ME314 Homework 4

Sean Morton

collaborators: Noah Yi

Submission instructions

Deliverables that should be included with your submission are shown in **bold** at the end of each problem statement and the corresponding supplemental material. **Your homework will be graded IFF you submit a single PDF, .mp4 videos of animations when requested and a link to a Google colab file that meet all the requirements outlined below.**

- List the names of students you've collaborated with on this homework assignment.
- Include all of your code (and handwritten solutions when applicable) used to complete the problems.
- Highlight your answers (i.e. **bold** and outline the answers) for handwritten or markdown questions and include simplified code outputs (e.g. .simplify()) for python questions.
- Enable Google Colab permission for viewing
- Click Share in the upper right corner
- Under "Get Link" click "Share with..." or "Change"
- Then make sure it says "Anyone with Link" and "Editor" under the dropdown menu
- Make sure all cells are run before submitting (i.e. check the permission by running your code in a private mode)
- Please don't make changes to your file after submitting, so we can grade it!
- Submit a link to your Google Colab file that has been run (before the submission deadline) and don't edit it afterwards!

NOTE: This Juputer Notebook file serves as a template for you to start homework. Make sure you first copy this template to your own Google driver (click "File" -> "Save a copy in Drive"), and then start to edit it.

In [1]:
```
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt
```

In [2]:
```
##############################################################################################
# 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
##############################################################################################
# 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)
```

Helper Functions

In [3]:
```
def compute_EL(lagrangian, q):
    '''
    Helper function for computing the Euler-Lagrange equations for a given system,
    so I don't have to keep writing it out over and over again.

    Inputs:
    - lagrangian: our Lagrangian function in symbolic (Sympy) form
    - q: our state vector [x1, x2, ...], in symbolic (Sympy) form

    Outputs:
    - eqn: the Euler-Lagrange equations in Sympy form
    '''

    # wrap system states into one vector (in SymPy would be Matrix)
    #q = sym.Matrix([x1, x2])
    qd = q.diff(t)
    qdd = qd.diff(t)

    # compute derivative wrt a vector, method 1
    # wrap the expression into a SymPy Matrix
    L_mat = sym.Matrix([lagrangian])
    dL_dq = L_mat.jacobian(q)
    dL_dqdot = L_mat.jacobian(qd)

    #set up the Euler-Lagrange equations
    LHS = dL_dq - dL_dqdot.diff(t)
    RHS = sym.Matrix([0,0]).T
    eqn = sym.Eq(LHS.T, RHS.T)

    return eqn

def solve_EL(eqn, var):
    '''
    Helper function to solve and display the solution for the Euler-Lagrange
```

```python
    equations.

    Inputs:
    - eqn: Euler-Lagrange equation (type: Sympy Equation())
    - var: state vector (type: Sympy Matrix). typically a form of q-doubledot
        but may have different terms

    Outputs:
    - Prints symbolic solutions
    - Returns symbolic solutions in a dictionary
    '''

    soln = sym.solve(eqn, var, dict = True)
    eqns_solved = []


    for i, sol in enumerate(soln):
        for x in list(sol.keys()):
            eqn_solved = sym.Eq(x, sol[x])
            eqns_solved.append(eqn_solved)

    return eqns_solved

def solve_constrained_EL(lamb, phi, q, lhs):
    """Now uses just the LHS of the constrained E-L equations,
    rather than the full equation form"""

    qd = q.diff(t)
    qdd = qd.diff(t)

    phidd = phi.diff(t).diff(t)
    lamb_grad = sym.Matrix([lamb * phi.diff(a) for a in q])
    q_mod = qdd.row_insert(2, sym.Matrix([lamb]))

    #format equations so they're all in one matrix
    expr_matrix = lhs - lamb_grad
    phidd_matrix = sym.Matrix([phidd])
    expr_matrix = expr_matrix.row_insert(2,phidd_matrix)

    print("Equations to be solved (LHS - lambda * grad(phi) = 0):")
    RHS = sym.zeros(len(expr_matrix), 1)
    disp_eq = sym.Eq(expr_matrix, RHS)
    display(disp_eq)
    print("Variables to solve for:")
    display(q_mod)

    #solve E-L equations
    eqns_solved = solve_EL(expr_matrix, q_mod)
    return eqns_solved
```

In [4]:
```python
def rk4(dxdt, x, t, dt):
    '''
    Applies the Runge-Kutta method, 4th order, to a sample function,
    for a given state q0, for a given step size. Currently only
    configured for a 2-variable dependent system (x,y).
    ==========
    dxdt: a Sympy function that specifies the derivative of the system of interest
    t: the current timestep of the simulation
    x: current value of the state vector
    dt: the amount to increment by for Runge-Kutta
    ======
    returns:
    x_new: value of the state vector at the next timestep
    '''
    k1 = dt * dxdt(t, x)
    k2 = dt * dxdt(t + dt/2.0, x + k1/2.0)
    k3 = dt * dxdt(t + dt/2.0, x + k2/2.0)
    k4 = dt * dxdt(t + dt, x + k3)
    x_new = x + (k1 + 2.0*k2 + 2.0*k3 + k4)/6.0

    return x_new

rk = rk4(lambda t, x: x**2, 0.5, 2, 0.1)
assert np.isclose(rk, 0.526315781526278075), f"RK4 value: {rk}" #from an online RK4 solver
print("assertion passed")
```

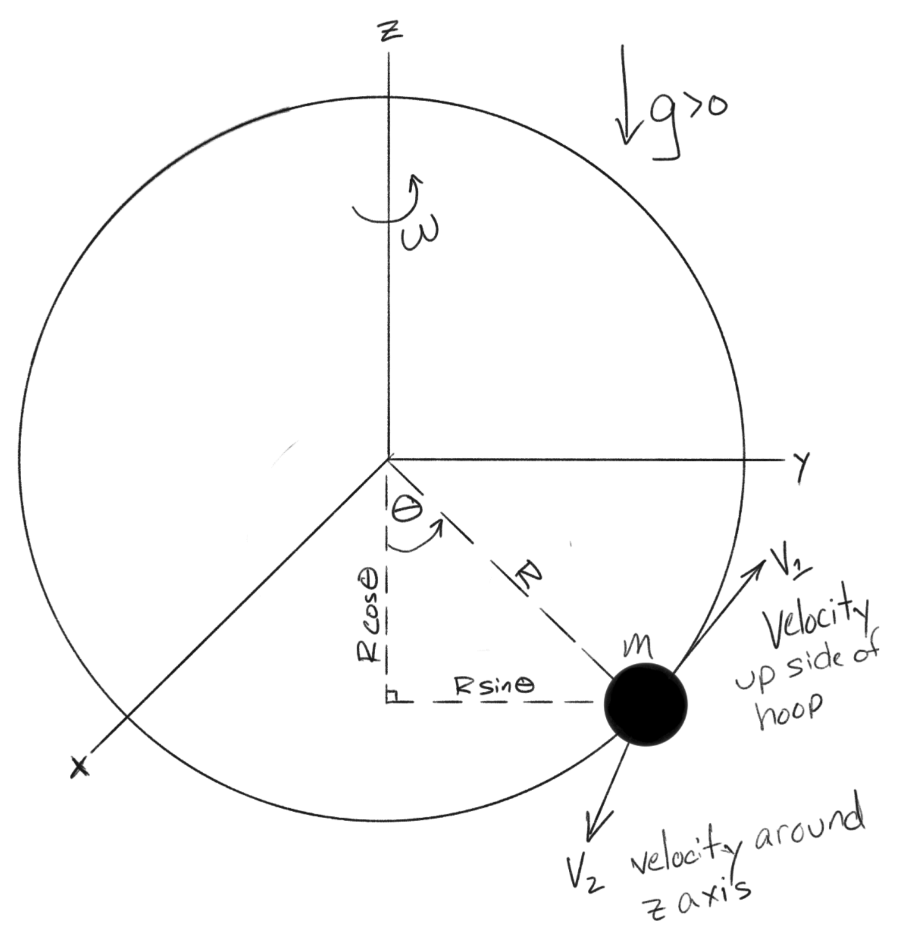
```
assertion passed
```

In [5]:
```python
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):
        t = tvec[i]
        xtraj[:,i]=integrate(f,x,t,dt)
        x = np.copy(xtraj[:,i])
    return xtraj
```

