ME314 Homework 4 (Template)

*Please note that a **single** PDF file will be the only document that you turn in, which will include your answers to the problems with corresponding derivations and any code used to complete the problems. When including the code, please make sure you also include **code outputs**, and you don't need to include example code. Problems and deliverables that should be included with your submission are shown in **bold**.*

This Juputer Notebook file serves as a template for you to start homework, since we recommend to finish the homework using Jupyter Notebook. You can start with this notebook file with your local Jupyter environment, or upload it to Google Colab. You can include all the code and other deliverables in this notebook Jupyter Notebook supports $\LaTeX$ for math equations, and you can export the whole notebook as a PDF file. But this is not the only option, if you are more comfortable with other ways, feel free to do so, as long as you can submit the homework in a single PDF file.

In [1]:

```
##################################################################################################
# If you're using Google Colab, uncomment this section by selecting the whole section and press
# ctrl+'/' on your and keyboard. Run it before you start programming, this will enable the nice
# # LaTeX "display()" function for you. If you're using the local Jupyter environment, leave it alone
# ##################################################################################################

# import sympy as sym
# def custom_latex_printer(exp,**options):
#     from google.colab.output._publish import javascript
#     url = "https://cdnjs.cloudflare.com/ajax/libs/mathjax/3.1.1/latest.js?config=TeX-AMS_HTML"
#     javascript(url=url)
#     return sym.printing.latex(exp,**options)
# sym.init_printing(use_latex="mathjax",latex_printer=custom_latex_printer)
```

Problem 1 (10pts)

Show that if $L = \frac{1}{2}\dot{q}^\top \mathcal{I} \dot{q}$, with $q \in \mathbb{R}^n$ (a column vector) and $\mathcal{I}$ is a symmetric matrix ($\mathcal{I} = \mathcal{I}^\top \in \mathbb{R}^{n\times n}$), then $\frac{\partial L}{\partial \dot{q}} = \dot{q}^\top \mathcal{I}$. \

Hint 1: You can either solve it directly (which means write down the analytical solution of $\frac{\partial L}{\partial \dot{q}}$ and $\dot{q}^\top \mathcal{I}$, then show they are equal) or use directional derivative.

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written solution. You can also use $\LaTeX$. If you use SymPy, then you just need to include a copy of code and the code outputs, with notes that explain why the code outputs indicate the two expressions are equal.

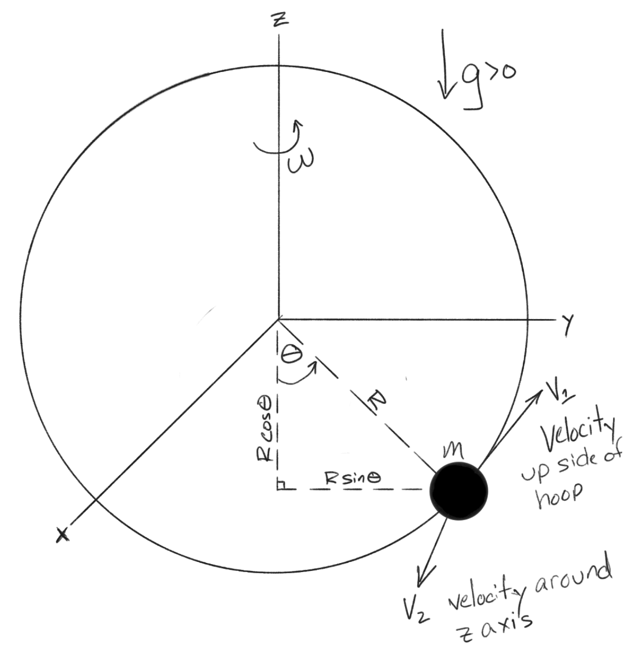
In [92]:

```
print("Hand written work included at bottom od pdf document")
print()
print()
```

```
Hand written work included at bottom od pdf document
```

In [2]:

```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/raw/master/dynhoop2.png' width=350' height='350'><
```



Problem 2 (20pts)

Take the bead on a hoop example shown in the image above, model it using a toeque input τ (about the vertical z axis) instead of a velocity input ω. You will need to add a configuration variable ψ that is the rotation about the z axis, so that the system configuration vector is $q = [\theta, \psi]$. Use Python's SymPy package to compute the equations of motion for this system in terms of θ, ψ.

Hint 1: Note that this should be a Lagrangian system with an external force.

Turn in: A copy of code used to symbolically solve for the equations of motion, also include the code outputs, which should be the equations of motion.

In [90]:

```python
import sympy as sym
from sympy import symbols, Matrix, Eq, Function, sin, cos, solve
from sympy.abc import t

th = Function(r'\theta')(t)
psi = Function(r'\psi')(t)
m = symbols('m')
r = symbols('R')
g = symbols('g')
tau = symbols(r'\tau')

q = Matrix([th, psi])
qdot = q.diff(t)
qddot = qdot.diff(t)

#find the x,y,z positions in 3D space
x = r * sin(th) * sin(psi)
y = r * sin(th) * cos(psi)
z = -r * cos(th)

xdot = x.diff(t)
ydot = y.diff(t)
zdot = z.diff(t)

KE = 0.5*m*(xdot**2 + ydot**2 + zdot**2)
PE = m*g*z
L = KE - PE
L = Matrix([L])
# display(L)

#take derivative of the Lagrangian wrt q
dLdq = L.jacobian(q).T
# print("dLdq: " )
# display(dLdq)

#symbolically compute dL/dq_dot terms of Euler-Lagrange equations
dLdq_dot = L.jacobian(qdot).T
# display(dLdq_dot)

#take time derivative of dL/dq_dot
ddt_dL_dqdot = dLdq_dot.diff(t)

#compute input forcing function
F = m * qdot[1]**2 * r
tau = F * r * sin(th)


#combine the previous terms to get lhs & rhs of Euler-Langrange equations
eL = Eq(dLdq - ddt_dL_dqdot, Matrix([0, tau])) #tau function applied only about z axis


# print('====================================================================================')
# print()
# print('Euler-Lagrangian equations with forcing function tau: ')
# display(eL)

#symbolically solve for the x_ddot and t_ddot variables contained in vector q_ddot
eL_solved = solve(eL, [qddot[0], qddot[1]]) #symbollically solve for each term in the E-L vector



#display the solution for x_ddot and t_ddot
print('================================================')
print()
print('Equations of motion: ')
print('(Cut off sections repeated at the bottom of pdf document) ')
for a in eL_solved:
    print()
    display(Eq(a, eL_solved[a]))

print()
print()
```

```
================================================

Equations of motion: 
(Cut off sections repeated at the bottom of pdf document)
```

