Engineering Dynamics

Ryan C. Cooper

Nov 16, 2020

CONTENTS

Applied Mechanics II (or Dynamics) is the study of how things move and interact. We are limiting our study to Newtonian mechanics. We won't consider quantum effects like wave-particle duality or relativistic effects. Our current interest is to describe free and constrained motion much less than the speed of light and with mass much larger than an atom, but much smaller than the sun (~1e-27 < m < 2e30 kg).

The two founding scientists of classical dynamics are Galileo Galilei (1564-1642) and Isaac Newton (1642-1726). Galileo described the geometry of moving objects and helped define our understanding of kinematics. Newton defined three kinetic laws that helped describe how force, impact, and energy relate to changes in motion.

This course explores the geometry of motion and the effect of forces, impact, and energy in engineering. **Kinematics** is the study of the geometry of motion. **Kinetics** is the study of forces, impacts, and energy on objects.

CHAPTER

ONE

ABOUT THE INSTRUCTOR

Ryan C. Cooper, Ph.D.Assistant Professor-in-ResidenceMechanical Engineering DepartmentUniversity of Connecticut191 Auditorium rdEngineering Building II room 314Storrs, CT 06269

email: mailto:ryan.c.cooper@uconn.edu

Office hours: Mon, Wed, Thu 9-11am

Preferred contact: post discussion questions on Piazza, for personal questions reach out via email.

CHAPTER

TWO

MODULE CONTENT

This textbook has one introduction module and five learning modules. Each learning module includes:

1. one reading assignment with discussion in Piazza
2. lecture and tutorial videos
3. one homework assigment
4. one quiz based upon the homework, readings, and videos
5. one Adams report

Make sure you reflect on things you learned, what you found easy, and what you struggled with after each module in the **Summary and Checklist**.

CHAPTER

THREE

MODULE 0 - GETTING STARTED

This introduction will get you up to speed on the tools we are using to explore dynamics: HuskyCT, Google Documents, Adams multibody dynamics, and Piazza discussion boar4.

3.1 Overview

Applied Mechanics II (or Dynamics) is the study of how things move and interact. We are limiting our study to Newtonian mechanics. We won't consider quantum effects like wave-particle duality or relativistic effects. Our current interest is to describe free and constrained motion much less than the speed of light and with mass much larger than an atom, but much smaller than the sun (~1e-27 < m < 2e30 kg).

The two founding scientists of classical dynamics are Galileo Galilei (1564-1642) and Isaac Newton (1642-1726). Galileo described the geometry of moving objects and helped define our understanding of kinematics. Newton defined three kinetic laws that helped describe how force, impact, and energy relate to changes in motion.

This introduction will get you up to speed on the tools we are using to explore dynamics: HuskyCT, Google Documents, Adams multibody dynamics, and Piazza discussion board.

3.2 Objectives

Upon completion of this module you will be able to

1. Recognize the course objectives, requirements, grading policy, required software, and materials
2. View video content and submit answers in a Google Form
3. Use the discussion board on Piazza
4. Locate course areas and features, and recognize their general purposes
5. Download and install Respondus Lockdown Browser with Monitor to ensure you can successfully take the graded quizzes.
6. Scan and upload a hand-written assignment
7. Open an Adams file and save a figure
8. Submit a Google Document in HuskyCT

3.3 Activities

- Read the syllabus and post a discussion response on Piazza
- Acquaint yourself with the course instructor.
- Verify your computer settings are HuskyCT compatible.
- Read about the course's organization and tools.
- Take a 5-minute practice quiz
- Submit a practice quiz.
- Download Respondus Lockdown Browser on the computer you plan to use in this course.
- Review the University of Connecticut's academic policies.
- Watch the Introduction videos and answer the questions
- Finish a tutorial on Adams and complete the Google form
- Turn in a 1-page report that has a figure from the Adams tutorial

3.4 Participation Videos

1. Course Tour Video{target="_blank"}
2. Adams tour: rolling on incline{target="_blank"} rolling.bin example file{target="_blank"}

CHAPTER

FOUR

MODULE 1 - DYNAMIC EQUATIONS AND DEFINITIONS

ball rolling on plane

In this module, we want to understand what makes objects move and how we can describe motion. In previous physics courses, you might remember the kinematic equations that describe the motion of a free-falling object. How do these equations relate to a ball rolling on the ground? or a domino falling over?

As an engineer, you will need to design objects that move or interact with moving objects. You need quantitative descriptions for impacts, forces, and work. This module will give you the tools to describe motion, forces, work, and impact.

4.1 Outline

1. Kinematics
    1. define position
    2. velocity
    3. acceleration
    4. degrees of freedom and coordinates
2. Kinetics
    1. Newton-Euler equations
    2. Impulse-momentum
    3. Work-energy
    4. equations of motion = second order diff eq
3. Solving dynamic problem
    1. Newton-Euler equations define eom
    2. Use ODE solution for x(t)

4.1.1 Geometry of motion - kinematics

Position

Classical physics describes the position of an object using three independent coordinates e.g.

$$\mathbf{r}_{P/O} = x\hat{i} + y\hat{j} + z\hat{k} \tag{4.1}$$

where $\mathbf{r}_{P/O}$ is the position of point P *with respect to the point of origin* O, $x,\ y,\ z$ are magnitudes of distance along a Cartesian coordinate system and $\hat{i},\ \hat{j}$ and $\hat{k}$ are unit vectors that describe three

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

Velocity

The velocity of an object is the change in *position* per length of time.

$$\mathbf{v}_{P/O} = \frac{d\mathbf{r}_{P/O}}{dt} = \dot{x}\hat{i} + \dot{y}\hat{j} + \dot{z}\hat{k} \tag{4.2}$$

> **Note:** The notation $\dot{x}$ and $\ddot{x}$ is short-hand for writing out $\frac{dx}{dt}$ and $\frac{d^2x}{dt^2}$, respectively.

The definition of velocity in equation (**??**) depends upon the change in position of all three independent coordinates, where $\frac{d}{dt}(x\hat{i}) = \dot{x}\hat{i}$.

> **Note:** Remember the chain rule: $\frac{d}{dt}(x\hat{i}) = \dot{x}\hat{i} + x\dot{\hat{i}}$, but $\dot{\hat{i}} = 0$ because this unit vector is not changing direction. You'll see other unit vectors later that do change.

Example - GPS vs speedometer

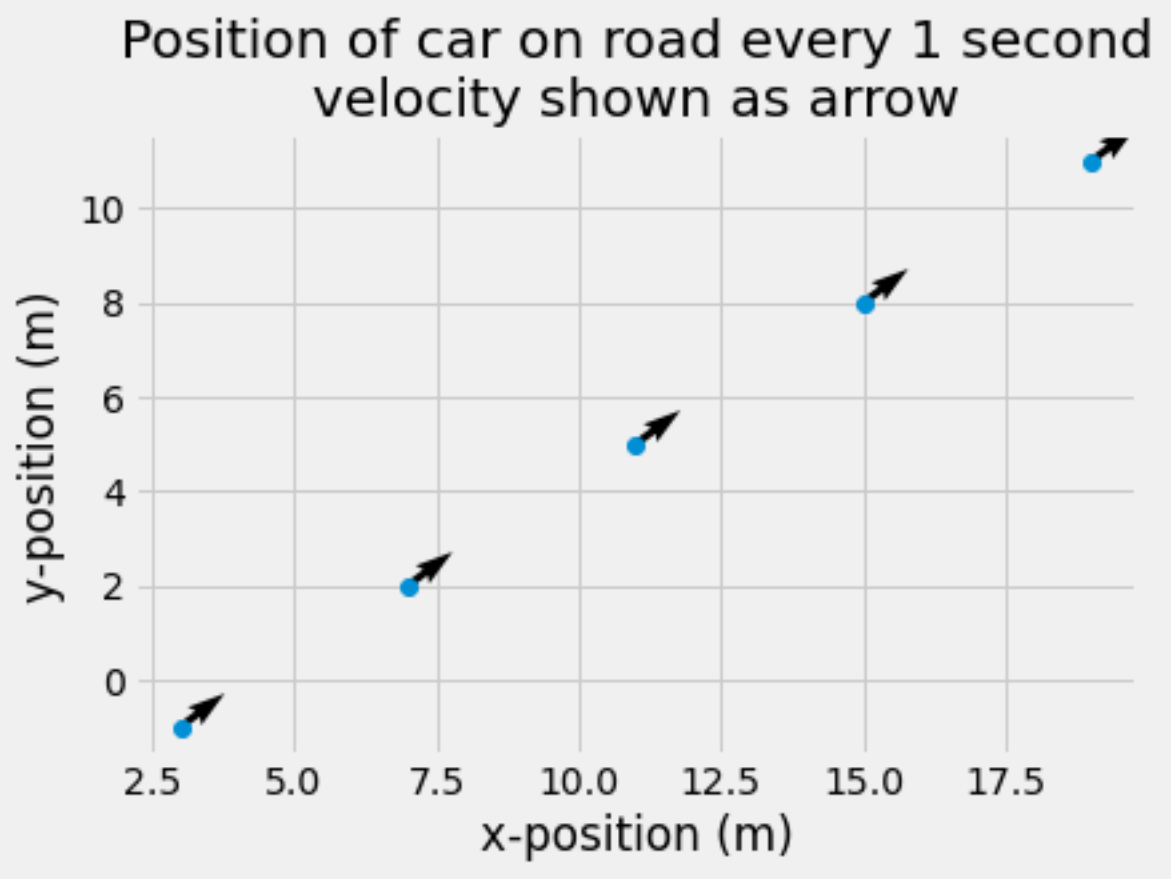
You can find velocity based upon postion, but you can only find changes in position with velocity. Consider tracking the motion of a car driving down a road using GPS. You determine its motion and create the position, $\mathbf{r} = x\hat{i} + y\hat{j}$, where