In [6]:

```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/raw/master/dynhoop2.png' width=500' height='350'></table>"))
```



Problem 1 (20pts)

Take the bead on a hoop example shown in the image above, model it using a torque 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 [7]:

```
#forced Euler-Lagrange equations, plus constraint equation
#constraint: x^2 + y^2 + z^2 - r^2 = 0

#grab variables
m, R, g = sym.symbols(r'm, R, g')
t = sym.symbols(r't')
tau = sym.symbols(r'\tau')

psi = sym.Function(r'\psi')(t)
theta = sym.Function(r'\theta')(t)

#express x, y, z in terms of angular variables
x = R * sym.sin(theta) * sym.cos(psi)
y = R * sym.sin(theta) * sym.sin(psi)
z = R * sym.cos(theta)
```

```python
xd = x.diff(t)
yd = y.diff(t)
zd = z.diff(t)

KE = 0.5 * m * (xd**2 + yd**2 + zd**2)
U = m * g * z

#build Lagrangian equation
lagrangian = KE - U
lagrangian = lagrangian.simplify()
print("Lagrangian:")
display(lagrangian)
```

Lagrangian:

$$Rm\left(0.5R\left(\sin^2(\theta(t))\left(\frac{d}{dt}\psi(t)\right)^2+\left(\frac{d}{dt}\theta(t)\right)^2\right)-g\cos(\theta(t))\right)$$

In [8]:
```python
q = sym.Matrix([theta, psi])
qd = q.diff(t)
qdd = qd.diff(t)

#construct E-L equations with forcing in psi direction
eqs = compute_EL(lagrangian, q)
forced_rhs = sym.Matrix([0, tau])
EL_eqns_new = sym.Eq(eqs.lhs, forced_rhs)

print("Forced Euler-Lagrange equations:")
display(EL_eqns_new)
```

Forced Euler-Lagrange equations:

$$\begin{bmatrix}-1.0R^2m\frac{d^2}{dt^2}\theta(t)+Rm\left(1.0R\sin(\theta(t))\cos(\theta(t))\left(\frac{d}{dt}\psi(t)\right)^2+g\sin(\theta(t))\right)\\ -1.0R^2m\sin^2(\theta(t))\frac{d^2}{dt^2}\psi(t)-2.0R^2m\sin(\theta(t))\cos(\theta(t))\frac{d}{dt}\psi(t)\frac{d}{dt}\theta(t)\end{bmatrix}=\begin{bmatrix}0\\ \tau\end{bmatrix}$$

In [9]:
```python
#solve equations of motion
eqns_solved = solve_EL(EL_eqns_new, qdd)
print("Solved:")
for eq in eqns_solved:
    display(eq.simplify())
```

Solved:

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

$$\frac{d^2}{dt^2}\psi(t)=-\frac{1.0R^2m\sin(2\theta(t))\frac{d}{dt}\psi(t)\frac{d}{dt}\theta(t)+1.0\tau}{R^2m\sin^2(\theta(t))}$$

Problem 2 (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 (invariant)? Is this invariance global or local?
2. Use small angle approximations to linearize your transformation(s) from the first question. The resulting new transformation(s) 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/these transformation(s).
3. Apply Noether's theorem to determine a conserved quantity. What does this quantity represent physically? Is there any rationale behind its 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 the x, y, and z axes seem to have the same influence on the system, rotation around some axes will influence the Lagrangian more than others will.

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.

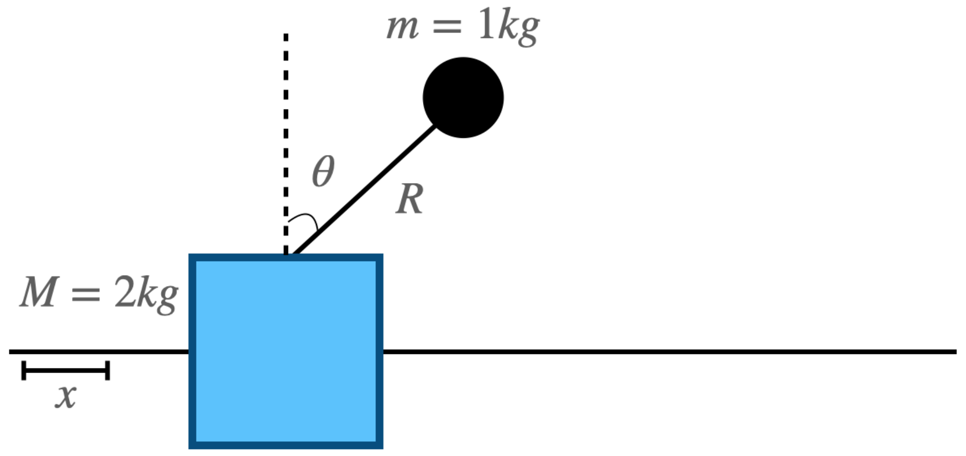
See written work

Problem 3 (20pts)

For the inverted cart-pendulum system in Homework 1 (feel free to make use of the provided solutions), compute the conserved momentum using Nöther's theorem. Plot the momentum for the same simulation parameters and initial conditions. Taking into account the conserved quantities, what is the minimal number of *states* in the cart/pendulum system that can vary? (Hint: In some coordinate systems it may *look* like all the states are varying, but if you choose your coordinates cleverly fewer states will vary.)

Turn in: A copy of code used to calculate the conserved quantity and your answer to the question. You don't need to turn in equations of motion, but you need to include the plot of the conserved quantity evaluated along the system trajectory.

In [10]:
```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/atulletaylor/ME314Figures/raw/main/hw4problem3.png' width='600' height='350'></td></tr></table>"))
```



In [11]:
```
#equations drawn from hw1

#1. grab variables
t = sym.symbols(r't') #ind. var.
R, m, M, g = sym.symbols(r'R, m, M, g') #constants

const_dict = {
    M: 2,
    m: 1,
    g: 9.8,
    R: 1
}

#2. define position, velocity, accel for masses 1 and 2 as functions
xm = sym.Function(r'x_m')(t)
theta = sym.Function(r'\theta')(t)

xmd = xm.diff(t)
xmdd = xmd.diff(t)

thetad = theta.diff(t)
thetadd = thetad.diff(t)

#intermediate variables: x and y of mass on pendulum
xp = xm + R * sym.sin(theta)
yp = R * sym.cos(theta)

xpd = xp.diff(t)
ypd = yp.diff(t)

#kinetic and potential energy terms

KE_M = 0.5 * M * xmd**2
KE_m = 0.5 * m * (xpd**2 + ypd**2)
KE_sys = KE_M + KE_m
U_sys = m * g * yp

print("Lagrangian function:")
lagrangian3 = KE_sys - U_sys
lagrangian3 = lagrangian3.simplify()
display(lagrangian3)
```

Lagrangian function:

$$0.5M\left(\frac{d}{dt}x_m(t)\right)^2 - Rgm\cos\left(\theta(t)\right) + 0.5m\left(R^2\left(\frac{d}{dt}\theta(t)\right)^2 + 2R\cos\left(\theta(t)\right)\frac{d}{dt}\theta(t)\frac{d}{dt}x_m(t) + \left(\frac{d}{dt}x_m(t)\right)^2\right)$$