$$\frac{d^2}{dt^2}\theta(t) = \frac{R\sin^2(\psi(t))\sin(\theta(t))\cos(\theta(t))\left(\frac{d}{dt}\psi(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(t))} + \frac{R\sin^2(\psi(t))\sin(\theta(t))\cos(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t)}$$

$$+ \frac{R\sin(\theta(t))\cos^2(\psi(t))\cos(\theta(t))\left(\frac{d}{dt}\psi(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(t))} + \frac{R\sin(\theta(t))\cos^2(\psi(t))\cos(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(}$$

$$- \frac{R\sin(\theta(t))\cos(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(t))} - \frac{g\sin(\theta(t))}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(}$$

d^2 $2.0\sin^2(\psi(t))\cos(\theta(t))\frac{d}{dt}\psi(t)\frac{d}{dt}\theta(t)$ $2.0\cos^2(\psi(t))\cos(\theta(t))\frac{d}{dt}\psi(t)\frac{d}{dt}\theta(t)$ $\left(\frac{d}{dt}\psi(t)\right)^2$

Problem 3 (30pts)

Consider a point mass in 3D space under the forces of gravity and a radial spring from the origin. The system's Lagrangian is:

$$L = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) - \frac{1}{2}k(x^2 + y^2 + z^2) - mgz$$

Consider the following rotation matrices, defining rotations about the z, y, and x axes respectively:

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_\psi = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}, \quad R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

and answer the following three questions:

1. Which, if any, of the transformations $q_\theta = R_\theta q$, $q_\psi = R_\psi q$, or $q_\phi = R_\phi q$ keeps the Lagrangian fixed? Is this invariance global or local?
2. Use small angle approximations to linearize your transformation(s) from the first question. The resulting new transformation should have the form $q_\epsilon = q + \epsilon G(q)$. Compute the difference in the Lagrangian $L(q_\epsilon, \dot{q}_\epsilon) - L(q, \dot{q})$ through this transformation.
3. Apply Noether's theorem to determine a conserved quantity. Physically what does this quantity represent? Is there any physical rationale behind it's conservation?

You can solve this problem by hand or use Python's SymPy to do the symbolic computation for you.

Hint 1: For question (1), try to imagine how this system looks: even though x, y, and z axes seem to have same influence on the system, based on the Lagrangian, rotation around some axes will different influence to the Lagrangian from others.

Hint 2: Global invariance here means for any magnitude of rotation the Lagrangian will remain fixed.

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written solution. You can also use $\LaTeX$. If you use SymPy, then you just need to include a copy of code and the code outputs, with notes that explain why the code outputs can answer the questions.

In [61]:

```python
#Solution to Problem 3.1:
import sympy as sym
from sympy import symbols, Matrix, Eq, Function, sin, cos, solve

th = symbols(r'\theta')
psi = symbols(r'\psi')
phi = symbols(r'\phi')
m = symbols('m')
r = symbols('R')
k = symbols('k')
g = symbols('g')

#need these cartesian functions as "dummy variables" to be later used in Lagrangian
x = Function(r'x')(t)
y = Function(r'y')(t)
z = Function(r'z')(t)

q = Matrix([x,y,z])
q_dot = q.diff(t)
q_ddot = q_dot.diff(t)


#define rotational matrices
r_th = Matrix([[cos(th), -sin(th), 0], [sin(th), cos(th), 0], [0, 0, 1]])
r_psi = Matrix([[cos(psi), 0, sin(psi)], [0,1,0], [-sin(psi), 0, cos(psi)]])
r_phi = Matrix([[1,0,0], [0, cos(phi), -sin(phi)], [0, sin(phi), cos(phi)]])

#compute the transformations:
q_th = r_th*q
q_psi = r_psi*q
q_phi = r_phi*q
#take time derivative to be used in KE calculations
q_th_dot = q_th.diff(t)
q_psi_dot = q_psi.diff(t)
q_phi_dot = q_phi.diff(t)



#now compute Lagrangian for each transformation separately. Each of the three results will be compared
#against the original given Lagrangian to check for global and local invariance

#extract x,y,z components from q_th transformation matrix
q_th_x = q_th[0]
q_th_y = q_th[1]
q_th_z = q_th[2]
q_th_dot_x = q_th_dot[0]
q_th_dot_y = q_th_dot[1]
q_th_dot_z = q_th_dot[2]


#extract x,y,z components from q_psi transformation matrix
q_psi_x = q_psi[0]
q_psi_y = q_psi[1]
q_psi_z = q_psi[2]
q_psi_dot_x = q_psi_dot[0]
q_psi_dot_y = q_psi_dot[1]
q_psi_dot_z = q_psi_dot[2]


#extract x,y,z components from q_phi transformation matrix
q_phi_x = q_phi[0]
q_phi_y = q_phi[1]
q_phi_z = q_phi[2]
q_phi_dot_x = q_phi_dot[0]
q_phi_dot_y = q_phi_dot[1]
q_phi_dot_z = q_phi_dot[2]


#calculate the respective Lagrangians
#Lagrangian from q_theta transform
ke_th = m * (q_th_dot_x**2 + q_th_dot_y**2 + q_th_dot_z**2)/2
pe_th = (k * (q_th_x**2 + q_th_y**2 + q_th_z**2)/2) + (m * g * q_th_z)
L_th = ke_th - pe_th

#Lagrangian from q_psi transform
ke_psi = m * (q_psi_dot_x**2 + q_psi_dot_y**2 + q_psi_dot_z**2)/2
pe_psi = (k * (q_psi_x**2 + q_psi_y**2 + q_psi_z**2)/2) + (m * g * q_psi_z)
L_psi = ke_psi - pe_psi

#Lagrangian from q_phi transform
ke_phi = m * (q_phi_dot_x**2 + q_phi_dot_y**2 + q_phi_dot_z**2)/2
pe_phi = (k * (q_phi_x**2 + q_phi_y**2 + q_phi_z**2)/2) + (m * g * q_phi_z)
L_phi = ke_phi - pe_phi

print()
print("Solution to Problem 3.1: ")


print('==================================================')
print()
print("Lagrangian of theta transform: ")
display(simplify(L_th))

print('==================================================')
print()
print("Lagrangian of psi transform: ")
display(simplify(L_psi))
```

```
print('=================================================')
print()
print("Lagrangian of phi transform: ")
display(simplify(L_phi))

print('=================================================')
print('Conclusion:')
print("Only a theta transformation about the z-axis keeps the Lagrangian fixed, as shown by the theta transform.")
print()
print()
```

```
Solution to Problem 3.1:
=================================================

Lagrangian of theta transform:
```

