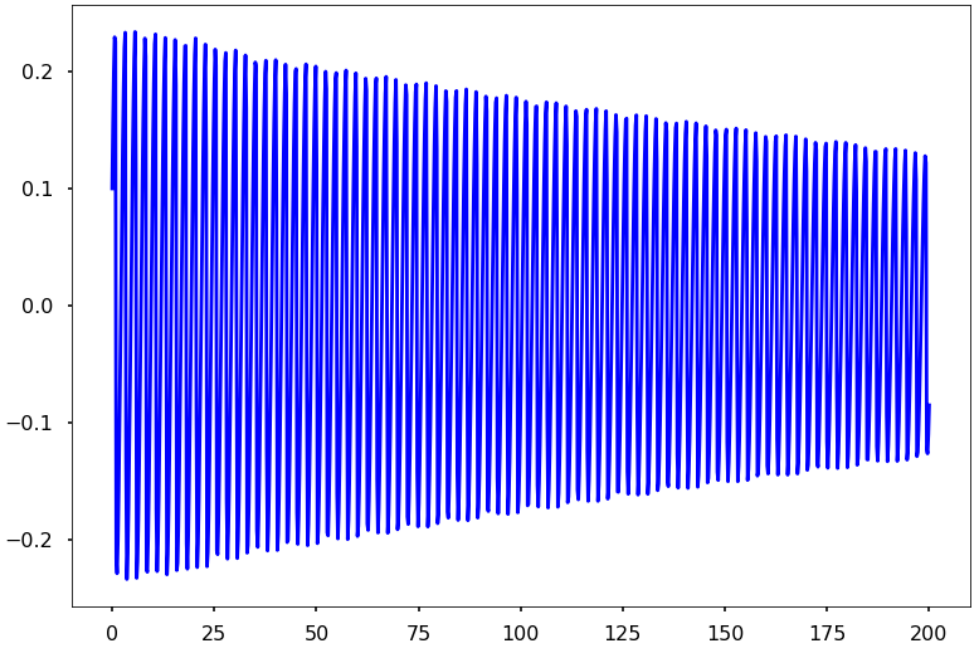


Figure 23. Plot depicting how the numerical solution for $\dot{\theta}$ varies with time - stage 1

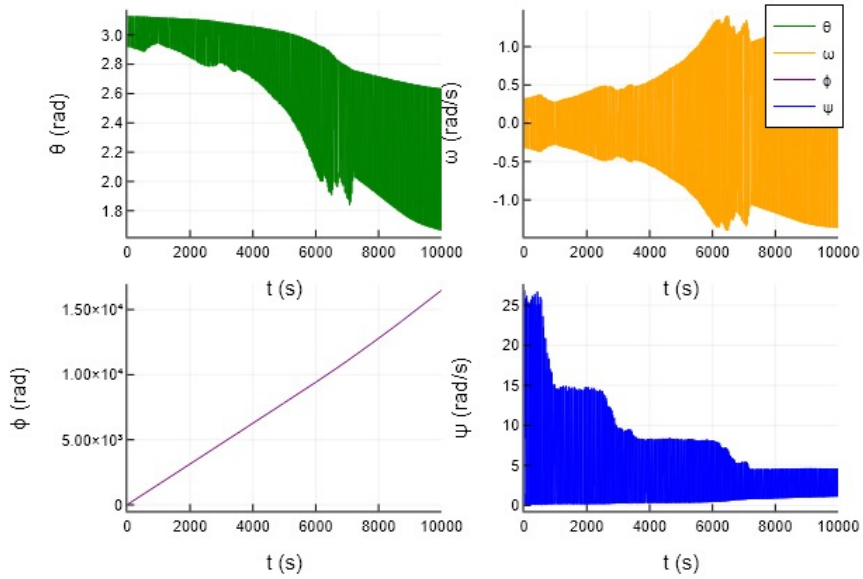


Figure 24. Plot depicting how the numerical solution for the angles and their derivatives vary with time - stage 2