In [12]:
```
#proposed transformation that conserves energy: given q = [x theta]T, G(q) = [1 0]
G = sym.Matrix([1,0])

#test local invariance by subbing in qe = q + eps * G(q) for q
eps = sym.symbols(r'\epsilon')
q = sym.Matrix([xm, theta])
qd = q.diff(t)

qe = q + eps * G
```

```python
qe_subs = {q[i] : qe[i] for i in range(len(q))}
lagrangian3_qe = lagrangian3.subs(qe_subs)
diff = lagrangian3_qe - lagrangian3

print("Transformed state vector q + eps * G(q):")
display(qe)
print("Lagrangian after transformation:")
display(lagrangian3_qe)
print("Difference L(q,qd) - L_eps(qe, qed):")
display(sym.simplify(diff))
```

```
Transformed state vector q + eps * G(q):
```

$$\begin{bmatrix}\epsilon + x_m(t)\\ \theta(t)\end{bmatrix}$$

```
Lagrangian after transformation:
```

$$0.5M\left(\frac{\partial}{\partial t}(\epsilon + x_m(t))\right)^2 - Rgm\cos(\theta(t)) + 0.5m\left(R^2\left(\frac{d}{dt}\theta(t)\right)^2 + 2R\cos(\theta(t))\frac{\partial}{\partial t}(\epsilon + x_m(t))\frac{d}{dt}\theta(t) + \left(\frac{\partial}{\partial t}(\epsilon + x_m(t))\right)^2\right)$$

```
Difference L(q,qd) - L_eps(qe, qed):
```

$$0$$

In [13]:

```python
#calculate conserved quantity via Noether

#define state of system
q = sym.Matrix([xm, theta])
qd = q.diff(t)
qdd = qd.diff(t)

n = len(q)
dLdQd_matrix = sym.zeros(n,1).T

for i, xd in enumerate(qd):
    dLdQd_matrix[i] = lagrangian3.diff(xd)

print("\ndL/dqdot terms for qd = (xd, thetad):")
display(sym.simplify(dLdQd_matrix))

#multiply dL_dQd by G(q) to get conserved quantity
p_cons = (dLdQd_matrix * G)[0]

print("G(q) in (x, theta):")
display(G)
print("Conserved quantity:")
display(p_cons.simplify())
```

```
dL/dqdot terms for qd = (xd, thetad):
```

$$\begin{bmatrix}1.0M\frac{d}{dt}x_m(t) + m\left(R\cos(\theta(t))\frac{d}{dt}\theta(t) + \frac{d}{dt}x_m(t)\right) & 1.0Rm\left(R\frac{d}{dt}\theta(t) + \cos(\theta(t))\frac{d}{dt}x_m(t)\right)\end{bmatrix}$$

```
G(q) in (x, theta):
```

$$\begin{bmatrix}1\\0\end{bmatrix}$$

```
Conserved quantity:
```

$$1.0M\frac{d}{dt}x_m(t) + m\left(R\cos(\theta(t))\frac{d}{dt}\theta(t) + \frac{d}{dt}x_m(t)\right)$$

Conserved quantity, therefore, is momentum of the cart in the x direction.

In [14]:

```python
#simulate system and plot
EL_eqns_p3 = compute_EL(lagrangian3, q)
eqns_solved = solve_EL(EL_eqns_p3, qdd)

print("Euler-Lagrange equations:")
display(EL_eqns_p3.simplify())
print("Solved:")
for eq in eqns_solved:
    display(eq.simplify())
```

```
Euler-Lagrange equations:
```

$$\begin{bmatrix}0\\0\end{bmatrix} = \begin{bmatrix}-1.0M\frac{d^2}{dt^2}x_m(t) - 1.0m\left(-R\sin(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + R\cos(\theta(t))\frac{d^2}{dt^2}\theta(t) + \frac{d^2}{dt^2}x_m(t)\right)\\ 1.0Rm\left(-R\frac{d^2}{dt^2}\theta(t) + g\sin(\theta(t)) - \cos(\theta(t))\frac{d^2}{dt^2}x_m(t)\right)\end{bmatrix}$$

```
Solved:
```

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

$$\frac{d^2}{dt^2}\theta(t) = \frac{\left(Mg - Rm\cos(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + gm\right)\sin(\theta(t))}{R\left(M + m\sin^2(\theta(t))\right)}$$

In [15]:
```
xmdd_sy = eqns_solved[0].simplify().subs(const_dict)
thetadd_sy = eqns_solved[1].simplify().subs(const_dict)

q_ext = sym.Matrix([q, qd])
xmdd_np = sym.lambdify(q_ext, xmdd_sy.rhs)
thetadd_np = sym.lambdify(q_ext, thetadd_sy.rhs)

print("Equations of motion after substitution:")
display(xmdd_sy)
display(thetadd_sy)
```

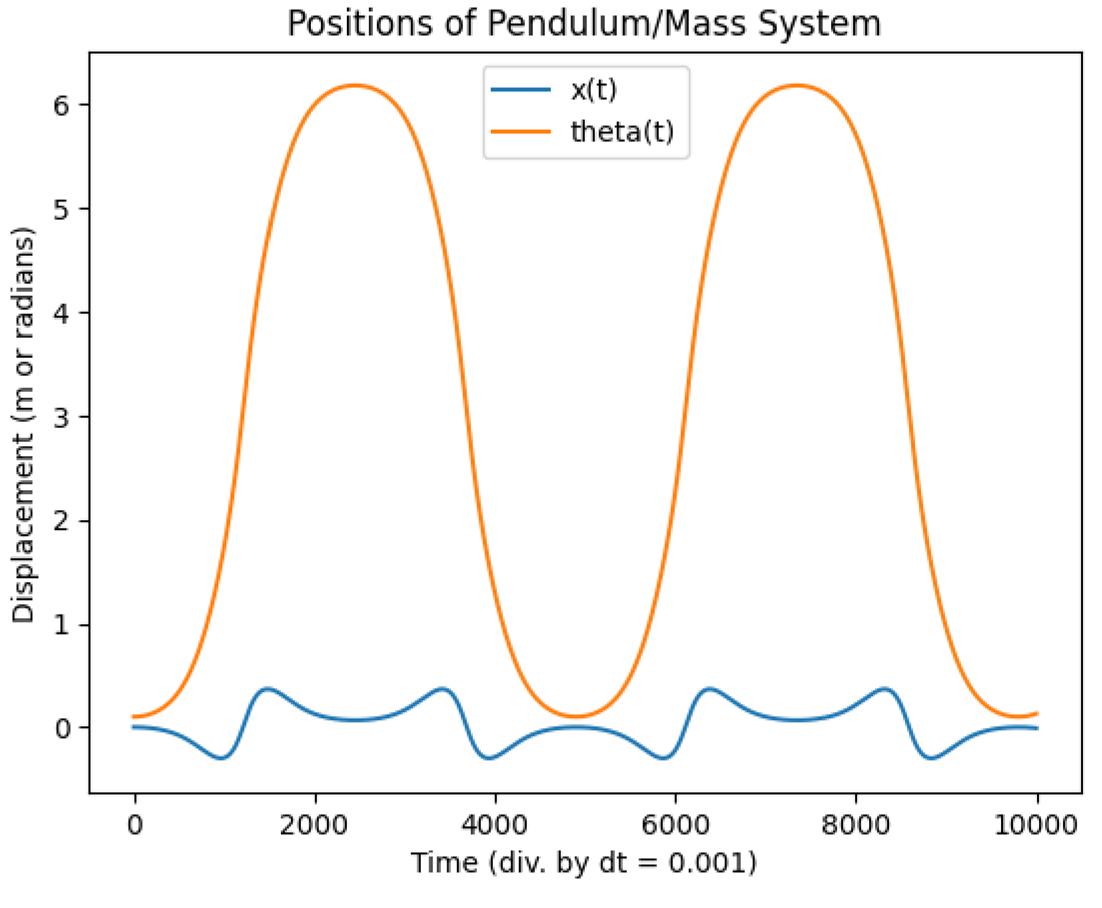
```
Equations of motion after substitution:
```

$$\frac{d^2}{dt^2}x_m(t) = \frac{\left(-9.8\cos\left(\theta(t)\right) + \left(\frac{d}{dt}\theta(t)\right)^2\right)\sin\left(\theta(t)\right)}{\sin^2\left(\theta(t)\right) + 2}$$

$$\frac{d^2}{dt^2}\theta(t) = \frac{\left(-\cos\left(\theta(t)\right)\left(\frac{d}{dt}\theta(t)\right)^2 + 29.4\right)\sin\left(\theta(t)\right)}{\sin^2\left(\theta(t)\right) + 2}$$