$$-gmz(t) - \frac{k\left(x^2(t)+y^2(t)+z^2(t)\right)}{2} + \frac{m\left(\left(\frac{d}{dt}x(t)\right)^2+\left(\frac{d}{dt}y(t)\right)^2+\left(\frac{d}{dt}z(t)\right)^2\right)}{2}$$

```
=================================================

Lagrangian of psi transform:
```

$$gm\left(x(t)\sin(\psi) - z(t)\cos(\psi)\right) - \frac{k\left(x^2(t)+y^2(t)+z^2(t)\right)}{2} + \frac{m\left(\left(\frac{d}{dt}x(t)\right)^2+\left(\frac{d}{dt}y(t)\right)^2+\left(\frac{d}{dt}z(t)\right)^2\right)}{2}$$

```
=================================================

Lagrangian of phi transform:
```

$$-gm\left(y(t)\sin(\phi) + z(t)\cos(\phi)\right) - \frac{k\left(x^2(t)+y^2(t)+z^2(t)\right)}{2} + \frac{m\left(\left(\frac{d}{dt}x(t)\right)^2+\left(\frac{d}{dt}y(t)\right)^2+\left(\frac{d}{dt}z(t)\right)^2\right)}{2}$$

```
=================================================
Conclusion:
Only a theta transformation about the z-axis keeps the Lagrangian fixed, as shown by the theta transform.
```

In [60]:

```
#Solution to Problem 3.2:
#In part 3.1 we found only the theta transform keeps Lagrangian fixed so only theta transform
#needs to be linearized.

#using small-angle approximation, linearize r_theta from part 3.1:
#sin(theta) = theta, cos(theta) = 1
r_th_lin = Matrix([[1, -th, 0], [th, 1, 0], [0, 0, 1]])

#extract x,y,z components from q_th transformation matrix
#copy from part 3.1:
q_th = r_th_lin * q
q_th_x = q_th[0]
q_th_y = q_th[1]
q_th_z = q_th[2]
q_th_dot_x = q_th_dot[0]
q_th_dot_y = q_th_dot[1]
q_th_dot_z = q_th_dot[2]


#recalculate Lagrangian after linearizing theta transform:
ke_th = m * (q_th_dot_x**2 + q_th_dot_y**2 + q_th_dot_z**2)/2
pe_th = (k * (q_th_x**2 + q_th_y**2 + q_th_z**2)/2) + (m * g * q_th_z)
L_th_lin = ke_th - pe_th

L_diff = L_th_lin - L_th
print("Difference in Lagrangians after linearizing the theta transform: ")
display(simplify(L_diff))
print()
print()
```

```
Difference in Lagrangians after linearizing the theta transform:
```

$$-\frac{\theta^2 k\left(x^2(t)+y^2(t)\right)}{2}$$

In [88]:

```
#Solution to Problem 3.2:
#conserved quantity using Noether's Theorem: conserved_qty = dL_dq_dot * G_q
x = Function(r'x')(t)
y = Function(r'y')(t)
z = Function(r'z')(t)

q = Matrix([x,y,z])
q_dot = q.diff(t)
q_ddot = q_dot.diff(t)

#derive original Lagrangian:
KE = m * (q_dot[0]**2 + q_dot[1]**2 + q_dot[2]**2) / 2
PE = (k * (q[0]**2 + q[1]**2 + q[2]**2)/2) + (m * g * q[2])

L = Matrix([KE - PE])

#conserved qty p = dL_dq_dot * G_q
dL_dq_dot = L.jacobian(q_dot)
G_q = Matrix([-q[1],q[0],0])
p = dL_dq_dot * G_q

print("conserved quantity, generalized momentum: ")
display(p)

print("As described in Lecture 10 & 11 pdf notes, the conserved quantity is the angular momentum about the origin")
print("As shown previously, the Lagrangian is locally invariant in the theta transform, showing us the ")
print("angular momentum in the theta direction is a conserved quantity")

print()
print()
```