$x(t) = 4t + 3$ and $y(t) = 3t - 1$

To get the velocity, calculate $\mathbf{v} = \dot{\mathbf{r}}$

$\mathbf{v} = 4\hat{i} + 3\hat{j}$

```
t = np.arange(0,5)
x = 4*t + 3
y = 3*t -1
plt.plot(x,y,'o')
plt.quiver(x,y,4,3)
plt.title('Position of car on road every 1 second'+
    '\nvelocity shown as arrow')
plt.xlabel('x-position (m)')
plt.ylabel('y-position (m)');
```



Speed

The speed of an object is the magnitude of the velocity,

$|\mathbf{v}_{P/O}| = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}$

Acceleration

The acceleration of an object is the change in velocity per length of time.

$\mathbf{a}_{P/O} = \frac{d\mathbf{v}_{P/O}}{dt} = \ddot{x}\hat{i} + \ddot{y}\hat{j} + \ddot{z}\hat{k}$

where $\ddot{x} = \frac{d^2x}{dt^2}$ and $\mathbf{a}_{P/O}$ is the acceleration of point P *with respect to the point of origin* O.

Rotation and Orientation

The definitions of position, velocity, and acceleration all describe a single point, but dynamic engineering systems are composed of rigid bodies is needed to describe the position of an object.

```
from IPython.core.display import SVG

SVG(filename='./images/position_angle.svg')
```

```
<IPython.core.display.SVG object>
```

In the figure above, the center of the block is located at $r_{P/O} = x\hat{i} + y\hat{j}$ *in both the left and right images, but the two locations are not the same. The orientation of the block is important for determining the position of all the material points.*

In general, a rigid body has a *pitch*, *yaw*, and *roll* that describes its rotational orientation, as seen in the animation below. We will revisit 3D motion in Module_05

```
from IPython.display import YouTubeVideo
vid = YouTubeVideo("li7t--8UZms?loop=1")
display(vid)
```

```
<IPython.lib.display.YouTubeVideo at 0x7f4888ac82b0>
```

Rotation in planar motion

Our initial focus is planar rotations e.g. yaw and roll are fixed. For a body constrained to planar motion, you need 3 independent measurements to describe its position e.g. x, y, and θ

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from IPython.core.display import SVG
```

4.1.2 Mechanical advantage

In the diagram below, there is a 50-N force, F, applied downwards on a constrained linkage system. There is a restoring force, R, that maintains a slow change in the angle of the mechanism, θ, as it changes from 89^o to -89^o. Your goal is to determine the necessary restoring force R as a function of angle θ.

Kinematics - geometry of motion

This system has two rigid bodies connected at point A by a pin and the whole system is pinned to the ground at O. Point B slides along a horizontal surface. The total degrees of freedom are

3 (link 1) + 3 (link 2) - 2 constraints (O) - 2 constraints (A) - 1 constraint (B) - 1 constraint (θ-fixed) = **0 DOF**

For a given angle θ, there can only be one configuration in the system. Create the constraint equations using the relative position of point A as such

$\mathbf{r}_A = \mathbf{r}_{A/B} + \mathbf{r}_B$

where

$\mathbf{r}_A = L(\cos\theta\hat{i} + \sin\theta\hat{j})$

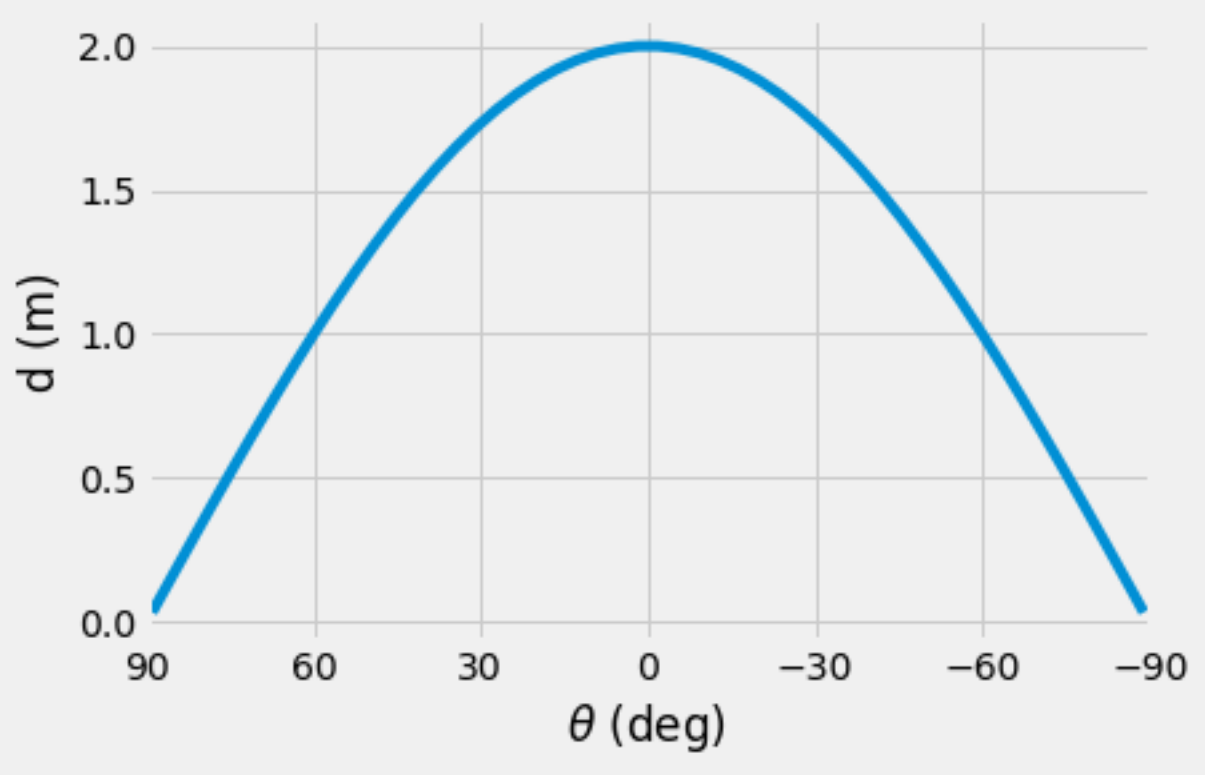
$\mathbf{r}_B = d\hat{i}$

$\mathbf{r}_{A/B} = L(-\cos\theta\hat{i} + \sin\theta\hat{j})$

solving for the $\hat{i}-$ and $\hat{j}-$components creates two equations that describe the state of the system

- $\theta = \theta$ which states because we have an isosceles triangle, two angles have to be equal
- $d = 2L\cos\theta$ so $\mathbf{r}_B = 2L\cos\theta\hat{i}$