In [16]:
```
def dxdt_p3(t, s):
    return np.array([s[2], s[3], xmdd_np(*s), thetadd_np(*s)])

#ICs:
t_span = [0, 10]
dt = 0.001
s0 = [0, 0.1, 0, 0]

q_array = simulate(dxdt_p3, s0, t_span, dt, rk4)

plt.plot(q_array[0])
plt.plot(q_array[1])

plt.legend(["x(t)", "theta(t)"])
plt.xlabel(f"Time (div. by dt = {dt})")
plt.ylabel("Displacement (m or radians)")
plt.title("Positions of Pendulum/Mass System")
```

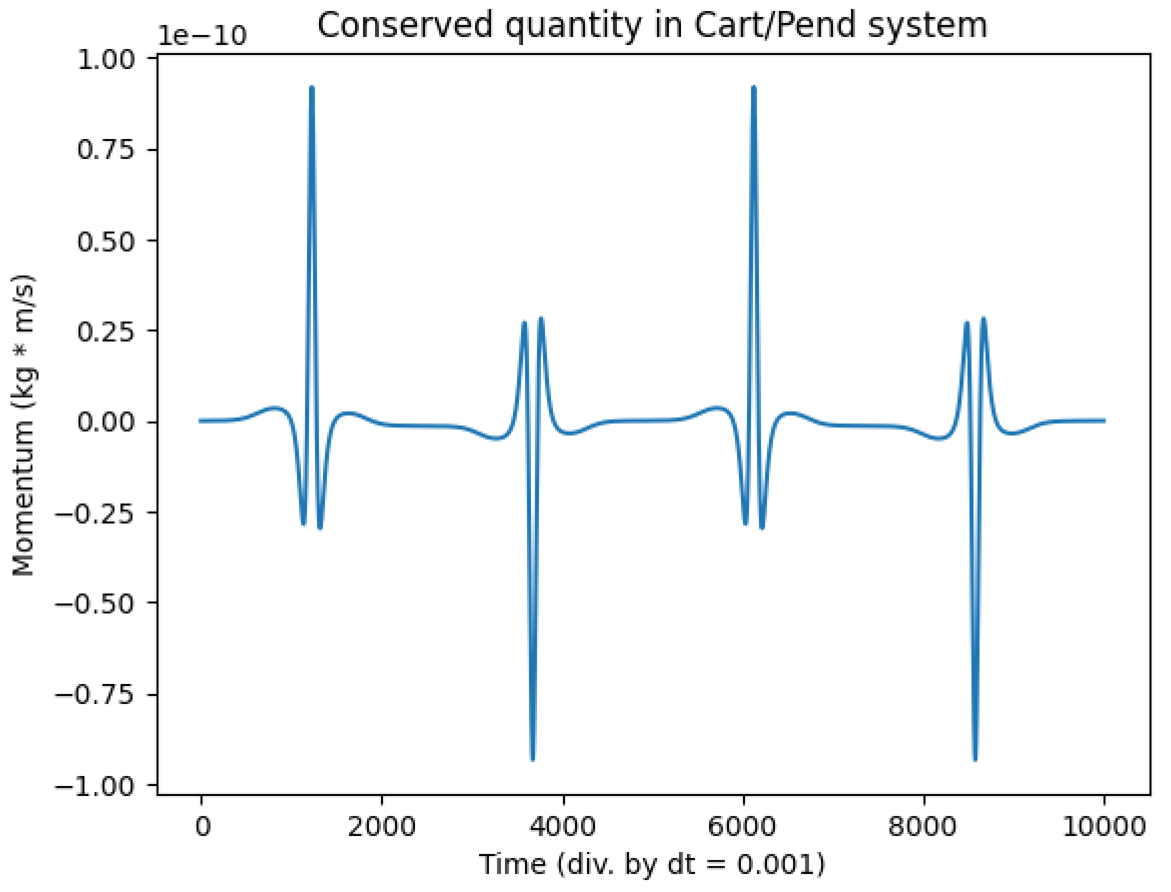
Out[16]:
```
Text(0.5, 1.0, 'Positions of Pendulum/Mass System')
```



In [17]:
```
#evaluate conserved quantity in system
p_cons = p_cons.subs(const_dict)
display(p_cons)

#xm, theta, xmd, thetad
p_cons_np = sym.lambdify(q_ext, p_cons)
#help(p_cons_np)

#apply conserved quantity calculation to system
p_cons_array = [p_cons_np(*s) for s in q_array.T]
plt.plot(p_cons_array)

plt.xlabel(f"Time (div. by dt = {dt})")
plt.ylabel("Momentum (kg * m/s)")
plt.title("Conserved quantity in Cart/Pend system")
```

$$1.0\cos\left(\theta(t)\right)\frac{d}{dt}\theta(t) + 3.0\frac{d}{dt}x_m(t)$$

Out[17]: Text(0.5, 1.0, 'Conserved quantity in Cart/Pend system')



The minimal number of states in the system that can vary are θ_m and x_M. Any Cartesian coordinate system for the mass would give an inaccurate number of states, as x and y of the smaller mass can be determined from x of the smaller mass and theta.

Problem 4 (30 pts)

Using the same inverted cart pendulum system, add a constraint such that the pendulum follows the path of a parabola with a vertex of $(1, 0)$.

Then, simulate the system using x and θ as the configuration variables for $t \in [0, 15]$ with $dt = 0.01$. The constants are $M = 2, m = 1, R = 1, g = 9.8$. Use the initial conditions $x = 0, \theta = 0, \dot{x} = 0, \dot{\theta} = 0.01$ for your simulation.

You should use the Runge–Kutta integration function provided in previous homework for simulation. Plot the simulated trajectory for x, θ versus time. We have a provided an animation function for testing.

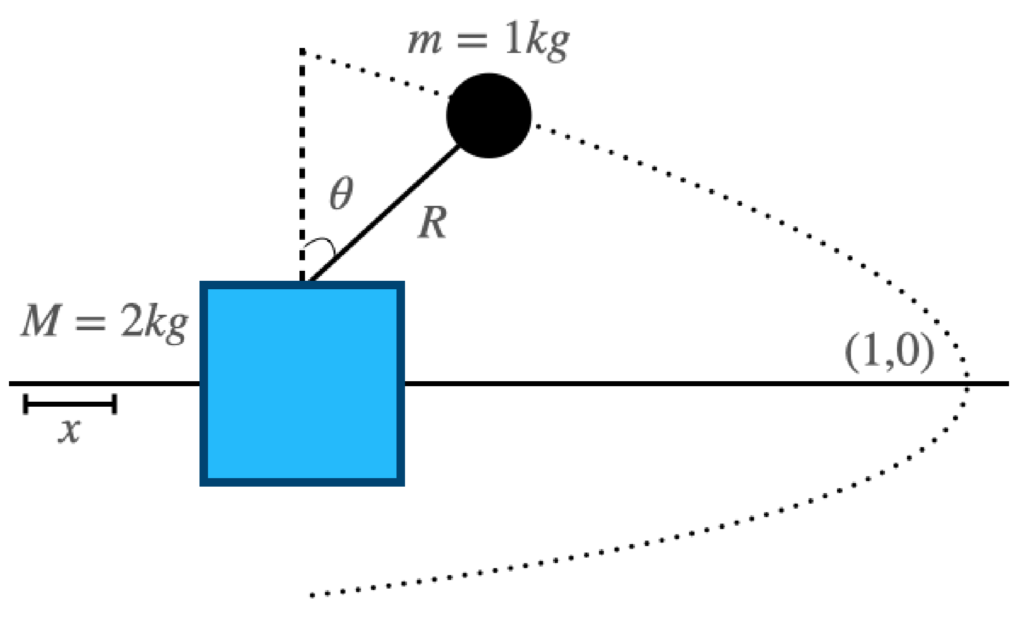
Hint 1: You will need the time derivatives of ϕ to solve the system of equations.