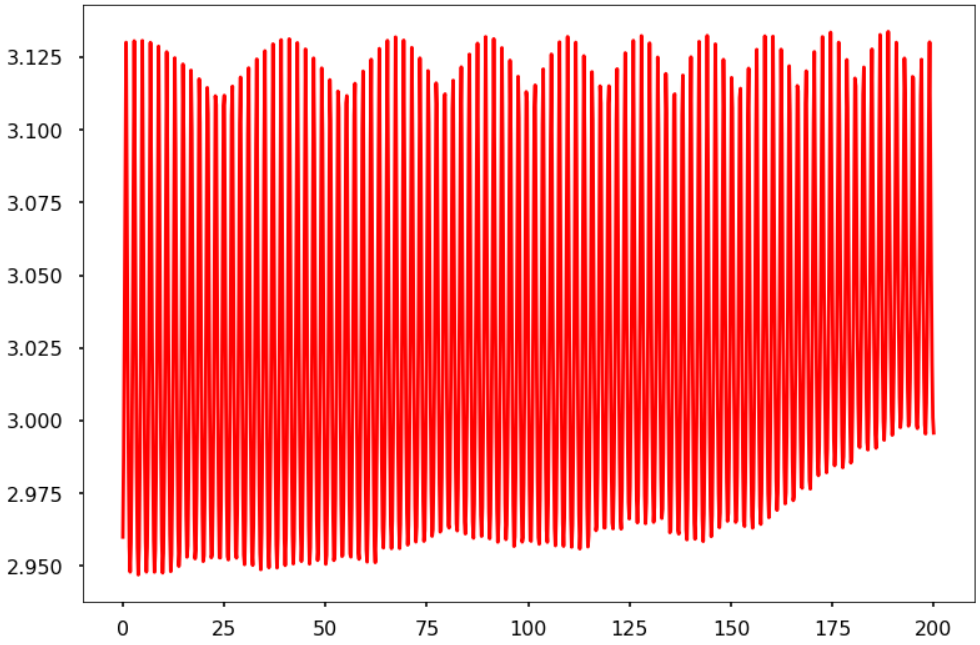


Figure 25. Plot depicting how the numerical solution for θ varies with time - stage 3

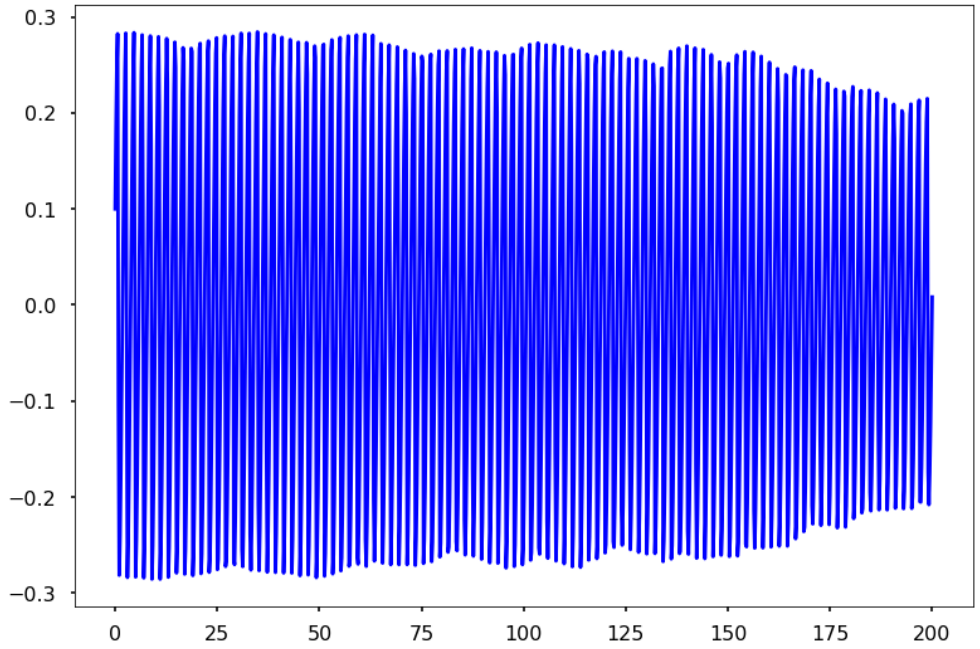


Figure 26. Plot depicting how the numerical solution for $\dot{\theta}$ varies with time - stage 3

IV. RESULTS AND DISCUSSIONS

In the last section, we presented the plots corresponding to each of the three stages of the recovery of the Dragon capsule. In this section, we are going to summarize the obtained results, and make inferences based on the plots of the previous section.

IV.1. observations

IV.1.1. Part 1: Lifting the module

1. **State variable plot:**
   (a) Since we assumed a quasi-steady process, we assumed radial velocity to be a small constant towards the centre($\dot{r} << 1$).
   (b) Similar to a spherical pendulum with constant length, θ and ω are oscillating. The difference is that there is a gradual increase in the peak to valley amplitude. This is because in order for the angular momentum to remain constant when the length is decreasing, we must have an increase in the angular speed amplitude.
   (c) Since the revolutions are completed in the horizontal direction, ϕ keeps on increasing.
2. **Trajectory:**
   (a) We observe a spiral motion of the capsule as the time increases. There is a net decrease in the magnitude of the z-coordinate due to reduction of length of the pendulum. But the decrease won't be monotonous because of the oscillations of the capsule (which increase the magnitude of $\cos\theta$ when the bob moves toward the mean position).
3. **Energy Plots:**
   (a) The kinetic energy of the capsule keeps on oscillating and is always positive (as it should, in case of a pendulum)
   (b) The magnitude of kinetic energy is much smaller compared to the magnitude of potential energy, primarily because the speed of the capsule should be small(desirably, to keep the oscillations to a minimum). Due to this, the total energy of the capsule mainly depends on the potential energy.
   (c) The potential energy is always negative since we have chosen a coordinate frame such that the z-coordinate of the capsule is always negative. The total energy is also negative, but it increases(i.e. the magnitude decreases, since total energy is negative) with time as the potential energy increases due to decrease in the z-coordinate's magnitude as the length of pendulum decreases.