```
theta = np.linspace(89,-89)*np.pi/180
d = 2*1*np.cos(theta)
plt.plot(theta*180/np.pi, d)
plt.xticks(np.arange(-90,91,30))
plt.xlabel(r'$\theta$ (deg)')
plt.xlim(90,-90)
plt.ylabel('d (m)');
```



Kinetics - applied forces and constraint forces

The applied force, $F = 50\ N$ is constant, but R is dependent upon the geometry of the system. You solved for the kinematics in the first part, here you can use the Newton-Euler equations to solve for R given θ and F. Separate the system into the left and right links,

The Newton-Euler equations:

- $\mathbf{F} = m\mathbf{a} = \mathbf{0}$ links moving slowly
- $M_G = I\alpha = 0$ links rotating slowly

Newton-Euler equations for the left bar:

1. $\mathbf{F}\cdot\hat{i} = N_{x1} + N_{x2} = 0$
2. $\mathbf{F}\cdot\hat{i} = N_{y1} + N_{y2} - F = 0$
3. $M_O = l\hat{b}_1 \times (-F\hat{j} + N_{x2}\hat{i} + N_{y2}\hat{j}) = 0$

Newton-Euler equations for the right bar:

1. $\mathbf{F} \cdot \hat{i} = -N_{x2} - R = 0$
2. $\mathbf{F} \cdot \hat{i} = -N_{y2} + N_{y3} = 0$
3. $M_A = l\hat{c}_1 \times (-R\hat{i} + N_{y3}\hat{j}) = 0$

   Note: Don't count F twice! You can use the applied force on the left or right, but not both. Try solving the equations placing it on the right bar.

The four equations for $\mathbf{F}$ relate the reaction forces $\mathbf{N}_1$, $\mathbf{N}_2$, *and* $\mathbf{N}_3$. The two moment equations relate R to F and θ as such

$N_{y2} L \cos\theta - RL \sin\theta = 0$

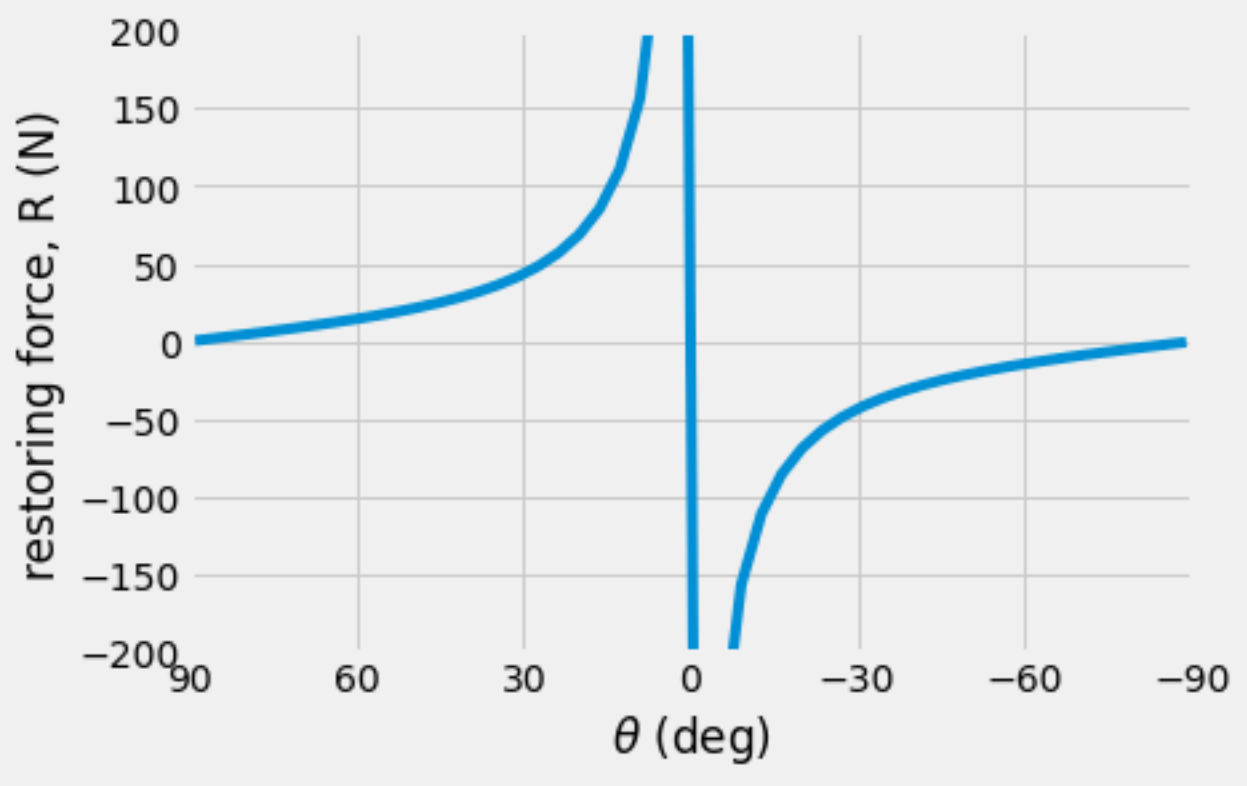
and

$-FL \cos\theta + N_{y2} \cos\theta + RL \sin\theta = 0$

combining there results

$F \cos\theta = 2R \sin\theta \rightarrow R = \frac{F}{2} \cot\theta$

```
R = 50/2*np.tan(theta)**-1
plt.plot(theta*180/np.pi, R)
plt.xticks(np.arange(-90,91,30))
plt.xlabel(r'$\theta$ (deg)')
plt.ylabel('restoring force, R (N)')
plt.xlim(90,-90)
plt.ylim(-200,200);
```



Wrapping up

Take a look at the mechanical advantage and disadvantage this system can create. For angles close to $\theta \approx 90^o$, the restoring force is close to zero. In this case, the applied force is mostly directed at constraints on the system. When the angles are close to $\theta \approx 0^o$, the required restoring force can be $> 100\times$ the input force.

Have you seen this type of linkage system in engineering devices?

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
plt.rcParams.update({
  "text.usetex": True,
  "font.sans-serif": ["Helvetica"]})
```

4.1.3 Driving forces for moving systems

In this case study, you want to accelerate a 0.1-kg flywheel with a piston. The desired acceleration of the flywheel is $\alpha = 50\ rad/s^2$. The piston is attached to the link and creates a horizontal driving force. This example demonstrates how we can describe constraints in mathematical forms.

We have two moving bodies:

1. a flywheel with radius $r = 0.2\ m$ and mass $m = 0.1\ kg$
2. a massless connecting link

Both objects move in a horizontal plane, so their positions have two degrees of freedom to describe position and one degree of freedom that describes orientation.

$DOF = 3 + 3 = 6$

```
from IPython.core.display import SVG

SVG(filename='./images/piston-flywheel.svg')
```

```
<IPython.core.display.SVG object>
```

Describing constraints

There are six degrees of freedom for a flywheel and a link, but there are 6 constraints on the system's motion:

1. the flywheel is pinned to the ground in the x-dir
2. the flywheel is pinned to the ground in the y-dir
3. the top of the link is pinned to the flywheel
4. the top of the link is pinned to the flywheel
5. the bottom of the link slides along the horizontal line
6. the angle of the flywheel has acceleration, $\alpha = 50\ rad/s^2$

$system\ DOF = 6 - 6 = 0\ DOF$

> **Note:** In general, a pin in a planar system creates 2 constraints on motion.

You should recognize at this point that there are *0 differential equations to solve*. Once you have calculated the kinematics based upon the system constraints, you can plug in the values and solve for force of the piston. So, start with the kinematic description of motion.