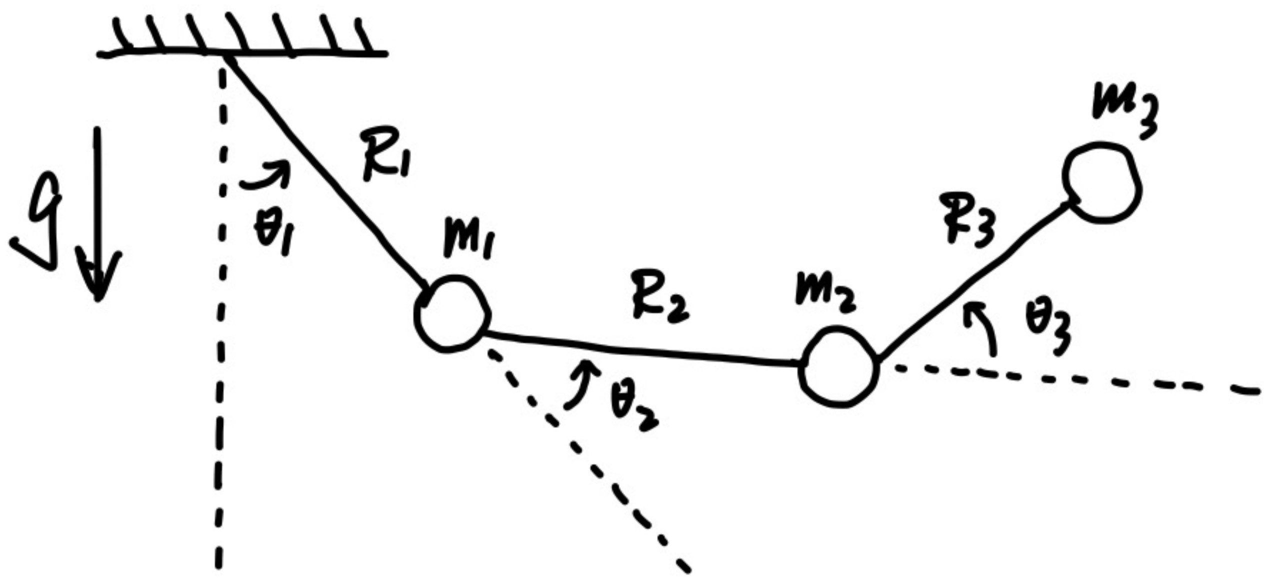
```
conserved quantity, generalized momentum:
```

$$\left[m x(t) \frac{d}{dt} y(t) - m y(t) \frac{d}{dt} x(t) \right]$$

```
As described in Lecture 10 & 11 pdf notes, the conserved quantity is the angular momentum about the origin
As shown previously, the Lagrangian is locally invariant in the theta transform, showing us the
angular momentum in the theta direction is a conserved quantity
```

In [64]:

```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/raw/master/tripend.png' width=700' height='350'></
```



Problem 4 (30pts)

Given the unconstrained triple-pendulum system shown in the image above, simulate the system in terms of $q = [\theta_1, \theta_2, \theta_3]$ with $R_1 = R_2 = R_3 = 1$ and $m_1 = m_2 = m_3 = 2$ for $t \in [0, 5], dt = 0.01$. Initial condition is $\theta_1 = \theta_2 = \theta_3 = \frac{\pi}{6}$. You should use the Runge–Kutta integration function provided in previous homework for simulation. Plot the simulated trajectory for $\theta_1, \theta_2, \theta_3$ versus time.

Hint 1: Given the complexity of the system, solving the equations of motion symbolically could take around 10 to 20 minutes. It is highly recommended to separate your code into several cells, thus you don't have to wait a long time to run each piece of your code.

Hint 2: You don't need to turn in the equations of motions in this problem---they are so long that it's not recommended to print them out.

Turn in: A copy of code used to simulate the system, you don't need to turn in equations of motion, but you need to include the plot of the simulated trajectories.

In [65]:
```
th1 = Function(r'theta_1')(t)
th2 = Function(r'theta_2')(t)
th3 = Function(r'theta_3')(t)
m1 = symbols('m_1')
m2 = symbols('m_2')
m3 = symbols('m_3')
r1 = symbols('R_1')
r2 = symbols('R_2')
r3 = symbols('R_3')
g = symbols('g')
l = symbols('L')

x1 = r1*sin(th1)
x2 = x1 + r2*sin(th1 + th2)
x3 = x2 + r3*sin(th1 + th2 + th3)

y1 = -r1*cos(th1)
y2 = y1 - r2*cos(th1 + th2)
y3 = y2 - r3*cos(th1 + th2 + th3)

q = Matrix([th1, th2, th3])
qdot = q.diff(t)
qddot = qdot.diff(t)

x1dot = x1.diff(t)
x2dot = x2.diff(t)
x3dot = x3.diff(t)

y1dot = y1.diff(t)
y2dot = y2.diff(t)
y3dot = y3.diff(t)

ke = 0.5*m1*(x1dot**2 + y1dot**2) + 0.5*m2*(x2dot**2 + y2dot**2) + 0.5*m3*(x3dot**2 + y3dot**2)
pe = m1*g*y1 + m2*g*y2 + m3*g*y3

ke = ke.subs({m1:2, m2:2, m3:2, r1:1, r2:1, r3:1, g:9.8})
pe = pe.subs({m1:2, m2:2, m3:2, r1:1, r2:1, r3:1, g:9.8})


L = ke - pe
# display(Eq(l, L))
L_mat = Matrix([L])
```

In [66]:
```
# display(q, qdot, qddot)
# display(L_mat)
```

In [67]:
```
from math import pi
from numpy import arange
import numpy as np
import matplotlib.pyplot as plt

dLdq = L_mat.jacobian(q).T
dLdqdot = L_mat.jacobian(qdot).T
d_dLdqdot_dt = dLdqdot.diff(t)

el = Eq(dLdq - d_dLdqdot_dt, Matrix([0, 0, 0]))

# print('==================================================')
# print()
# print("dLdq")
# display(dLdq)

# print('==================================================')
# print()
# print("dLdqdot")
# display(dLdqdot)

# print('==================================================')
# print()
# print("Euler Lagrange equations for triple pendulum system")
# display(el)
```