IV.1.2. Part 2: Rotating the A-frame

1. **State variable plot:**
   (a) Here we assume the length of the pendulum to be constant and hence the radial velocity of the plot is zero and the state is mainly determined by the variations in θ and ϕ.
2. **Trajectory:**
   (a) The trajectory is a lean bent helix and as the time increases, we moves from the bottom of the helix to its tip. The thickness of the helix is increasing with time as amplitude of θ oscillations increases.
3. **Energy Plots:**
   (a) Kinetic energy oscillates with respect to time with increasing amplitude.
   (b) Potential energy is positive as the origin of the coordinate frame is below the moving point and hence z is positive. Potential energy increases and reaches a maximum and then decreases, following the similar trend in z as seen in the trajectory. This is because the the A-frame is assumed to rise to an obtuse angle from the horizontal surface.
   (c) Total energy, being strongly dependent on potential energy than kinetic energy, follows a similar trend as the potential energy.

IV.1.3. Part 3: Placing the module in the dragon's nest

1. **State variable plot:**
   (a) Since we assumed a quasi-steady process, we assumed radial velocity to be a small constant away the centre($\dot{r} << 1$).
   (b) θ and ω are oscillating with decreasing peak to valley amplitude, which is opposite to the change in the first stage. This is because the length of the pendulum increases and in order for the angular momentum to remain conserved, the angular speed must gradually decrease(the amplitude decreases).
   (c) Since the revolutions are completed in the horizontal direction, ϕ keeps on increasing similar to part-1.
2. **Trajectory:**

(a) We observe a downward spiral motion of the capsule as the time increases opposite to that of part-1. There is a net increase in the magnitude of the z-coordinate due to increase of length of the pendulum. But the increase won't be monotonous because of the oscillations of the capsule (which decrease the magnitude of $\cos\theta$ when the bob moves away from the mean position).

3. **Energy Plots:**

(a) Similar to part-1, the kinetic energy of the capsule is oscillating and is always positive.

(b) The kinetic energy is of lower orders of magnitude compared to the potential energy and contributes very less to total energy.

(c) The potential energy is always negative since we have chosen a coordinate frame such that the z-coordinate of the capsule is always negative. The total energy is also negative, but it decreases (i.e. the magnitude increases, since total energy is negative) with time as the potential energy decreases due to increase in the z-coordinate's magnitude as the length of pendulum increases.

IV.2. Effect of Tolerances

Relative tolerances and absolute tolerances are changed by order of 2 on both sides of the default values (10^{-3} for relative tolerances, 10^{-6} for absolute tolerances). We can see that the loose tolerances cause the iterations to stop prematurely and conclude at a wrong solution for θ and ϕ. When the tolerance was increased further after the first increase, it was seen that the change in the solutions was negligible, but the computation time was higher. We prefer to choose such values for tolerances which optimize the performance of the function - providing reasonably accurate solutions without sacrificing too much time for computation (as in the second case).

V. CONCLUSION

In conclusion, it can be said that the considered mathematical model is a good approximation for the considered physical problem when we want to get the approximate position of the point of the Dragon capsule to which the string is attached. This model does not provide knowledge of the motion of all the points on the capsule, and this limitation can be overcome by considering the rotation of the capsule about the point at which it is attached to the string, and not considering it as a point object. According to the model considered, it can be seen that the angle of oscillation θ and the angular speed ω decrease (convergent nature) in phase 3 as time increases, which is highly desirable since we don't want the capsule to oscillate too much while it is being placed. In order to prevent the angle and angular speed from increasing uncontrollably, we see that there are two more strings attaching the base of the A-frame to the capsule such that the strings get taut in case the capsule undergoes large oscillations.

[1] "https://diffeq.sciml.ai/release-2.0/solvers/ode_solve.html," .
[2] "https://diffeq.sciml.ai/stable/tutorials/ode_example/example-2 : -solving - systems - of - equations," .
[3] Scipy, "scipy/odepack.py at v1.7.1 · scipy/scipy," .