$\mathbf{r}_2 = \mathbf{r}_{2/3} + \mathbf{r}_3$

```
SVG(filename = './images/flywheel-constraints.svg')
```

```
<IPython.core.display.SVG object>
```

$r(\sin\theta\hat{i} - \cos\theta\hat{j}) = -L(\cos\theta_L\hat{i} - \sin\theta_L\hat{j}) + d\hat{i}$

this description creates two independent equations

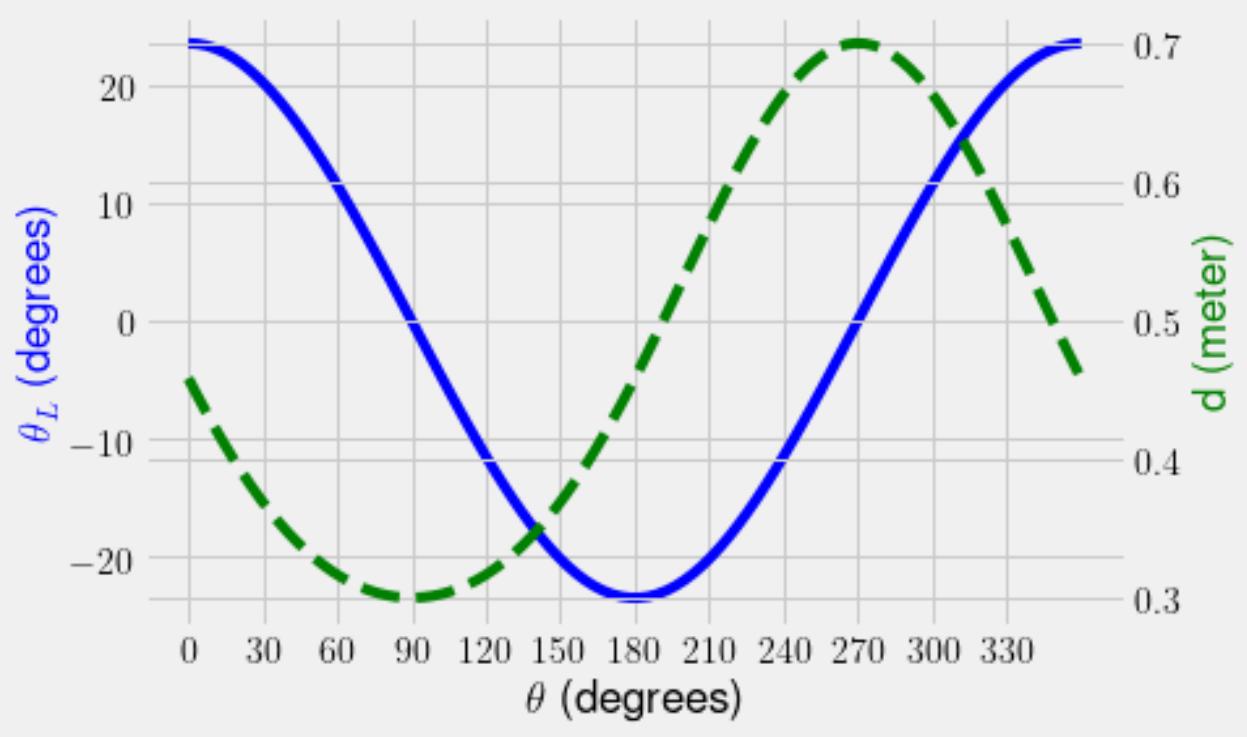
1. $-r\sin\theta = -L\cos\theta_L + d$
2. $r\cos\theta = -L\sin\theta_L$

The constraint on θ says that $\theta(t) = \frac{\alpha t^2}{2}$. You need to solve for θ_L and d to determine the full state of the system.

$\theta_L = \sin^{-1}\left(\frac{r}{L}\cos\theta\right)$

$d = L\cos\theta_L - r\sin\theta$

```
r = 0.2
L = 0.5
theta = np.linspace(0,2*np.pi,100)
thetaL = np.arcsin(r/L*np.cos(theta))
d = L*np.cos(thetaL)-r*np.sin(theta)
f, ax = plt.subplots()
ax.plot(theta*180/np.pi, thetaL*180/np.pi,'b-',label = r'\theta_L')
ax.set_xlabel(r'$\theta$ (degrees)')
ax.set_ylabel(r'$\theta_L$ (degrees)',color = 'blue')
plt.xticks(np.arange(0,360,30))
ax2 = ax.twinx()
ax2.plot(theta*180/np.pi, d, 'g--', label = 'piston d')
ax2.set_ylabel('d (meter)', color = 'green');
plt.show()
```



Exercise:

What is the angular velocity and angular acceleration of the link? $\dot{\theta}_L$ and $\ddot{\theta}_L$

Now, we have solved all of the kinematics we need to solve for the applied piston force. Sometimes, you can look at the Newton-Euler equations for the syatem as a whole, but here we need to separate each component and sum forces and moments.

Flywheel:

$\sum \mathbf{F} = \mathbf{0}$

$\sum M_G = I\alpha$

> **Note:** If a body is moving, you either have to sum moments about a fixed point or its center of mass. Otherwise, the description of angular acceleration is more involved.

Link:

$\sum \mathbf{F} = \mathbf{0}$

$\sum M = 0$

> **Note:** when links, cables, or pulleys are assumed to be massless they still obey Newtonian mechanics, but momentum is always zero. So the sum of forces or moments is equal to 0.

The three kinetic equations for the wheel are as such,

1. $\sum \mathbf{F} \cdot \hat{i} = F_{2x} + F_{1x} = 0$
2. $\sum \mathbf{F} \cdot \hat{j} = F_{2y} + F_{1y} = 0$

3. $\sum M_1 = r\hat{e}_r \times (F_{2x}\hat{i} + F_{2y}\hat{j}) = \frac{mr^2}{2}\alpha$

and the three kinetic equations for the link are as such,

1. $\sum \mathbf{F} \cdot \hat{i} = R - F_{2x} = 0$
2. $\sum \mathbf{F} \cdot \hat{j} = -F_{2y} - F_{piston} = 0$
3. $\sum M_2 = L\hat{b}_1 \times (-F_{piston}\hat{i} + R\hat{j}) = 0$

The third equation for the link is rearranged to solve for $R = F\tan\theta_L$. The third equation for the flywheel is rearranged to solve for F as such

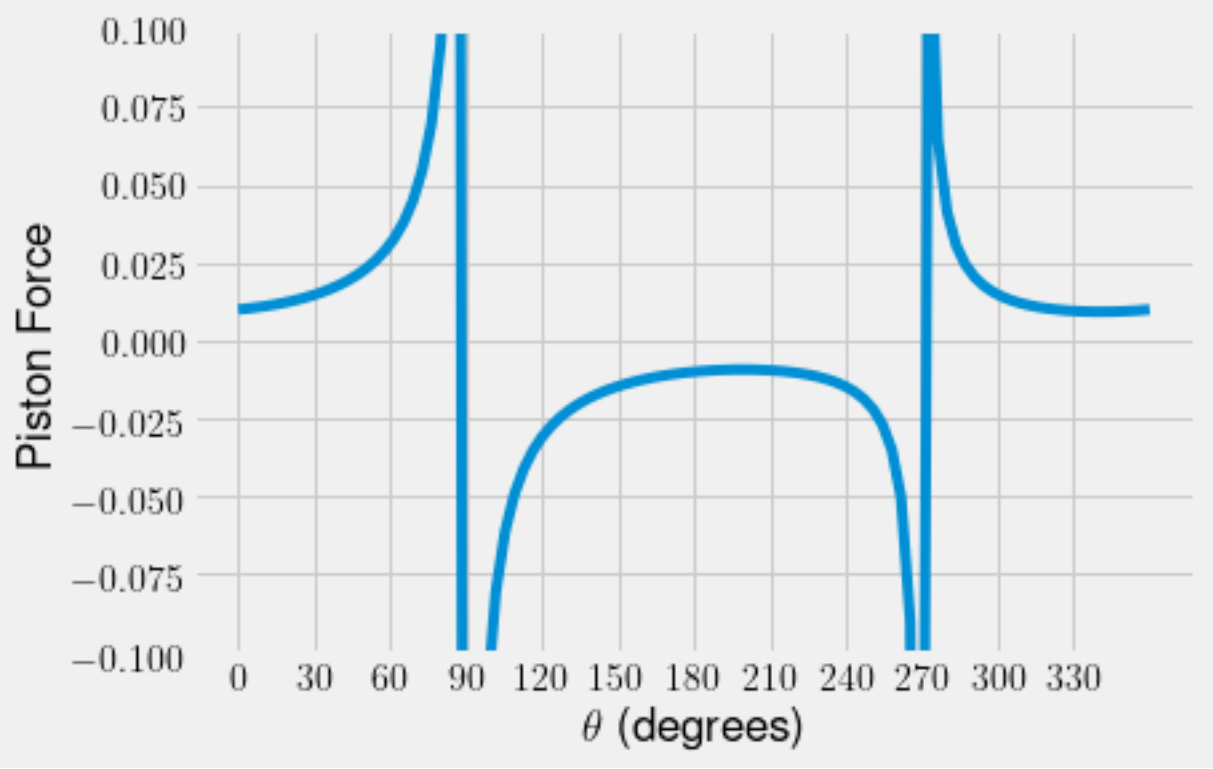
$rF\cos\theta - rF\tan\theta_L \sin\theta = \frac{mr^2}{2}\alpha$

finally arriving at

$F = \frac{mr\alpha}{2}(\cos\theta - \tan\theta_L \sin\theta)^{-1}$

Plotted below as a function of flywheel angle, θ,

```
m = 0.1
F = m*r/2*(np.cos(theta)-np.tan(thetaL)*np.sin(theta))**(-1)
plt.plot(theta*180/np.pi, F)
plt.ylim(-0.1,0.1)
plt.xticks(np.arange(0,360,30))
plt.xlabel(r'$\theta$ (degrees)')
plt.ylabel('Piston Force');
```