In [68]:
```
# solve for the equations of motion:
el_soln = solve(el, qddot)
```

In [14]:

```
# display(el_soln)
# for i in qddot:
#     print('=================================================')
#     print()
#     display(Eq(i, el_soln[i]))
# print('=================================================')
# print()


# th1ddot_soln = el_soln[qddot[0]].subs({m1:2, m2:2, m3:2, r1:1, r2:1, r3:1, g:9.8})
# th2ddot_soln = el_soln[qddot[1]].subs({m1:2, m2:2, m3:2, r1:1, r2:1, r3:1, g:9.8})
# th3ddot_soln = el_soln[qddot[2]].subs({m1:2, m2:2, m3:2, r1:1, r2:1, r3:1, g:9.8})

th1ddot_func = sym.lambdify([th1, th2, th3, qdot[0], qdot[1], qdot[2]], el_soln[qddot[0]], modules = sym)
th2ddot_func = sym.lambdify([th1, th2, th3, qdot[0], qdot[1], qdot[2]], el_soln[qddot[1]], modules = sym)
th3ddot_func = sym.lambdify([th1, th2, th3, qdot[0], qdot[1], qdot[2]], el_soln[qddot[2]], modules = sym)
```

In [22]:

```python
#carry over integrate, simulate, and dyn functions from previous homework
import sympy as sym
from sympy import symbols, simplify, lambdify
import matplotlib.pyplot as plt
import numpy as np
from math import pi

def integrate(f, xt, dt):
    """
    This function takes in an initial condition x(t) and a timestep dt,
    as well as a dynamical system f(x) that outputs a vector of the
    same dimension as x(t). It outputs a vector x(t+dt) at the future
    time step.

    Parameters
    ============
    dyn: Python function
        derivate of the system at a given step x(t),
        it can considered as \dot{x}(t) = func(x(t))
    xt: NumPy array
        current step x(t)
    dt:
        step size for integration

    Return
    ============
    new_xt:
        value of x(t+dt) integrated from x(t)
    """
    k1 = dt * f(xt)
    k2 = dt * f(xt+k1/2.)
    k3 = dt * f(xt+k2/2.)
    k4 = dt * f(xt+k3)
    new_xt = xt + (1/6.) * (k1+2.0*k2+2.0*k3+k4)
    return new_xt

def simulate(f, x0, tspan, dt, integrate):
    """
    This function takes in an initial condition x0, a timestep dt,
    a time span tspan consisting of a list [min_time, max_time],
    as well as a dynamical system f(x) that outputs a vector of the
    same dimension as x0. It outputs a full trajectory simulated
    over the time span of dimensions (xvec_size, time_vec_size).

    Parameters
    ============
    f: Python function
        derivate of the system at a given step x(t),
        it can considered as \dot{x}(t) = func(x(t))
    x0: NumPy array
        initial conditions
    tspan: Python list
        tspan = [min_time, max_time], it defines the start and end
        time of simulation
    dt:
        time step for numerical integration
    integrate: Python function
        numerical integration method used in this simulation

    Return
    ============
    x_traj:
        simulated trajectory of x(t) from t=0 to tf
    """
    N = int((max(tspan)-min(tspan))/dt)
    x = np.copy(x0)
    tvec = np.linspace(min(tspan),max(tspan),N)
    xtraj = np.zeros((len(x0),N))
    for i in range(N):
        xtraj[:,i]=integrate(f,x,dt)
        x = np.copy(xtraj[:,i])
    return xtraj

###############################################
def dyn(s):
    """
    System dynamics function (extended)

    Parameters
    ============
    s: NumPy array
        s = [theta1, theta2, theta3, theta1dot, theta2dot, theta3dot]

    Return
    ============
    sdot: NumPy array
        time derivative of input state vector,
        sdot = [theta1dot, theta2dot, theta3dot, theta1_ddot, theta2_ddot, theta3_ddot]
    """
    return np.array([s[3], s[4], s[5], th1ddot_func(s[0], s[1], s[2], s[3], s[4], s[5]), th2ddot_func(s[0], s[1], s[2], s[3], s[
    #Should be outputting theta1dot, theta2dot, theta1_ddot, theta2_ddot

# define initial state
s0 = np.array([pi/6, pi/6, pi/6, 0, 0, 0]) # inital values for theta1, theta2, theta3, theta1dot, theta2dot, theta3
# simulate from t=0 to 5, since dt=0.01, the returned trajectory
# will have 5/0.01= 500 time steps, each time step contains extended
# system state vector [theta1, theta2, theta3, theta1dot, theta2dot, theta3dot]
```

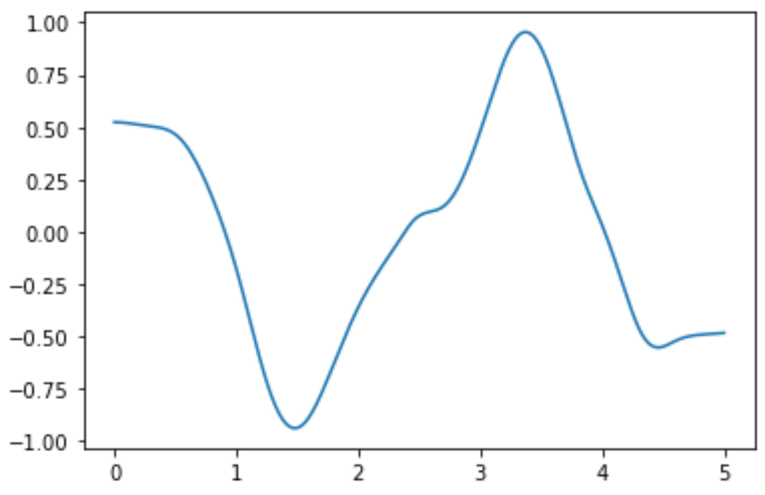
```
print()
print('shape of traj: ', traj.shape)
```

```
shape of traj:  (6, 500)
```