Appendix : Script files for the numerical solvers

Script for simple pendulum

```
using DifferentialEquations
using Plots
plotlyjs()

#constants
l = 1.0                              # length [m]
m = 1.0                              # mass[Kg]
g = 9.81                             # gravitational acceleration [m/s^2]

#function to compute ODE
function pendulum!(du,u,p,t)
    du[1] = u[2]                     # θ'(t) = ω(t)
    du[2] = -(g\l)*sin(u[1])   # ω'(t) = -(g/l) sin θ(t)
end

#initial conditions
θ0 = π/6                          # initial angular deflection [rad]
ω0 = 0.0                             # initial angular velocity [rad/s]
u0 = [θ0, ω0]                        # initial state vector
tspan = (0.0,100.0)                   # time interval

#defining the problem and solving
prob = ODEProblem(pendulum!,u0,tspan)
sol = solve(prob,saveat=0.1)

#plotting the solution
display(plot(sol,linewidth=2,xaxis="t (s)",
    color=["green" "blue"],label=["θ [rad]" "ω [rad/s]"],layout=(2,1)))
savefig("./plots/simple pendulum-plot.png")

# trajectory
t=sol.t
U=sol[1:end,:]
(θ,ω)=[U[x,:] for x in 1:size(U,1)]
x=l.*sin.(π.-θ)
y=zeros(size(t))
z=l.*cos.(π.-θ)
display(plot(x,y,z,linewidth=2,label="trajectory",
    xaxis="x (m)",yaxis="y (m)",zaxis="z (m)",grid=(:on,:black)))
savefig("./plots/simple pendulum-trajectory.png")
```

Script for simple pendulum with damping

```
using DifferentialEquations
using Plots
plotlyjs()

# constants
l = 1.0                              # length [m]
m = 1.0                              # mass[Kg]
g = 9.81                             # gravitational acceleration [m/s^2]
b = 0.05                            # damping coefficient [s^-1]

#function to compute the ODEs
function pendulum!(du,u,p,t)
    du[1] = u[2]                     # θ'(t) = ω(t)
    du[2] = -(b*u[2])-(g\l)*sin(u[1]) # ω'(t) = -(g/l) sin θ(t) + 3/(ml^2)M(t)
end

#initial conditions
θ0 = π/6                           # initial angular deflection [rad]
ω0 = 0                              # initial angular velocity [rad/s]
u0 = [θ0, ω0]                        # initial state vector
tspan = (0.0,200.0)                   # time interval

#defining the problem and solving
prob = ODEProblem(pendulum!,u0,tspan)
sol = solve(prob,saveat=0.1)

#plotting the solution
display(plot(sol,linewidth=2,
    xaxis="t (s)",color=["green" "blue"],
    label=["θ [rad]" "ω [rad/s]"],layout=(2,1)))
savefig("./plots/simple damped pendulum-plot.png")

#trajectory
t=sol.t
U=sol[1:end,:]
(θ,ω)=[U[x,:] for x in 1:size(U,1)]
x=l.*sin.(π.-θ)
y=zeros(size(t))
z=l.*cos.(π.-θ)
display(plot(x,y,z,
    linewidth=2,label="trajectory",xaxis="x (m)",
    yaxis="y (m)",zaxis="z (m)",grid=(:on,:black)))
savefig("./plots/simple damped pendulum-trajectory.png")
```

Script for simple pendulum with changing length

```
using DifferentialEquations
using Plots
plotlyjs()

#constants
m = 1.0                                 # mass[Kg]
g = 9.81                                # gravitational acceleration [m/s^2]

#function to compute the ODEs
function pendulum!(du,u,p,t)
    r,v,θ,ω=u
    du[1]=dr=v
    du[2]=dv=r*ω^2+g*cos(θ)
    du[3]=dθ=ω
    du[4] =dω=-(g\r)*sin(θ) - (2*v*ω)/r
end

#initial conditions
r0=4                                  #initial radial distance
v0=-0.1                                #initial radial velocity
θ0 = 1π/18                              # initial angular deflection [rad]
ω0 = 1                               # initial angular velocity [rad/s]
u0 = [r0,v0,θ0, ω0]                          # initial state vector
tspan = (0.0,5.0)                    # time interval

#defining the problem and solving
prob = ODEProblem(pendulum!,u0,tspan)
sol = solve(prob,saveat=0.1)

#plotting
p1=plot(sol,vars=(0,1),label="r",
    color="purple", xaxis="t (s)", yaxis="r (m)")
p2=plot(sol,vars=(0,2),label="v",
    color="blue", xaxis="t (s)", yaxis="v (m/s)")
p3=plot(sol,vars=(0,3),label="θ",
    color="green", xaxis="t (s)", yaxis="θ (rad)")
p4=plot(sol,vars=(0,4),label="ω",
    color="orange", xaxis="t (s)", yaxis="ω (rad/s)")
display(plot(p1,p2,p3,p4,layout=(2,2)))
savefig("./plots/simple length changing pendulum-plot.png")

#trajectory
t=sol.t
U=sol[1:end,:]
(r,v,θ,ω)=[U[x,:] for x in 1:size(U,1)]
x=r.*sin.(θ)
y=zeros(size(t))
z=r.*cos.(θ)
display(plot(x,y,z,linewidth=2,label="trajectory",
    xaxis="x (m)",yaxis="y (m)",zaxis="z (m)",grid=(:on,:black)))
savefig("./plots/simple length changing pendulum-trajectory.png")
```