CHAPTER

FIVE

MODULE 2 - CONSTRAINED DYNAMIC MOTION

pendulum motion

1. Types of constraints and degrees of freedom
2. pin joint
3. sliding constraint
4. rigid attachment
5. Greubler count
6. Equations of motion with constraints
7. count degrees of freedom choose variables
8. define forces and kinematics with chosen variable
9. solve for constrained equation of motion
10. Using smart coordinate systems
11. using mixed unit vectors
12. radial and spherical
13. rotating systems

CHAPTER

SIX

MODULE 3 - CONSTRAINED ENGINEERING SYSTEMS

4-bar linkage

1. Engineering systems with many moving parts
    1. constrain the motion (0-DOF)
    2. to maintain motion need constraint forces
    3. constraints doing work vs constraint force
2. Mechanical advantage
    1. gear ratios
    2. levers
    3. linkages
3. Relative motion
    1. fixing coordinate systems to moving parts
    2. absolute and relative motion
    3. constraint forces and work done

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import pretty_plots # script to set up LaTex and increase line-width and font size
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-5108677927e6> in <module>
      2 import matplotlib.pyplot as plt
      3 from scipy.integrate import odeint
----> 4 import pretty_plots # script to set up LaTex and increase line-width and font␣
↪size

ModuleNotFoundError: No module named 'pretty_plots'
```

```
pretty_plots.setdefaults()
```

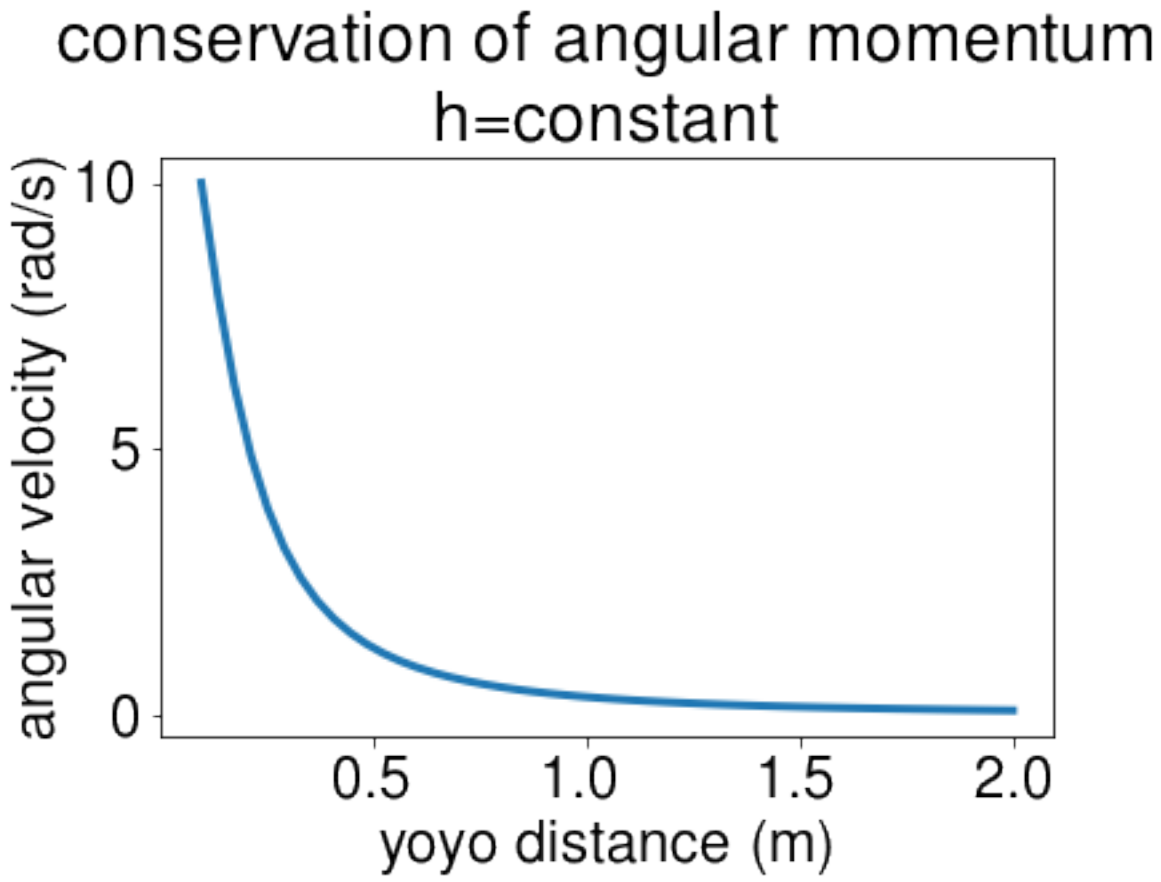
6.1 Numerical Solution of Yoyo despinning

$\ddot{r} = r\dot{\theta}^2$

$\ddot{r} = r\left(\frac{2mr_0^2+MR^2/2}{2mr^2+MR^2/2}\right)^2 \dot{\theta}(0)$

Here we plot the relation between angular velocity and distance from center of cylinder just based upon conservation of angular momentum:

```
m=0.1 #kg
M=1 #kg
R=0.1 #meter
w0=10 # rad/s
r=np.linspace(0.1,2)
h0=(2*m*r[0]**2+M*r[0]**2/2)*w0

wt = h0/(2*m*r**2+M*R**2/2)

plt.plot(r,wt)
plt.xlabel('yoyo distance (m)')
plt.ylabel('angular velocity (rad/s)')
plt.title('conservation of angular momentum\n h=constant');
```



6.2 Define the state and d/dt(state)

In this part, we define the second order differential equation as a state-space form

the state = $[r,\ \dot{r}]$

and the derivative of the state is

d/dt(state) = $[\dot{r},\ \ddot{r}] = [\dot{r},\ \frac{F_r}{m}]$

We call the function, `yoyo_ode(y,t)`, where `y` is the state and `t` is the current time.

```
def yoyo_ode(y,t):
    '''define d2r/dt2= r*(h0/2m)^2/(M*R^2/4m+r^2)^2'''
    dr=np.zeros(np.shape(y))
    dr[0]=y[1]
    dr[1]=y[0]*h0**2/(2*m*y[0]**2+M*R**2/2)**2
    return dr
```