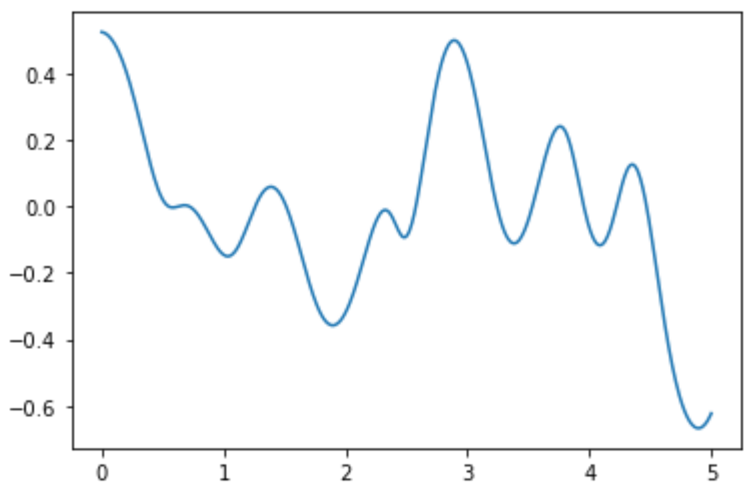
In [20]:

```
#plot the theta trajectories
num_pts = int(5/.01)
axis = np.linspace(0,5, num_pts )

print('theta1 trajectory')
plt.plot(axis, traj[0])
plt.show()
print('theta2 trajectory')
plt.plot(axis, traj[1])
plt.show()
print('theta3 trajectory')
plt.plot(axis, traj[2])
plt.show()
```

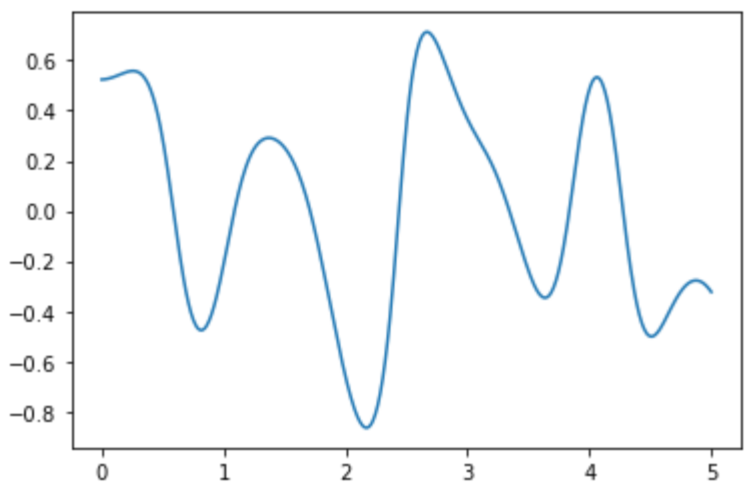
```
theta1 trajectory
```



```
theta2 trajectory
```



```
theta3 trajectory
```



Problem 5 (10pts)

Based on the animation function provided in previous homeworks for double-pendulum system, create a new animation function called "animate_triple_pend", so it can animate the simulated trajectory for triple pendulum system you got from problem 4.

Hint 1: The previous animation function for double pendulum is provided below. You don't need to change the "browser configuration" and "simulation layout" parts.

Hint 2: The first part you should pay attention to is the part converting the angle-based trajectory into x-y coordinates. In the double-pendulum animation function, only the first two pendulums will be converted, you will need to add some code to include the third pendulum.

Hint 3: After that, you need to modify the "data" and "frames" variables in the function to include the x-y coordinated trajectory of the third pendulum. One suggestion here is to look at the color defined for each variable, which could give you some clue about what each variables means in the animation. Do some modifications and call the function with some simple dummy data to see if the changes are effective.

Hint 4: It's not hard and it's not supposed to include many modifications, eventually you will just need to duplicate some parts of the code and change some configuration like color or trajectory array. If you want to see official documents (which, from my perspective, is not necessary), you can check: "https://plotly.com/python/animations/ ".