Script for spherical pendulum

```
using DifferentialEquations
using Plots
plotlyjs()

#constants
g=9.81        #acceleration due to gravity [ms^-2]
m=3000        #mass of capsule [Kg]

#initial conditions
u₀=[3,0.1,0.1,0.5]
r₀=6
v₀=0

#function to compute lagrangian
function lagrangian!(du,u,p,t)
    θ,ω,φ,ψ=u
    v=v₀
    r=r₀+v*t
    du[1]=dθ=ω
    du[2]=dω=sin(θ)*cos(θ)*ψ^2-((2*v*ω)/r)+((g*sin(θ))/r)
    du[3]=dφ=ψ
    du[4]=dψ=-((2*v*ψ)/r)-(2*cot(θ)*ω*ψ)
end

#time interval
tspan=(0.0,100.0)

#defining the problem and solving
prob=ODEProblem(lagrangian!,u₀,tspan)
sol=solve(prob,saveat=0.1)

#extracting solution
U=sol[1:end,:]
(θ,ω,φ,ψ)=[U[x,:] for x in 1:size(U,1)]
t=sol.t
r=r₀ .+ v₀*t
v=v₀*ones(size(t))
x=r.*sin.(θ).*cos.(φ)
y=r.*sin.(θ).*sin.(φ)
z=r.*cos.(θ)
uₜ=U[:,end]
rₜ=r[end]
vₜ=v[end]
FC=[rₜ,vₜ,uₜ]

#plotting the solution
display(plot(x,y,z,label="trajectory",xaxis="x (m)",
    yaxis="y (m)",zaxis="z (m)",grid=(:on,:black)))
savefig("./plots/trajectory-sp.png")

p1=plot(sol.t,r,label="r",color="black",
    xaxis="t (s)", yaxis="r (m)")
p2=plot(sol.t,v,label="v",color="red",
    xaxis="t (s)", yaxis="v (m/s)")
p3=plot(sol,vars=(0,1),label="θ",
    color="green", xaxis="t (s)", yaxis="θ (rad)")
p4=plot(sol,vars=(0,2),label="ω",
    color="orange", xaxis="t (s)", yaxis="ω (rad/s)")
p5=plot(sol,vars=(0,3),label="φ",
    color="purple", xaxis="t (s)", yaxis="φ (rad)")
p6=plot(sol,vars=(0,4),label="ψ",
    color="blue", xaxis="t (s)", yaxis="ψ (rad/s)")
display(plot(p1,p2,p3,p4,p5,p6,layout=(3,2)))
savefig("./plots/plots-sp.png")

#energy plot
KE=0.5*m*(v.^2 .+ (r.*ω).^2 .+ (r.*ψ.*sin.(θ)).^2)
PE=m*g.*z
TE=KE.+PE
p7=plot(t,KE,label="KE",
    color="black",xaxis="t (s)", yaxis="KE (J)")
p8=plot(t,PE,label="PE",
    color="green",xaxis="t (s)", yaxis="PE (J)")
p9=plot(t,TE,label="TE",
    color="red",xaxis="t (s)", yaxis="TE (J)")
display(plot!(p7,p8,p9))
savefig("./plots/energyplot-sp.png")
```