```
yoyo_ode([0.15,1],0)
```

```
array([ 1.        ,  8.14404432])
```

The function `odeint` integrates our `yoyo_ode` based upon the initial condtions,

$[r(0),\ \dot{r}(0)] = [0.1\ m,\ 0\ m/s]$

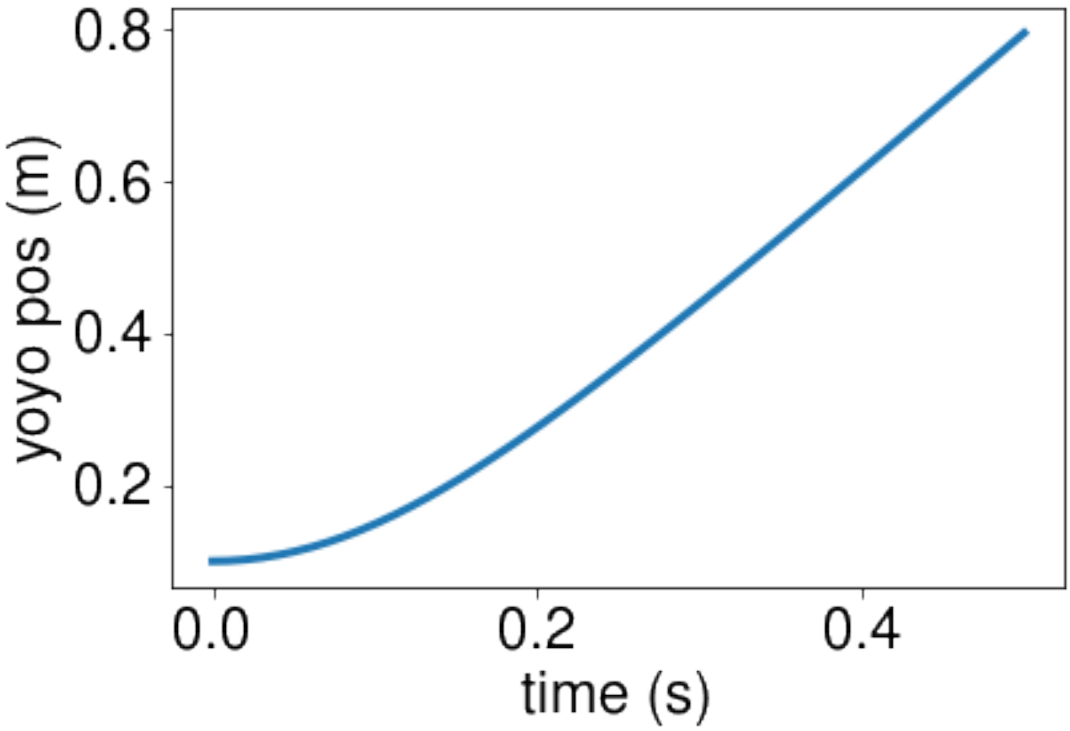
and the time span of interest,

$t = [0 - 0.5\ s]$

in the line, `t=np.linspace(0,0.5)`

```
t=np.linspace(0,0.5)
r=odeint(yoyo_ode,[0.1,0],t)

plt.plot(t,r[:,0])
plt.xlabel('time (s)')
plt.ylabel('yoyo pos (m)')
```

```
Text(0,0.5,'yoyo pos (m)')
```



6.3 To save to file from python output

Have to join the time, r, and $\dot{r}$ into an array then save to a file:

```
np.savetxt('t_r_rdot.csv',np.array([t,r[:,0],r[:,1]]).T,delimiter=',')
```

This line of code organizes a comma-separated-value file into the file t_r_rdot.csv with no headers.

time (s)	r (m)	$\dot{r}$ (m/s)
0	0.1	0
...	...	...

CHAPTER

SEVEN

MODULE 4 - MOMENTUM AND ENERGY

yoyo despin

This module uses energy and momentum methods directly.

1. Energy and Momentum are different forms of Newton's second law
    1. Impulse-momentum is Fdt=dp
    2. Work-energy is Fdx=dT
    3. momentum is a vector
    4. energy is a scalar
2. Describes states of system
3. Impulse happens over short time in impact
4. energy in conservative system ia conserved
5. combining methods

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

7.1 Numerical Solution of Yoyo despinning

$\ddot{r} = r\dot{\theta}^2$

$\ddot{r} = r\left(\frac{2mr_0^2+MR^2/2}{2mr^2+MR^2/2}\right)^2 \dot{\theta}(0)$

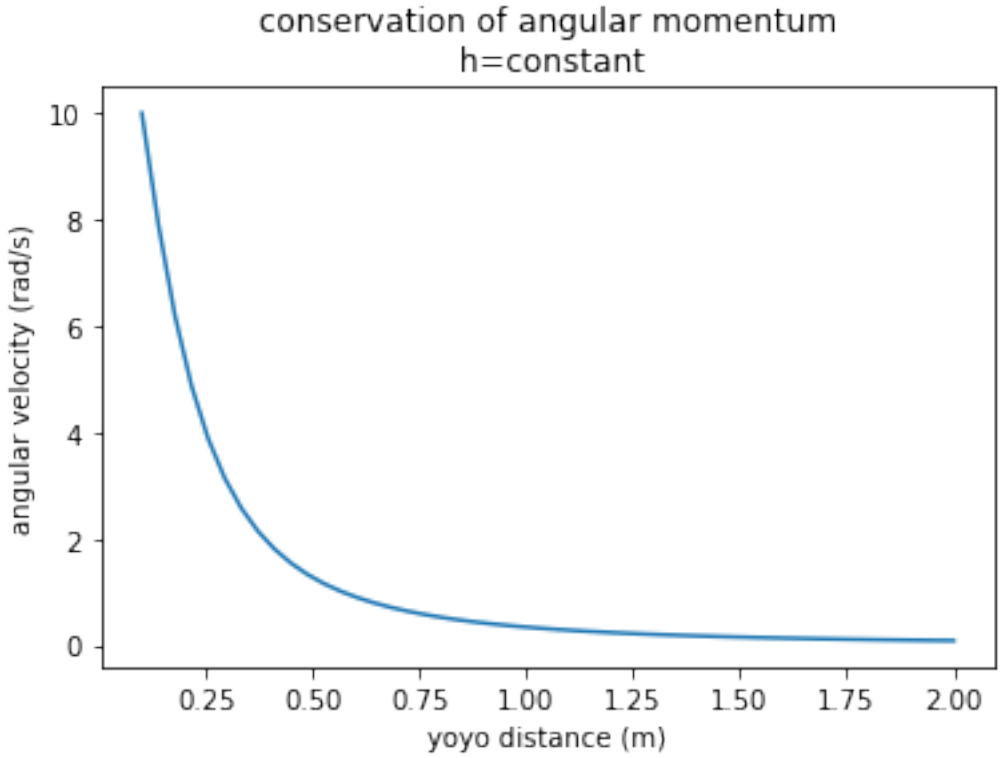
Here we plot the relation between angular velocity and distance from center of cylinder just based upon conservation of angular momentum:

```
m=0.1 #kg
M=1 #kg
R=0.1 #meter
w0=10 # rad/s
r=np.linspace(0.1,2)
h0=(2*m*r[0]**2+M*r[0]**2/2)*w0

wt = h0/(2*m*r**2+M*R**2/2)

plt.plot(r,wt)
```

(continues on next page)

(continued from previous page)

```
plt.xlabel('yoyo distance (m)')
plt.ylabel('angular velocity (rad/s)')
plt.title('conservation of angular momentum\n h=constant');
```



7.2 Define the state and d/dt(state)

In this part, we define the second order differential equation as a state-space form

the state = $[r,\ \dot{r}]$

and the derivative of the state is

d/dt(state) = $[\dot{r},\ \ddot{r}] = [\dot{r},\ \frac{F_r}{m}]$

We call the function, `yoyo_ode(y,t)`, where `y` is the state and `t` is the current time.

```
def yoyo_ode(y,t):
    '''define d2r/dt2= r*(h0/2m)^2/(M*R^2/4m+r^2)^2'''
    dr=np.zeros(np.shape(y))
    dr[0]=y[1]
    dr[1]=y[0]*h0**2/(2*m*y[0]**2+M*R**2/2)**2
    return dr
```