Hint 2: Make sure to be solving for λ at the same time as your equations of motion.

Hint 3: Note that if you make your initial condition velocities faster or dt lower resolution, you may not be able to simulate the system because this is challenging constraint.

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 [18]:
```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/atulletaylor/ME314Figures/raw/main/hw4p4.png' width='600' height='350'></td></tr></table>"))
```



In [19]:
```
#set up left hand side of constrained Euler-Lagrange equations
#this uses a lot of code from problem 3 so make sure to run that first
EL_lhs = EL_eqns_p3.lhs
lamb = sym.symbols(r'\lambda')

#let constraint be approximately (x-1) =-y^2
phi = xp - 1 + yp**2 #equals 0

#need gradient of constraint for RHS of equation and for additional constraint EQ
eqns_solved = solve_constrained_EL(lamb, phi, q, EL_lhs)
```

Equations to be solved (LHS - lambda * grad(phi) = 0):

$$\left[\begin{matrix}-1.0M\frac{d^2}{dt^2}x_m(t) - \lambda - 0.5m\left(-2R\sin(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + 2R\cos(\theta(t))\frac{d^2}{dt^2}\theta(t) + 2\frac{d^2}{dt^2}x_m(t)\right)\\ Rgm\sin(\theta(t)) - 1.0Rm\sin(\theta(t))\frac{d}{dt}\theta(t)\frac{d}{dt}x_m(t) - \lambda\left(-2R^2\sin(\theta(t))\cos(\theta(t)) + R\cos(\theta(t))\right) - 0.5m\left(2R^2\frac{d^2}{dt^2}\theta(t) - 2R\sin(\theta(t))\frac{d}{dt}\theta(t)\frac{d}{dt}x_m(t) + 2R\cos(\theta(t))\frac{d^2}{dt^2}x_m(t)\right)\\ 2R^2\sin^2(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 - 2R^2\sin(\theta(t))\cos(\theta(t))\frac{d^2}{dt^2}\theta(t) - 2R^2\cos^2(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 - R\sin(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + R\cos(\theta(t))\frac{d^2}{dt^2}\theta(t) + \frac{d^2}{dt^2}x_m(t)\end{matrix}\right] = \left[\begin{matrix}0\\0\\0\end{matrix}\right]$$

Variables to solve for:

$$\left[\begin{matrix}\frac{d^2}{dt^2}x_m(t)\\ \frac{d^2}{dt^2}\theta(t)\\ \lambda\end{matrix}\right]$$

In [20]:
```python
new_eqs = []
for i, eq in enumerate(eqns_solved):

    if i == 2:
        continue

    new_eq = eq.simplify()
    new_eqs.append(new_eq)
    display(new_eq)
```

$$\frac{d^2}{dt^2}x_m(t) = \frac{m\left(4.0R^3\sin^4(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 - 8.0R^3\sin^2(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + 4.0R^3\left(\frac{d}{dt}\theta(t)\right)^2 - 2.0R^2\sin^3(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + 2.0Rg\sin(\theta(t))\cos(\theta(t)) + 1.0R\left(\frac{d}{dt}\theta(t)\right)^2 - 1.0g\cos(\theta(t))\right)\sin(\theta(t))}{0.5MR^2\cdot(1-\cos(4\theta(t))) - 4.0MR\sin(\theta(t))\cos^2(\theta(t)) + M\cos^2(\theta(t)) + 0.5R^2m(1-\cos(4\theta(t))) - m\cos^2(\theta(t)) + m}$$

$$\frac{d^2}{dt^2}\theta(t) = \frac{-1.0MR^3\sin(4\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + 0.5MR^2\cos(\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + 1.5MR^2\cos(3\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + 0.5MR\sin(2\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 - 1.0R^3m\sin(4\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 - 0.5Rm\sin(2\theta(t))\left(\frac{d}{dt}\theta(t)\right)^2 + 1.0gm\sin(\theta(t))}{R\left(0.5MR^2\cdot(1-\cos(4\theta(t))) - 4.0MR\sin(\theta(t))\cos^2(\theta(t)) + M\cos^2(\theta(t)) + 0.5R^2m(1-\cos(4\theta(t))) - m\cos^2(\theta(t)) + m\right)}$$

In [21]:
```python
#substitute in variables and lambdify
xdd_sy = new_eqs[0].rhs.subs(const_dict)
thetadd_sy = new_eqs[1].rhs.subs(const_dict)

q_ext = sym.Matrix([xm, theta, xmd, thetad])

xdd_np = sym.lambdify(q_ext, xdd_sy)
thetadd_np = sym.lambdify(q_ext, thetadd_sy)
```

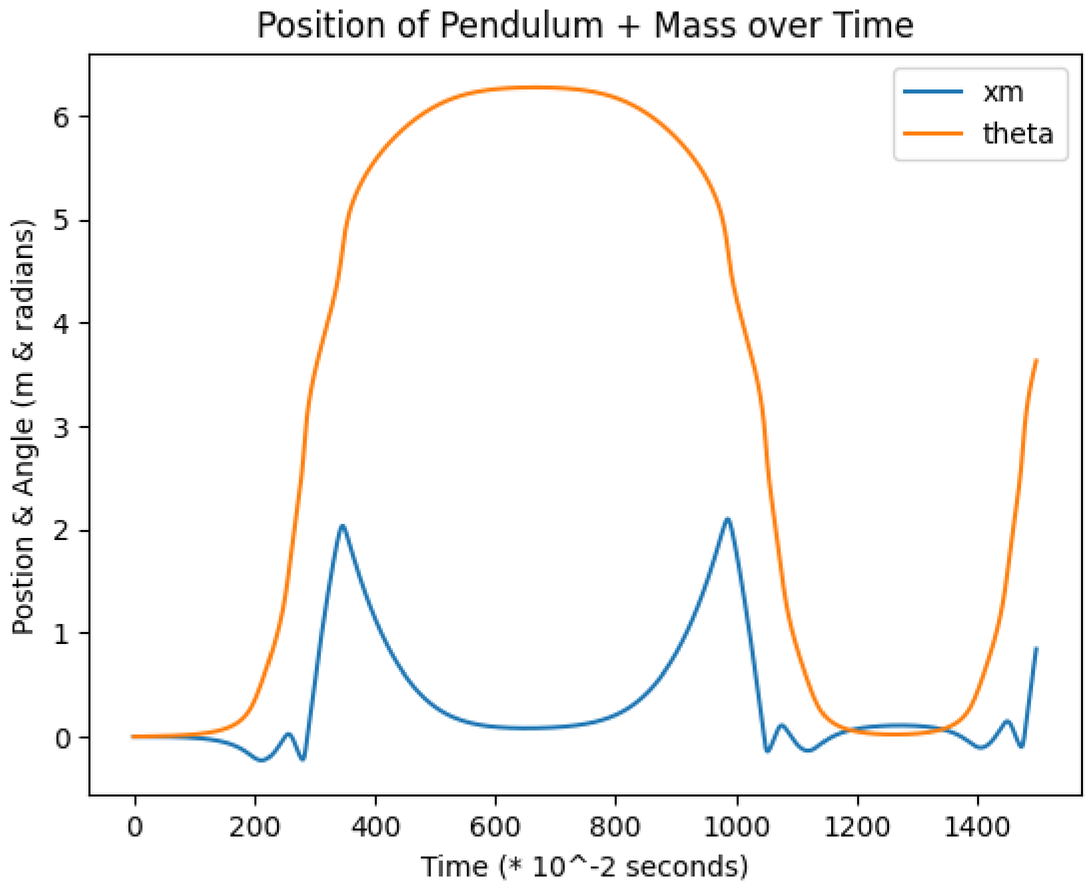
In [22]:
```python
#simulate motion of system
def dxdt_p4(t, s):
     return np.array([s[2],s[3], xdd_np(*s), thetadd_np(*s)])

x0 = [0, 0, 0, 0.01]
tspan = [0,15]
dt = 0.01

q_array = simulate(dxdt_p4, x0, tspan, dt, rk4)
```