Script for spherical pendulum with changing length (Julia) - Stage 1

```
using DifferentialEquations
using Plots
plotlyjs()

function ex2_case_1(IC)

#constants
g=9.81        #acceleration due to gravity [ms^-2]
m=3000        #mass of capsule [Kg]

#initial conditions
r_0,v_0,u_0=IC
# u_0=[3,0.1,0.1,0.1]
# r_0=6
# v_0=-0.25

#function to compute lagrangian
function lagrangian!(du,u,p,t)
    θ,ω,φ,ψ=u
    v=v_0
    r=r_0+v*t
    du[1]=dθ=ω
    du[2]=dω=sin(θ)*cos(θ)*ψ^2-((2*v*ω)/r)+((g*sin(θ))/r)
    du[3]=dφ=ψ
    du[4]=dψ=-((2*v*ψ)/r)-(2*cot(θ)*ω*ψ)
end

#time interval
tspan=(0.0,8.0)

#defining the problem and solving
prob=ODEProblem(lagrangian!,u_0,tspan)
sol=solve(prob,saveat=0.1)

#extracting solution
U=sol[1:end,:]
(θ,ω,φ,ψ)=[U[x,:] for x in 1:size(U,1)]
t=sol.t
r=r_0 .+ v_0*t
v=v_0*ones(size(t))
x=r.*sin.(θ).*cos.(φ)
y=r.*sin.(θ).*sin.(φ)
z=r.*cos.(θ)
u_t=U[:,end]
r_t=r[end]
v_t=v[end]

# # changing coordinate frames
x_p =x.-(5*(3^0.5))
y_p=y
z_p=z.+5
trajectory=hcat(x_p,y_p,z_p)

FC=(r_t,v_t,u_t,trajectory)

#plotting the solution
display(plot(x,y,z,label="trajectory",
    xaxis="x (m)",yaxis="y (m)",zaxis="z (m)",grid=(:on,:black)))
savefig("./plots/trajectory-1.png")

p1=plot(sol.t,r,label="r",
    color="black",xaxis="t (s)", yaxis="r (m)")
p2=plot(sol.t,v,label="v",
    color="red",xaxis="t (s)", yaxis="v (m/s)")
p3=plot(sol,vars=(0,1),label="θ",
    color="green", xaxis="t (s)", yaxis="θ (rad)")
p4=plot(sol,vars=(0,2),label="ω",
    color="orange", xaxis="t (s)", yaxis="ω (rad/s)")
p5=plot(sol,vars=(0,3),label="φ",
    color="purple", xaxis="t (s)", yaxis="φ (rad)")
p6=plot(sol,vars=(0,4),label="ψ",
    color="blue", xaxis="t (s)", yaxis="ψ (rad/s)")
display(plot(p1,p2,p3,p4,p5,p6,layout=(3,2)))
savefig("./plots/plots-1.png")

#energy plot
KE=0.5*m*(v.^2 .+ (r.*ω).^2 .+ (r.*ψ.*sin.(θ)).^2)
PE=m*g.*z
```

```
TE=KE.+PE
p7=plot(t,KE,label="KE",
    color="black",xaxis="t (s)", yaxis="KE (J)")
p8=plot(t,PE,label="PE",
    color="green",xaxis="t (s)", yaxis="PE (J)")
p9=plot(t,TE,label="TE",
    color="red",xaxis="t (s)", yaxis="TE (J)")
display(plot!(p7,p8,p9))
savefig("./plots/energyplot-1.png")

return FC

end
```

Script for spherical pendulum with changing length (Python) - Stage 1

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 26 13:07:51 2021

@author: AYUSH
"""
from scipy.integrate import solve_ivp
from numpy import sin,cos
import matplotlib.pyplot as pt
import numpy as np
from mpl_toolkits import mplot3d

tspan=[0,8]
teval=np.linspace(tspan[0], tspan[1], 1000)
g=9.81

def e_l_eqns(t,u):
    v=-0.25
    r=6+(v*t)
    theta,omega,phi,psi=u
    du=[omega,(sin(theta)*cos(theta)*(psi**2))-((2*v*omega)/r)+((g*sin(theta))/r
     ),psi,-((2*v*psi)/r)-(2*cos(theta)*omega*psi/sin(theta))]
    return du

res=solve_ivp(e_l_eqns, tspan, [2.96,0.1,0.1,0.1], t_eval=teval)
theta,omega,phi,psi=res.y
t=res.t
r1=6 - (0.25*t)
x=r1*sin(theta)*cos(phi)
y=r1*sin(theta)*sin(phi)
z=r1*cos(theta)
pt.plot(t,theta,color='red')
pt.show()
pt.plot(t,omega,color='blue')
pt.show()
pt.plot(t,phi)
pt.show()
pt.plot(t,psi)
pt.show()
pt.style.use('seaborn-poster')
g = pt.figure(figsize = (12,12))
ax = pt.axes(projection='3d')
ax.grid()
ax.plot3D(x, y, z)
ax.set_title('3D Parametric Plot')
ax.set_xlabel('x', labelpad=20)
ax.set_ylabel('y', labelpad=20)
ax.set_zlabel('z', labelpad=20)
pt.show()
```

Script for spherical pendulum with rotating suspension point - Stage 2

```
using DifferentialEquations
using Plots
plotlyjs()

function ex2_case_2(IC)

#constants
g=9.81          #acceleration due to gravity [ms^-2]
m=3000          #mass of capsule [Kg]
L=10            #height of A-frame [m]

#initial conditions
r0,v0,u0,traj=IC #traj-trajectory of capsule till now-not needed for calculation
ζ0=π/6 # angle subtended by the A-frame [rad]
β0=dζ=90*π/(180*45)

#function to compute lagrangian
function lagrangian!(du,u,p,t)
    θ,ω,φ,ψ=u
    r=r0
    β=dζ=β0
    ζ=ζ0+β*t
    du[1]=dθ=ω
    du[2]=dω=(ψ^2)*sin(θ)*cos(θ) + (g*sin(θ)/r) -
        (L*(β^2)*(cos(ζ)*cos(θ)*cos(φ) + sin(ζ)*sin(θ))/r)
    du[3]=dφ=ψ
    du[4]=dψ=(L*(β^2)*cos(ζ)*sin(φ)/(r*sin(θ))) - (2*ω*ψ*cot(θ))
end