```
yoyo_ode([0.15,1],0)
```

```
array([1.        , 8.14404432])
```

The function `odeint` integrates our `yoyo_ode` based upon the initial condtions,

$[r(0),\ \dot{r}(0)] = [0.1\ m,\ 0\ m/s]$

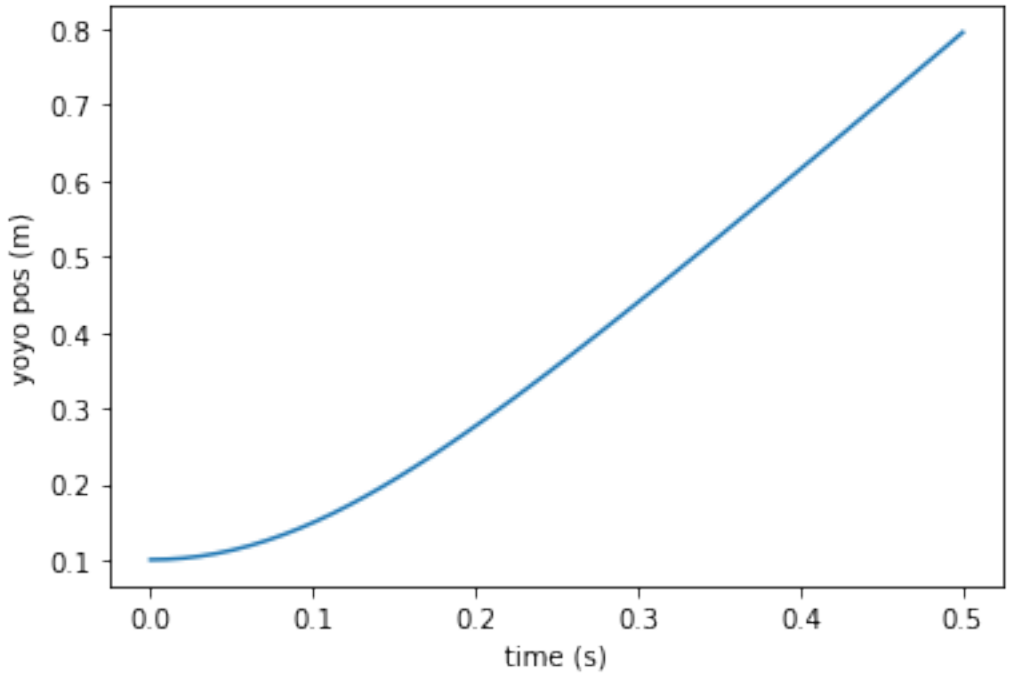
and the time span of interest,

$t = [0 - 0.5\ s]$

in the line, `t=np.linspace(0,0.5)`

```
t=np.linspace(0,0.5)
r=odeint(yoyo_ode,[0.1,0],t)

plt.plot(t,r[:,0])
plt.xlabel('time (s)')
plt.ylabel('yoyo pos (m)')
```

```
Text(0, 0.5, 'yoyo pos (m)')
```



7.3 To save to file from python output

Have to join the time, r, and $\dot{r}$ into an array then save to a file:

```
np.savetxt('t_r_rdot.csv',np.array([t,r[:,0],r[:,1]]).T,delimiter=',')
```

This line of code organizes a comma-separated-value file into the file t_r_rdot.csv with no headers.

time (s)	r (m)	$\dot{r}$ (m/s)
0	0.1	0
...	...	...

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
plt.style.use('fivethirtyeight')
```

7.4 Yoyo despin revisited (cord constraint)

```
from IPython.core.display import SVG

SVG(filename='./yoyo-rocket.svg')
```

```
<IPython.core.display.SVG object>
```

A rocket yoyo-despinning mechanism uses cords wrapped around the payload. These cords unravel and slow the spinning of the rocket. In this tutorial, you will consider the engineering system, conservation of angular momentum, and conservation of energy.

7.5 Engineering system - kinematics

As the yoyo mass unravels, it moves further from the payload. The total distance from the payload center of mass (COM) is described by

$\mathbf{r}_{P/G} = R\hat{e}_R + l\hat{e}_\theta$

where R is the payload radius, l is the length of the cord, and $\hat{e}_R$ and $\hat{e}_\theta$ are unit vectors in a cylindrical coordinate system. The length of the cord depends upon the angle of the payload, θ. Consider a spool of thread rolling across the floor, the thread left on the floor is equal to distance traveled or, $R\theta$. Now, the position of yoyo P is

$\mathbf{r}_{P/G} = R\hat{e}_R + R\theta\hat{e}_\theta$

where θ is the change in angle of the payload after the yoyos are released. The velocity of mass P is $\dot{\mathbf{r}}_{P/G}$, using the transport equation

$\mathbf{v}_{P/G} = \frac{d}{dt}(R\hat{e}_R + R\theta\hat{e}_\theta) + {}^I\omega^C \times (R\hat{e}_R + R\theta\hat{e}_\theta)$

where the total angular velocity is the combination of the payload's angular velocity ω_B and the angular velocity of the yoyo relative to the payload, ${}^B\omega^C = \dot{\theta}\hat{k}$. The addition of payload and yoyo angular velocity is the total

${}^I\omega^C = \omega_B\hat{k} + \dot{\theta}\hat{k}$

Note: Consider a 1-kg payload with radius $R = 0.1\ m$ and two yoyos of mass, $m_P = m_Q = 0.1\ kg$. The system is released at $t = 0\ s$ when the payload is spinning at $\omega_B^0 = 10\ rad/s$.

```
# Set up the Python variables
M = 1
R = 0.1
I = M*R**2/2
m = 0.1
w0 = 10
```

The yoyos P and Q will remain anti-symmetric if released at the same time, so

$\mathbf{r}_{P/G} = -\mathbf{r}_{Q/G},\ |\mathbf{r}_{P/G}| = |\mathbf{r}_{Q/G}|$

and

$\mathbf{v}_{P/G} = -\mathbf{v}_{Q/G},\ v_P = v_Q.$

The equations for position and velocity are the essential kinematic equations to define angular momentum and kinetic energy, the two kinetic equations.

7.6 Engineering system - kinetics (conservation of angular momentum)

The angular momentum is constant because $\sum \mathbf{M}_G = 0 = \frac{d}{dt}\mathbf{h}_G$. The total angular momentum is as such

$\mathbf{h}_G = I_G\omega_B\hat{k} + m_P\mathbf{r}_{P/G} \times \mathbf{r}_{P/G} + m_Q\mathbf{r}_{Q/G} \times \mathbf{r}_{Q/G}$

where $m_P = m_Q = m$

$\mathbf{h}_G = I_G\omega_B\hat{k} + 2mR^2(\omega_B + \theta^2(\omega_B + \dot{\theta})) = (I_G + 2mR^2)\omega_B^0$

where ω_B^0 is the initial spinning rate of the payload, ω_B is the angular speed of the payload after the yoyos are released, θ is the angle of the yoyo cord, and $\dot{\theta}$ is the relative angular velocity of the yoyo cord. At time $t = 0$, the yoyos are release and $\theta = \dot{\theta} = 0$. There are three unknown variables in this angular momentum equation, ω_B, θ, and $\dot{\theta}$. You need another equation, use work-energy.

7.7 Engineering system - kinetics (work-energy formulation)

There is no external work done to the system, the tension in the yoyo cords are internal constraint forces, so the total work done is 0, e.g. for a $T\Delta l$ there is an equal and opposite $-T\Delta l$ on the other side.

$T_1 + W_{1\to 2} = T_2$

$\frac{1}{2}I_G(\omega_B^0)^2 + 2mR^2(\omega_B^0)^2 = \frac{1}{2}I_G(\omega_B)^2 + m(R^2\theta^2(\omega_B + \dot{\theta})^2 + R^2\omega_B^2)$

combining terms and simplifying

$\left(\frac{I_G}{2mR^2} + 1\right)((\omega_B^0)^2 - \omega_B) = \theta^2(\omega_B + \dot{\theta})^2$

7.8 Combining equations and solving

Substitute $c = \left(\frac{I_G}{2mR^2} + 1\right)$ so you are left with

1. conservation of angular momentum

$c(\omega_B^0 - \omega_B) = \theta^2(\omega_B + \dot{\theta})$

1. work-energy

$c(\omega_B^0 - \omega_B^2)(\omega_B^0 + \omega_B^2) = \theta^2(\omega_B + \theta)^2$

dividing work-energy by conservation of angular momentum,

$\frac{c(\omega_B^0 - \omega_B^2)(\omega_B^0 + \omega_B^2)}{c(\omega_B^0 - \omega_B)} = \frac{\theta^2(\omega_B + \theta)^2}{\theta^2(\omega_B + \theta)}$