In [23]:
```python
print(q_array)
```

```
[[-8.16747488e-09 -6.53556090e-08 -2.20636083e-07 ...  7.61583247e-01
   8.02052798e-01  8.42266266e-01]
 [ 1.00018171e-04  2.00105399e-04  3.00310827e-04 ...  3.58194666e+00
   3.60589272e+00  3.62943791e+00]
 [-2.45046508e-06 -9.80502298e-06 -2.20702267e-05 ...  4.05984387e+00
   4.03411115e+00  4.00861726e+00]
 [ 1.00044518e-02  1.00138129e-02  1.00280938e-02 ...  2.41623850e+00
   2.37378936e+00  2.33598433e+00]]
```

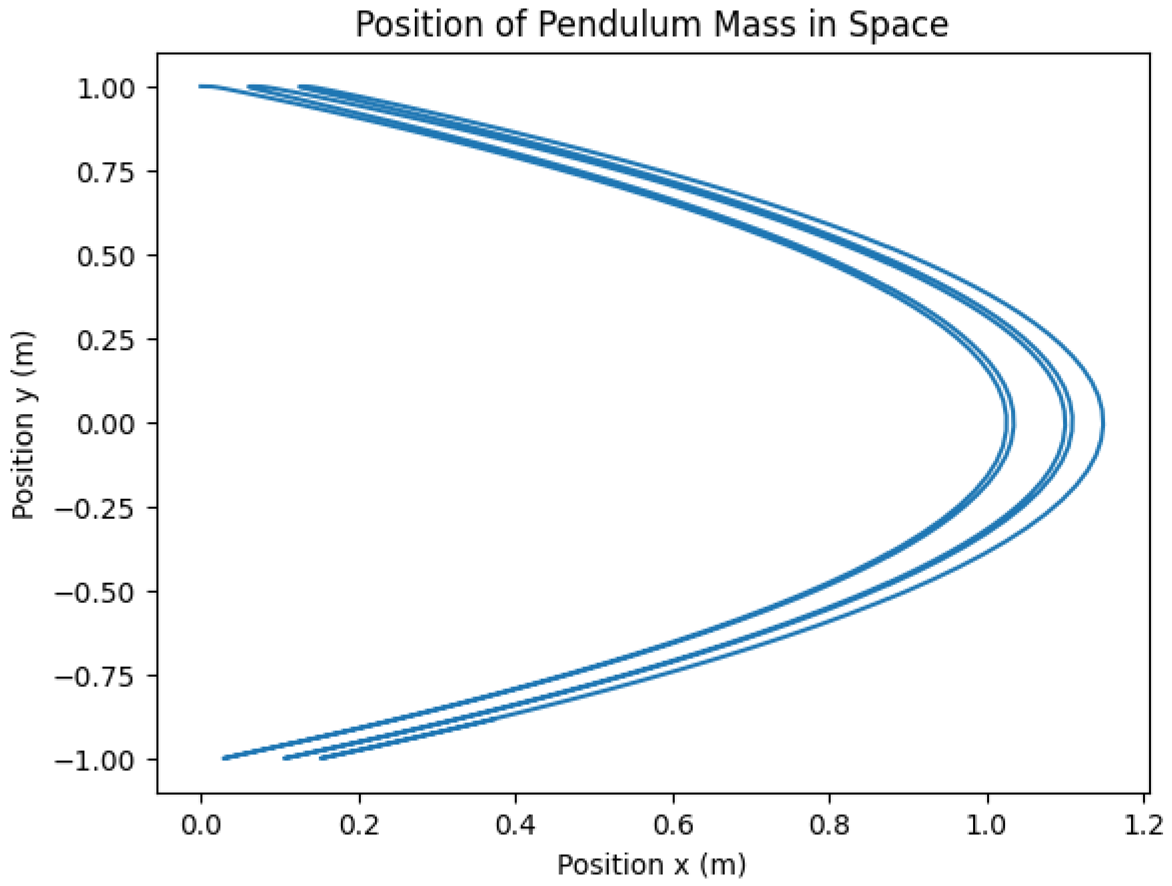
In [24]:
```python
xm_array = q_array[0]
theta_array = q_array[1]
plt.plot(xm_array)
plt.plot(theta_array)

plt.xlabel("Time (* 10^-2 seconds)")
plt.ylabel("Postion & Angle (m & radians)")
plt.title("Position of Pendulum + Mass over Time")
plt.legend(['xm','theta'])
```

Out[24]: <matplotlib.legend.Legend at 0x22b70df4430>



In [25]:
```
#plot position of mass on pendulum over time to ensure it's constrained
Rg = const_dict[R]
xp_array = xm_array + Rg * np.sin(theta_array)
yp_array = Rg * np.cos(theta_array)

#plot x and y
plt.plot(xp_array, yp_array)
plt.xlabel("Position x (m)")
plt.ylabel("Position y (m)")
plt.title("Position of Pendulum Mass in Space")
```

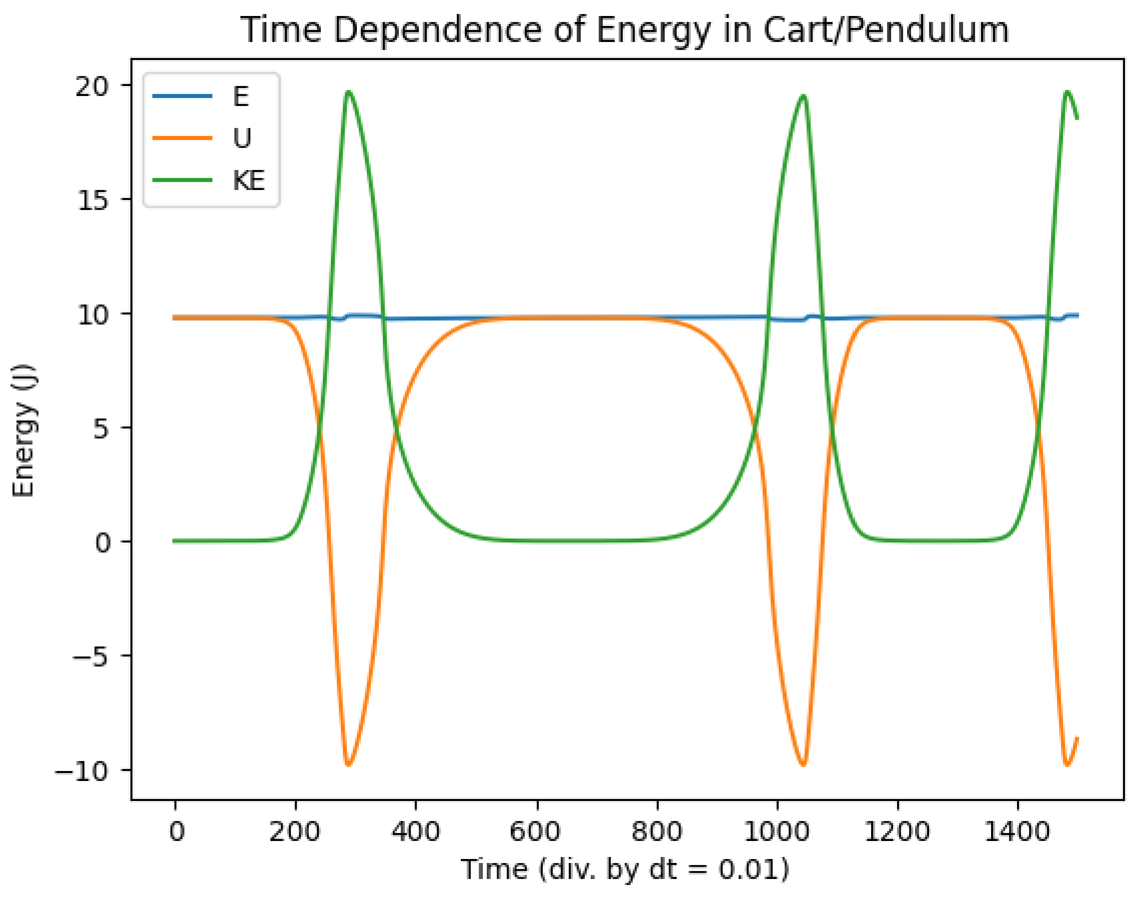
Out[25]: Text(0.5, 1.0, 'Position of Pendulum Mass in Space')