#time interval
tspan=(0.0,45.0)

#solving
prob=ODEProblem(lagrangian!,u0,tspan)
sol=solve(prob,saveat=0.1)

#post-processing
U=sol[1:end,:]
(θ,ω,φ,ψ)=[U[x,:] for x in 1:size(U,1)]
t=sol.t
r=5 .* ones(size(t))
β=dζ=β0
ζ=ζ0.+β.*t
x=r.*sin.(θ).*cos.(φ)-L.*cos.(ζ)
y=r.*sin.(θ).*sin.(φ)
z=r.*cos.(θ)+L.*sin.(ζ)
ut=U[:,end]
rt=r[end]
vt=-v0


# changing coordinate frames (no change)
trajectory=hcat(x,y,z)
FC=(rt,vt,ut,trajectory)

#plotting the solution
 display(plot(x,y,z,label="trajectory",
    xaxis="x (m)",yaxis="y (m)",zaxis="z (m)",
    grid=(:on,:black)))
savefig("./plots/trajectory-2.png")

 p1=plot(sol,vars=(0,1),label="θ",
    color="green", xaxis="t (s)", yaxis="θ (rad)")
 p2=plot(sol,vars=(0,2),label="ω",
    color="orange", xaxis="t (s)", yaxis="ω (rad/s)")
 p3=plot(sol,vars=(0,3),label="φ",
    color="purple", xaxis="t (s)", yaxis="φ (rad)")
 p4=plot(sol,vars=(0,4),label="ψ",
    color="blue", xaxis="t (s)", yaxis="ψ (rad/s)")
display(plot(p1,p2,p3,p4,layout=(2,2)))
savefig("./plots/plots-2.png")

#energy plot
KE=0.5*m*((L.*β).^2 .+ (r.*ω).^2 .+ (r.*ψ.*sin.(θ)).^2 .+
     2*L.*r.*β.*(ω.*sin.(ζ).*cos.(θ).*cos.(φ)
     .-ω.*cos.(ζ).*sin.(θ).-β.*sin.(θ).*sin.(ζ).*sin.(φ)))
PE=m*g.*z
TE=KE.+PE
```

```
p7=plot(t,KE,label="KE",color="black",
    xaxis="t (s)", yaxis="KE (J)")
p8=plot(t,PE,label="PE",color="green",
    xaxis="t (s)", yaxis="PE (J)")
p9=plot(t,TE,label="TE",color="red",
    xaxis="t (s)", yaxis="TE (J)")
display(plot!(p7,p8,p9))
savefig("./plots/energyplot-2.png")

return FC

end
```

Script for spherical pendulum with changing length - Stage 3

```
using DifferentialEquations
using Plots
plotlyjs()

function ex2_case_3(IC)

#constants
g=9.81        #acceleration due to gravity [ms^-2]
m=3000        #mass of capsule [kg]

#initial conditions
r_0,v_0,u_0,traj=IC #traj-trajectory
#of capsule till now-not needed for calculation

#function to compute lagrangian
function lagrangian!(du,u,p,t)
    θ,ω,φ,ψ=u
    v=v_0
    r=r_0+v*t
    du[1]=dθ=ω
    du[2]=dω=sin(θ)*cos(θ)*ψ^2-((2*v*ω)/r)+((g*sin(θ))/r)
    du[3]=dφ=ψ
    du[4]=dψ=-((2*v*ψ)/r)-(2*cot(θ)*ω*ψ)
end

#time interval
tspan=(0.0,4.0)

#defining the problem and solving
prob=ODEProblem(lagrangian!,u_0,tspan)
sol=solve(prob,saveat=0.1)

#extracting solution
U=sol[1:end,:]
(θ,ω,φ,ψ)=[U[x,:] for x in 1:size(U,1)]
t=sol.t
r=r_0 .+ v*t
v=v*ones(size(t))
x=r.*sin.(θ).*cos.(φ)
y=r.*sin.(θ).*sin.(φ)
z=r.*cos.(θ)
u_t=U[:,end]
r_t=r[end]
v_t=v[end]

# changing coordinate frames
x_p =x.+5
y_p=y
z_p=z.+(5*(3^0.5))
trajectory=hcat(x_p,y_p,z_p)
FC=(r_t,v_t,u_t,trajectory)

#plotting the solution
 display(plot(x,y,z,label="trajectory",
    xaxis="x (m)",yaxis="y (m)",
    zaxis="z (m)",grid=(:on,:black)))
savefig("./plots/trajectory-3.png")

p1=plot(sol.t,r,label="r",
    color="black",xaxis="t (s)", yaxis="r (m)")
p2=plot(sol.t,v,label="v",
    color="red",xaxis="t (s)", yaxis="v (m/s)")
p3=plot(sol,vars=(0,1),label="θ",
    color="green", xaxis="t (s)", yaxis="θ (rad)")
p4=plot(sol,vars=(0,2),label="ω",
    color="orange", xaxis="t (s)", yaxis="ω (rad/s)")
p5=plot(sol,vars=(0,3),label="φ",
    color="purple", xaxis="t (s)", yaxis="φ (rad)")
p6=plot(sol,vars=(0,4),label="ψ",
    color="blue", xaxis="t (s)", yaxis="ψ (rad/s)")
display(plot(p1,p2,p3,p4,p5,p6,layout=(3,2)))
savefig("./plots/plots-3.png")