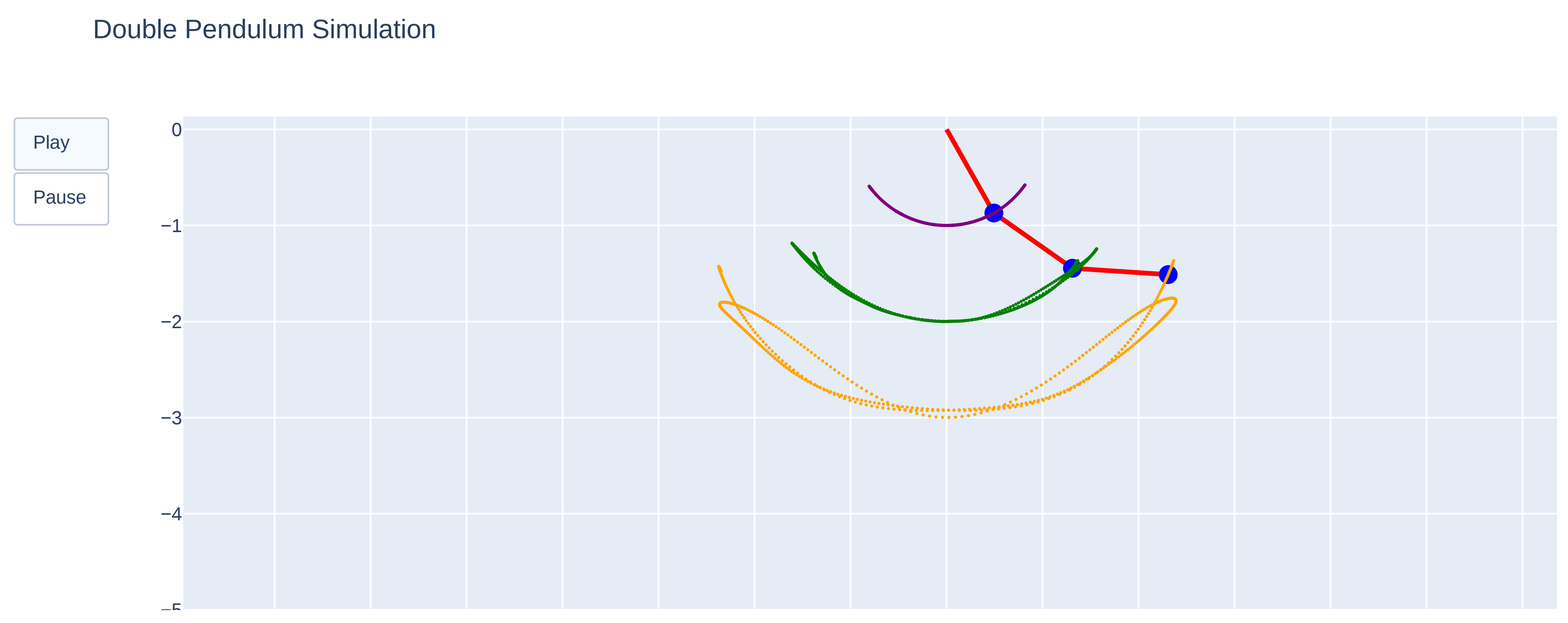
Hint 5: After you finished the function, generate some dummy trajectory (like the example provided for double-pendulum animation function) to make sure *everything works well.*

Turn in: A copy of code for your animation function. Also, upload a video of the simulated triple pendulum system you got from Problem 4. The video can be uploaded separately through Canvas, and it should be in ".mp4" format. You can use screen capture or record the screen directly with your phone.

In [25]:

```python
def animate_triple_pend(theta_array,L1=1,L2=1,L3=1,T=5):
    """
    Function to generate web-based animation of double-pendulum system

    Parameters:
    ================================================
    theta_array:
        trajectory of theta1 and theta2, should be a NumPy array with
        shape of (2,N)
    L1:
        length of the first pendulum
    L2:
        length of the second pendulum
    T:
        length/seconds of animation duration

    Returns: None
    """

    ################################
    # Imports required for animation. (leave this part)
    from plotly.offline import init_notebook_mode, iplot
    from IPython.display import display, HTML
    import plotly.graph_objects as go

    #######################
    # Browser configuration. (leave this part)
    def configure_plotly_browser_state():
        import IPython
        display(IPython.core.display.HTML('''
            <script src="/static/components/requirejs/require.js"></script>
            <script>
              requirejs.config({
                paths: {
                  base: '/static/base',
                  plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
                },
              });
            </script>
            '''))
    configure_plotly_browser_state()
    init_notebook_mode(connected=False)

    ###############################################
    # Getting data from pendulum angle trajectories. (add some code to include the third pendulum)
    xx1=L1*np.sin(theta_array[0])
    yy1=-L1*np.cos(theta_array[0])
    xx2=xx1+L2*np.sin(theta_array[0]+theta_array[1])
    yy2=yy1-L2*np.cos(theta_array[0]+theta_array[1])
    N = len(theta_array[0]) # Need this for specifying length of simulation
    xx3=xx2+L3*np.sin(theta_array[0]+theta_array[1]+theta_array[2])
    yy3=yy2-L3*np.cos(theta_array[0]+theta_array[1]+theta_array[2])

    ####################################
    # Using these to specify axis limits. (this needs to be adjusted too)
    xm=np.min(xx3)-0.5
    xM=np.max(xx3)+0.5
    ym=np.min(yy3)-2.5
    yM=np.max(yy3)+1.5

    ###########################
    # Defining data dictionary. (add some code to include the third pendulum)
    # Trajectories are here.
    data=[dict(x=xx1, y=yy1,
               mode='lines', name='Arm',
               line=dict(width=2, color='blue')
              ),
          dict(x=xx1, y=yy1,
               mode='lines', name='Mass 1',
               line=dict(width=2, color='purple')
              ),
          dict(x=xx2, y=yy2,
               mode='lines', name='Mass 2',
               line=dict(width=2, color='green')
              ),
          dict(x=xx3, y=yy3,
               mode='lines', name='Mass 3',
               line=dict(width=2, color='yellow')
              ),
          dict(x=xx1, y=yy1,
               mode='markers', name='Pendulum 1 Traj',
               marker=dict(color="purple", size=2)
              ),
          dict(x=xx2, y=yy2,
               mode='markers', name='Pendulum 2 Traj',
               marker=dict(color="green", size=2)
              ),
          dict(x=xx3, y=yy3,
               mode='markers', name='Pendulum 3 Traj',
               marker=dict(color="orange", size=2)
              ),
        ]

    ################################
    # Preparing simulation layout. (leave this part)
    # Title and axis ranges are here.
    layout=dict(xaxis=dict(range=[xm, xM], autorange=False, zeroline=False,dtick=1),
                title='Double Pendulum Simulation',
                hovermode='closest',
                updatemenus= [{'type': 'buttons',
                               'buttons': [{'label': 'Play','method': 'animate',
                                            'args': [None, {'frame': {'duration': T, 'redraw': False}}]},
                                           {'args': [[None], {'frame': {'duration': T, 'redraw': False}, 'mode': 'immediate',
                                            'transition': {'duration': 0}}],'label': 'Pause','method': 'animate'}
                                          ]
                              }]
               )

    #########################################
    # Defining the frames of the simulation. (add some code to include the third pendulum)
    # This is what draws the lines from
    # joint to joint of the pendulum.
    frames=[dict(data=[dict(x=[0,xx1[k],xx2[k],xx3[k]],
                            y=[0,yy1[k],yy2[k],yy3[k]],
                            mode='lines',
                            line=dict(color='red', width=3)
                            ),
                       go.Scatter(
                            x=[xx1[k]],
                            y=[yy1[k]],
                            mode="markers",
                            marker=dict(color="blue", size=12)),
                       go.Scatter(
                            x=[xx2[k]],
                            y=[yy2[k]],
                            mode="markers",
                            marker=dict(color="blue", size=12)),
                       go.Scatter(
                            x=[xx3[k]],
                            y=[yy3[k]],
                            mode="markers",
                            marker=dict(color="blue", size=12)),
                      ]) for k in range(N)]

    #######################################
    # Putting it all together and plotting.
    figure1=dict(data=data, layout=layout, frames=frames)
    iplot(figure1)

##################################################
# Example of animation

# provide a trajectory of double-pendulum
# (note that this array below is not an actual simulation,
# but lets you see this animation code work)
import numpy as np
sim_traj = np.array([np.linspace(-1, 1, 100), np.linspace(-1, 1, 100)])
print('shape of trajectory: ', sim_traj.shape)

# second, animate!
animate_triple_pend(traj,L1=1,L2=1,L3=1,T=5)
```

```
shape of trajectory:  (2, 100)
```



```
In [ ]:
```

```python
print()
print()
```

```
==================================================

Equations of motion:
```