In [26]:
```
#plot energy in system over time to ensure it's conserved
Rc = const_dict[R]
mc = const_dict[m]
Mc = const_dict[M]
gc = const_dict[g]


def KE_m(s):
    [xm, theta, xmd, thetad] = s
    xpd = xmd + Rc * np.cos(theta) * thetad
    ypd = -Rc * np.sin(theta) * thetad
    return 0.5 * mc * (xpd**2 + ypd**2)

def KE_M(s):
    [xm, theta, xmd, thetad] = s
    return 0.5 * Mc * xmd**2

def U_m(s):
    [xm, theta, xmd, thetad] = s
    yp = Rc * np.cos(theta)
```

```
    return mc * gc * yp

def E(s):
    return KE_m(s) + KE_M(s) + U_m(s)

KE_array = [KE_m(s) + KE_M(s) for s in q_array.T]
U_array = [U_m(s) for s in q_array.T]
E_array = [E(s) for s in q_array.T]

plt.plot(E_array)
plt.plot(U_array)
plt.plot(KE_array)
plt.legend(['E','U','KE'])

plt.title("Time Dependence of Energy in Cart/Pendulum")
plt.xlabel(f"Time (div. by dt = {dt})")
plt.ylabel('Energy (J)')
```

Out[26]: Text(0, 0.5, 'Energy (J)')