#energy plot
KE=0.5*m*(v.^2 .+ (r.*ω).^2 .+ (r.*ψ.*sin.(θ)).^2)
PE=m*g.*z
TE=KE.+PE
p7=plot(t,KE,label="KE",color="black",
```

```
    xaxis="t (s)", yaxis="KE (J)")
p8=plot(t,PE,label="PE",color="green",
    xaxis="t (s)", yaxis="PE (J)")
p9=plot(t,TE,label="TE",color="red",
    xaxis="t (s)", yaxis="TE (J)")
display(plot!(p7,p8,p9))
savefig("./plots/energyplot-3.png")

return FC

end
```

Script for all stages together

```
using DifferentialEquations
using Plots
plotlyjs()

# include the case files
include("ex2-case-1.jl")
include("ex2-case-2.jl")
include("ex2-case-3.jl")

#initial conditions
r_0=6
v_0=-0.25
u_0=[17*π/18,0.1,0.1,0.1]

#calling the functions to run them
IC=[r_0,v_0,u_0]
IC2=ex2_case_1(IC)
IC3=ex2_case_2(IC2)
FC=ex2_case_3(IC3);

#overall trajectory
t1=IC2[4]
t2=IC3[4]
t3=FC[4]
t=vcat(t1,t2,t3)
display(plot(t[:,1],t[:,2],t[:,3],label="trajectory",
    xaxis="x (m)",yaxis="y (m)",zaxis="z (m)",grid=(:on,:black)))
savefig("./plots/trajectory.png")
```

Script for error control

```
using DifferentialEquations
using Plots
plotlyjs()



#constants
g=9.81        #acceleration due to gravity [ms^-2]
m=3000        #mass of capsule [Kg]

#initial conditions
u₀=[3,0.1,0.1,0.5]
r₀=6
v₀=0

#function to compute lagrangian
function lagrangian!(du,u,p,t)
    θ,ω,φ,ψ=u
    v=v₀
    r=r₀+v*t
    du[1]=dθ=ω
    du[2]=dω=sin(θ)*cos(θ)*ψ^2-((2*v*ω)/r)+((g*sin(θ))/r)
    du[3]=dφ=ψ
    du[4]=dψ=-((2*v*ψ)/r)-(2*cot(θ)*ω*ψ)
end

#time interval
tspan=(0.0,20.0)

#defining the problem and solving
prob=ODEProblem(lagrangian!,u₀,tspan)
sol1=solve(prob,saveat=0.1,reltol=1e-1,abstol=1e-4)
sol2=solve(prob,saveat=0.1)
sol3=solve(prob,saveat=0.1,reltol=1e-5,abstol=1e-8)

#extracting solution
U1=sol1[1:end,:]
(θ1,ω1,φ1,ψ1)=[U1[x,:] for x in 1:size(U1,1)]
t=sol1.t

U2=sol2[1:end,:]
(θ2,ω2,φ2,ψ2)=[U2[x,:] for x in 1:size(U2,1)]
t=sol2.t

U3=sol3[1:end,:]
(θ3,ω3,φ3,ψ3)=[U3[x,:] for x in 1:size(U3,1)]
t=sol3.t


#plotting the solutions for different tolerances

p1=plot(sol1,vars=(0,1),label="rtol=1e-1,atol=1e-4",
    color="green", xaxis="t (s)", yaxis="θ (rad)")
p2=plot!(p1,sol2,vars=(0,1),label="rtol=1e-3,atol=1e-6",
    color="blue", xaxis="t (s)", yaxis="θ (rad)")
p3=plot!(p2,sol3,vars=(0,1),label="rtol=1e-5,atol=1e-8",
    color="red", xaxis="t (s)", yaxis="θ (rad)")
display(plot(p3))
savefig("./plots/error_control_1.png")

p4=plot(sol1,vars=(0,3),label="rtol=1e-1,atol=1e-4",
    color="green", xaxis="t (s)", yaxis="φ (rad)")
p5=plot!(p4,sol2,vars=(0,3),label="rtol=1e-3,atol=1e-6",
    color="blue", xaxis="t (s)", yaxis="φ (rad)")
p6=plot!(p5,sol3,vars=(0,3),label="rtol=1e-5,atol=1e-8",
    color="red", xaxis="t (s)", yaxis="φ (rad)")
display(plot(p6))
savefig("./plots/error_control_2.png")
```