with the solution for $\dot{\theta}$

$\omega_B^0 + \omega_B = \omega_B + \dot{\theta} \to \omega_B^0 = \dot{\theta}.$

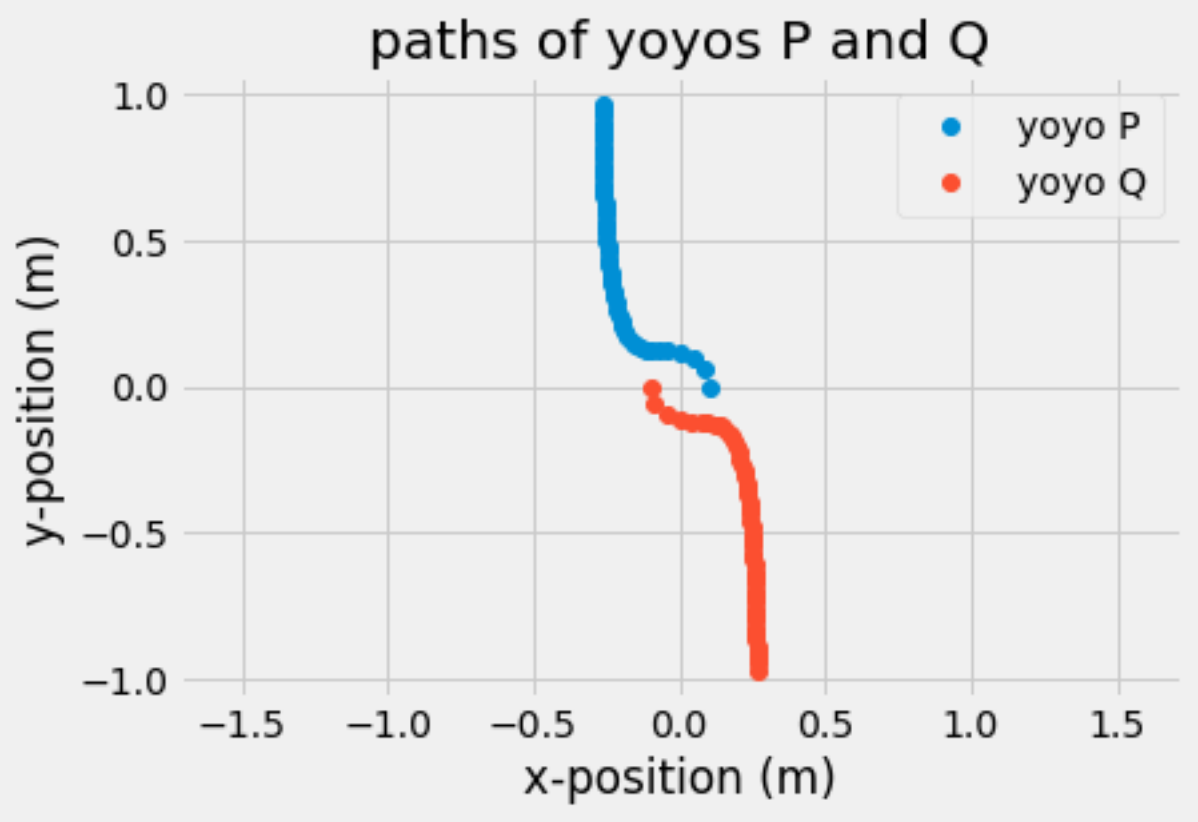
```
t = np.linspace(0,1)
theta = w0*t
```

This result, *combining conservation of angular momentum and work-energy*, tells you that the angular velocity of the yoyos will be be equal to the initial angular velocity of the payload. The angle θ will continue increase as $\omega_B^0 t$ until released. Plug this result into the orginal conservation of angular momentum equation to solve for ω_B

$c(\omega_B^0 - \omega_B) = (\omega_B^0 t)^2(\omega_B + \omega_B^0)$

$\omega_B(t) = \frac{c-(\omega_B^0 t)^2}{c+(\omega_B^0 t)^2}\omega_B^0$.

```
c = np.sqrt(I/2/m/R**2 + 1)
wB = lambda t: (c-w0**2*t**2)/(c+w0**2*t**2)*w0
wC = wB(t) + w0
x = R*np.cos(wC*t) - R*w0*t*np.sin(wC*t)
y = R*np.sin(wC*t) + R*w0*t*np.cos(wC*t)
plt.plot(x,y, 'o', label = 'yoyo P')
plt.plot(-x,-y, 'o', label = 'yoyo Q')
plt.axis('equal')
plt.title('paths of yoyos P and Q')
plt.xlabel('x-position (m)')
plt.ylabel('y-position (m)')
plt.legend();
```



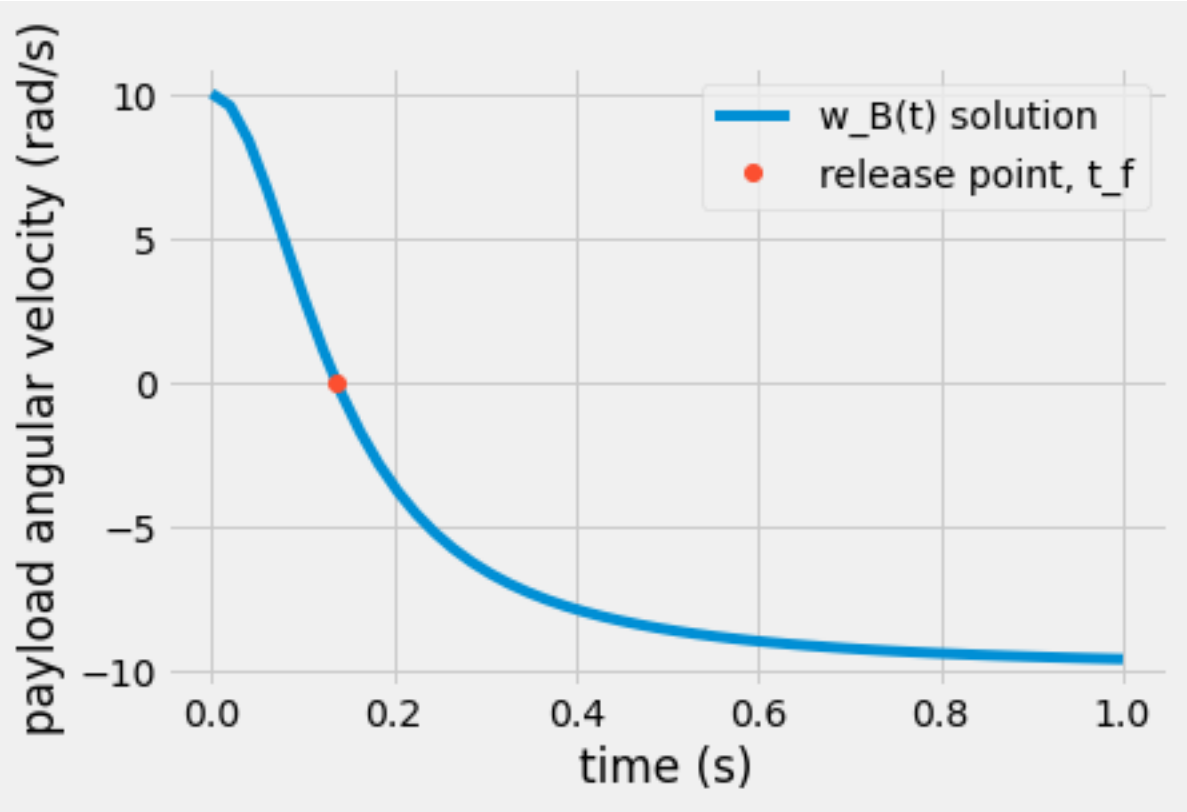
The added benefit of using cords to release the yoyos is that the payload angular velocity can be reduced to 0 rad/s at time, t_f.

$\omega_B(t_f) = 0 = c - (\omega_B^0 t)^2 \rightarrow t_f = \frac{\sqrt{c}}{\omega_B^0} = \frac{1}{\omega_B^0}\sqrt{\frac{I}{2mR^2} + 1}$.

The final cord length, is unraveling distance, $l_F = R\theta = R\omega_B^0 t_f$

$l_F = \sqrt{\frac{I}{2m} + R^2}$

```
tf = np.sqrt(c)/w0
lf = R*np.sqrt(c)
plt.plot(t,wB(t),label = 'w_B(t) solution')
plt.plot(tf,wB(tf),'o', label = 'release point, t_f')
plt.legend();
plt.xlabel('time (s)')
plt.ylabel('payload angular velocity (rad/s)');
```



CHAPTER

EIGHT

MODULE 5 - DYNAMICS IN THREE DIMENSIONS

spinning top design

1. 3D Newton-Euler equations
    1. 6 DOFs - 6 equations of motion
    2. angular momentum in 3D
    3.