$$\frac{d^2}{dt^2}\theta(t) = \frac{R\sin^2(\psi(t))\sin(\theta(t))\cos(\theta(t))\left(\frac{d}{dt}\psi(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(t))} + \frac{R\sin^2(\psi(t))\sin(\theta(t))\cos(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(t))}$$
$$+ \frac{R\sin(\theta(t))\cos^2(\psi(t))\cos(\theta(t))\left(\frac{d}{dt}\psi(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(t))} + \frac{R\sin(\theta(t))\cos^2(\psi(t))\cos(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(t))}$$
$$- \frac{R\sin(\theta(t))\cos(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(t))} - \frac{g\sin(\theta(t))}{R\sin^2(\psi(t))\cos^2(\theta(t)) + R\sin^2(\theta(t)) + R\cos^2(\psi(t))\cos^2(\theta(t))}$$

$$\frac{d^2}{dt^2}\psi(t) = -\frac{2.0\sin^2(\psi(t))\cos(\theta(t))\frac{d}{dt}\psi(t)\frac{d}{dt}\theta(t)}{\sin^2(\psi(t))\sin(\theta(t)) + \sin(\theta(t))\cos^2(\psi(t))} - \frac{2.0\cos^2(\psi(t))\cos(\theta(t))\frac{d}{dt}\psi(t)\frac{d}{dt}\theta(t)}{\sin^2(\psi(t))\sin(\theta(t)) + \sin(\theta(t))\cos^2(\psi(t))} - \frac{\left(\frac{d}{dt}\psi(t)\right)^2}{\sin^2(\psi(t))\sin(\theta(t)) + \sin(\theta(t))\cos^2(\psi(t))}$$

Problem 3 (30pts)

Consider a point mass in 3D space under the forces of gravity and a radial spring from the origin. The system's Lagrangian is:

$$L = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2) - \frac{1}{2}k(x^2 + y^2 + z^2) - mgz$$

Consider the following rotation matrices, defining rotations about the z, y, and x axes respectively:

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_\psi = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}, \quad R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

and answer the following three questions:

1. Which, if any, of the transformations $q_\theta = R_\theta q$, $q_\psi = R_\psi q$, or $q_\phi = R_\phi q$ keeps the Lagrangian fixed? Is this invariance global or local?
2. Use small angle approximations to linearize your transformation(s) from the first question. The resulting new transformation should have the form $q_\epsilon = q + \epsilon G(q)$. Compute the difference in the Lagrangian $L(q_\epsilon, \dot{q}_\epsilon) - L(q, \dot{q})$ through this transformation.
3. Apply Noether's theorem to determine a conserved quantity. Physically what does this quantity represent? Is there any physical rationale behind it's conservation?

You can solve this problem by hand or use Python's SymPy to do the symbolic computation for you.

Hint 1: For question (1), try to imagine how this system looks: even though x, y, and z axes seem to have same influence on the system, based on the Lagrangian, rotation around some axes will different influence to the Lagrangian from others.

Hint 2: Global invariance here means for any magnitude of rotation the Lagrangian will remain fixed.

Turn in: A scanned (or photograph from your phone or webcam) copy of your hand written solution. You can also use $\LaTeX$. If you use SymPy, then you just need to include a copy of code and the code outputs, with notes that explain why the code outputs can answer the questions.

1) $L = \frac{1}{2}\dot{q}^T I \dot{q}$, $q \in \mathbb{R}^n$, I is symmetric

Show $\frac{\partial L}{\partial \dot{q}} = \dot{q}^T I$

Directional derivative:

$$DL \cdot v = \frac{d}{d\epsilon} L(\dot{q} + \epsilon v)\Big|_{\epsilon=0}$$

$$= \frac{d}{d\epsilon}\left[\frac{1}{2}(\dot{q}+\epsilon v)^T I (\dot{q}+\epsilon v)\right]\Big|_{\epsilon=0}$$

$$= (\dot{q} + \cancelto{0}{\epsilon v})^T I \cdot v \Big|_{\epsilon=0}$$

$$= \dot{q}^T I \cdot v$$

$$\hookrightarrow \boxed{\frac{\partial L}{\partial \dot{q}} = \dot{q}^T I}$$