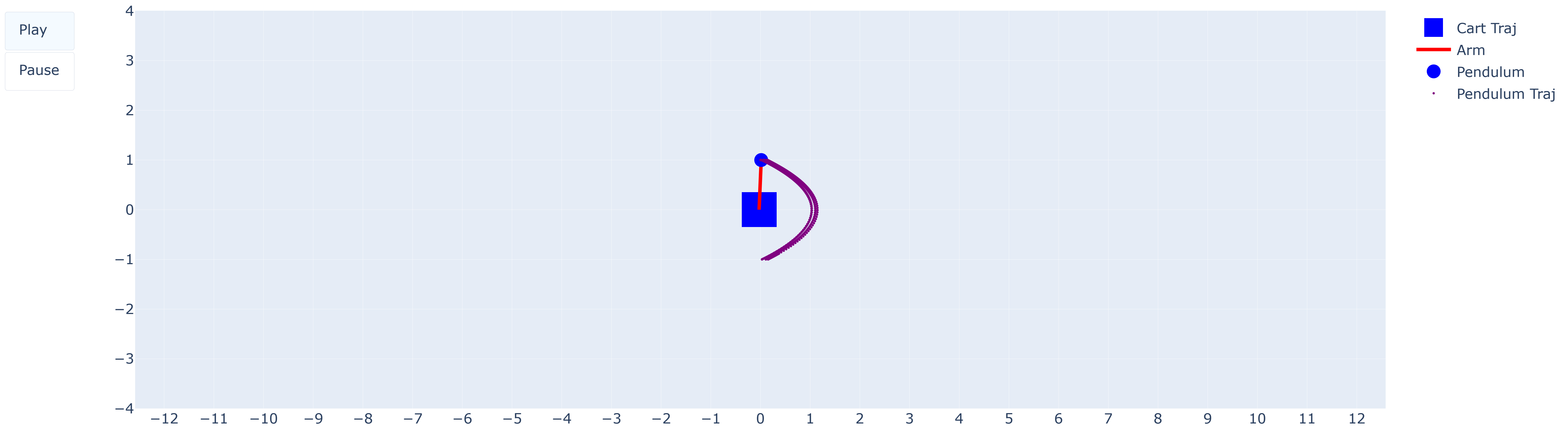


In [27]:
```
def animate_cart_pend(traj_array,R=1,T=15):
    """
    Function to generate web-based animation of double-pendulum system

    Parameters:
    ================================================
    traj_array:
        trajectory of theta and x, should be a NumPy array with
        shape of (2,N)
    R:
        length of the pendulum
    T:
        length/seconds of animation duration

    Returns: None
    """

    ################################
    # Imports required for animation.
    from plotly.offline import init_notebook_mode, iplot
    from IPython.display import display, HTML
    import plotly.graph_objects as go

    #######################
    # Browser configuration.
    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.
    xcart=traj_array[0]
    ycart = 0.0*np.ones(traj_array[0].shape)
    N = len(traj_array[1])

    xx1=xcart+R*np.sin(traj_array[1])
    yy1=R*np.cos(traj_array[1])

     # Need this for specifying length of simulation

    ####################################
    # Using these to specify axis limits.
    xm=-4
    xM= 4
    ym=-4
    yM= 4

    ###########################
    # Defining data dictionary.
    # Trajectories are here.
    data=[
          dict(x=xcart, y=ycart,
               mode='markers', name='Cart Traj',
               marker=dict(color="green", size=2)
              ),
          dict(x=xx1, y=yy1,
               mode='lines', name='Arm',
               line=dict(width=2, color='blue')
              ),
           dict(x=xx1, y=yy1,
               mode='lines', name='Pendulum',
               line=dict(width=2, color='purple')
              ),

           dict(x=xx1, y=yy1,
               mode='markers', name='Pendulum Traj',
               marker=dict(color="purple", size=2)
              ),
        ]

    ################################
    # Preparing simulation layout.
    # Title and axis ranges are here.
    layout=dict(xaxis=dict(range=[xm, xM], autorange=False, zeroline=False,dtick=1),
                yaxis=dict(range=[ym, yM], autorange=False, zeroline=False,scaleanchor = "x",dtick=1),
                title='Cart 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.
    # This is what draws the lines from
    # joint to joint of the pendulum.
    frames=[dict(data=[go.Scatter(
                            x=[xcart[k]],
                            y=[ycart[k]],
                            mode="markers",
                            marker_symbol="square",
                            marker=dict(color="blue", size=30)),

                       dict(x=[xx1[k],xcart[k]],
                            y=[yy1[k],ycart[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)),

                      ]) for k in range(N)]

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

In [28]:
```
animate_cart_pend(q_array, R=Rc, T=15)
```

Cart Pendulum Simulation



In [ ]:

2) 1. Rotation matrices: | Small-angle approx.:

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_{\theta\varepsilon} = \begin{bmatrix} 1 & -\varepsilon & 0 \\ \varepsilon & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_\psi = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix} \qquad R_{\psi\varepsilon} = \begin{bmatrix} 1 & 0 & \varepsilon \\ 0 & 1 & 0 \\ -\varepsilon & 0 & 1 \end{bmatrix}$$

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \qquad R_{\phi\varepsilon} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\varepsilon \\ 0 & \varepsilon & 1 \end{bmatrix}$$

Only rotation about the z axis, R_θ, keeps the Lagrangian fixed/invariant. This invariance is global as rotation by any amount θ about the z axis will not change the Lagrangian.

Format of problem 2.1: (for use in understanding my written work)

1. Show R_θ transformation is locally invariant in L_ε

2. Show R_θ transformation is globally invariant in L_ε

3. Show R_ϕ and R_ψ transformations are not locally invariant in L_ε, and therefore cannot be globally invariant in L_ε

2.1) Check for local invariance of R_θ transformation using small-angle approximation:

$$q = \begin{bmatrix} x \\ y \\ z \end{bmatrix} ; \quad q_\varepsilon = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \varepsilon \begin{bmatrix} -y \\ x \\ 0 \end{bmatrix}$$

$$\dot{q}_\varepsilon = \frac{d}{dt}\left(q + \varepsilon G(q) \right)$$
$$= \dot{q} + \varepsilon \dot{G}q$$
$$= \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} + \varepsilon \begin{bmatrix} -\dot{y} \\ \dot{x} \\ 0 \end{bmatrix}$$

Substitute in the perturbed versions of q, $\dot{q}_\varepsilon$, into the Lagrangian:

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

$$L_\varepsilon = \frac{1}{2}m\left((\dot{x} - \varepsilon\dot{y})^2 + (\dot{y} + \varepsilon\dot{x})^2 + \dot{z}^2 \right)$$
$$- \frac{1}{2}k\left((x - \varepsilon y)^2 + (y + \varepsilon x)^2 + z\right)^2 - mgz$$
$$= \frac{1}{2}m\left(\dot{x}^2 - \cancel{2\varepsilon\dot{x}\dot{y}} + \varepsilon^2\dot{y}^2 + \dot{y}^2 + \cancel{2\varepsilon\dot{x}\dot{y}} + 2\varepsilon\dot{x}^2 \right.$$
$$\left. + \dot{z}^2 \right) - \frac{1}{2}k\left(x^2 - \cancel{2\varepsilon xy} + \varepsilon^2 y^2 + y^2 + \cancel{2\varepsilon xy} + \varepsilon^2 x^2 \right)$$
$$- mgz$$

$$L_\varepsilon = \frac{1}{2}m\left(\dot{x}^2 + \dot{y}^2 + \varepsilon^2(\dot{x}^2 + \dot{y}^2) + \dot{z}^2\right) - \frac{1}{2}k\left(x^2 + y^2 + \varepsilon^2(x^2 + y^2)\right.$$
$$\left. + z^2\right) - mgz$$

2.1)

Subtract $L_\varepsilon - L$:

$L_\varepsilon - L =$

$$\left[\frac{1}{2}m\left(\cancel{\dot{x}^2} + \cancel{\dot{y}^2} + \varepsilon^2(\dot{x}^2 + \dot{y}^2) + \cancel{\dot{z}^2}\right) - \frac{1}{2}k\left(\cancel{x^2} + \cancel{y^2} + \varepsilon^2(x^2+y^2) + \cancel{z^2}\right) - \cancel{mgz}\right]$$

$$- \left[\frac{1}{2}m\left(\cancel{\dot{x}^2} + \cancel{\dot{y}^2} + \cancel{\dot{z}^2}\right) - \frac{1}{2}k\left(\cancel{x^2} + \cancel{y^2} + \cancel{z^2}\right) - \cancel{mgz}\right]$$

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

To check for local invariance: let $\frac{\partial}{\partial\varepsilon}(L_\varepsilon)\Big|_{\varepsilon=0} \to 0$

$$\frac{\partial}{\partial\varepsilon}(L_\varepsilon)\Big|_{\varepsilon=0} = \varepsilon\left(m(\dot{x}^2+\dot{y}^2) - k(x^2+y^2)\right)\Big|_{\varepsilon=0}$$
$$= 0$$

The transformation corresponding to R_θ, an rotation about z axis, leaves the Lagrangian locally invariant.

2.1) Check global invariance of R_θ transformation by modeling transformation without small-angle approximation:

$$q = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad q_\varepsilon = \begin{bmatrix} x\cos(\varepsilon) - y\sin(\varepsilon) \\ x\sin(\varepsilon) + y\cos(\varepsilon) \\ z \end{bmatrix}$$

$$\dot{q}_\varepsilon = \frac{d}{dt}(q_\varepsilon) = \begin{bmatrix} \dot{x}\cos(\varepsilon) - \dot{y}\sin(\varepsilon) \\ \dot{x}\sin(\varepsilon) + \dot{y}\cos(\varepsilon) \\ \dot{z} \end{bmatrix}$$ → assume ε term has no dependence on time t

Lagrangian: substitute in q_ε and $\dot{q}_\varepsilon$ to see if perturbations change L

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

$$L_{0,\varepsilon} = \frac{1}{2}m\left((\dot{x}\cos\varepsilon - \dot{y}\sin\varepsilon)^2 + (\dot{x}\sin\varepsilon + \dot{y}\cos\varepsilon)^2 + \dot{z}^2\right) - \frac{1}{2}k\left((x\cos\varepsilon - y\sin\varepsilon)^2 + (x\sin\varepsilon + y\cos\varepsilon)^2 + z^2\right) - mgz$$

$$= \frac{1}{2}m\left[\dot{x}^2\cos^2\varepsilon - \cancel{2\dot{x}\dot{y}\cos\varepsilon\sin\varepsilon} + \dot{y}^2\sin^2\varepsilon + \dot{x}^2\sin^2\varepsilon + \cancel{2\dot{x}\dot{y}\cos\varepsilon\sin\varepsilon} + \dot{y}^2\cos^2\varepsilon + \dot{z}^2\right]$$
$$- \frac{1}{2}k\left[x^2\cos^2\varepsilon - \cancel{2xy\cos\varepsilon\sin\varepsilon} + y^2\sin^2\varepsilon + x^2\sin^2\varepsilon + \cancel{2xy\cos\varepsilon\sin\varepsilon} + y^2\cos^2\varepsilon + z^2\right]$$
$$- mgz$$

2.1)

$$L_{\theta,\varepsilon} = \frac{1}{2}m\left((\dot{x}^2)(\cos^2\varepsilon + \sin^2\varepsilon) + (\dot{y}^2)(\cos^2\varepsilon + \sin^2\varepsilon) + \dot{z}^2\right)$$
$$- \frac{1}{2}k\left((x^2)(\cos^2\varepsilon + \sin^2\varepsilon) + (y^2)(\cos^2\varepsilon + \sin^2\varepsilon) + z^2\right)$$
$$- mgz$$

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

$L_{\theta\varepsilon} - L = 0 \longrightarrow$ The Lagrangian of the system does not depend on the magnitude of perturbation ε.

2.1) Test of local invariance with rotation about x axis $R\phi$:

$$\begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \varepsilon \begin{bmatrix} 0 \\ -z \\ y \end{bmatrix}$$

Sub into the Lagrangian to see if it changes:

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

$$L_{\phi\varepsilon} = \frac{1}{2}m\left(\dot{x}^2 + (\dot{y} - \varepsilon\dot{z})^2 + (\dot{z} + \varepsilon\dot{y})^2\right)$$
$$- \frac{1}{2}k\left(x^2 + (y - \varepsilon z)^2 + (z + \varepsilon y)^2\right)$$
$$- mg(z + \varepsilon y)$$

2.1)

$$L_{\varphi,\varepsilon} = \frac{1}{2}m\left(\dot{x}^2 + \dot{y}^2 - \cancel{2\varepsilon\dot{y}\dot{z}} + \varepsilon^2\dot{z}^2 + \dot{z}^2 + \cancel{2\varepsilon\dot{y}\dot{z}} + \varepsilon^2\dot{y}^2\right)$$
$$-\frac{1}{2}k\left(x^2 + y^2 - \cancel{2\varepsilon yz} + \varepsilon^2 z^2 + z^2 + \cancel{2\varepsilon yz} + \varepsilon^2 y^2\right)$$
$$-mg(z + \varepsilon y)$$

$$= \frac{1}{2}m\left(\dot{x}^2 + \dot{y}^2 + \dot{z}^2 + \varepsilon^2(\dot{y}^2 + \dot{z}^2)\right)$$
$$-\frac{1}{2}k\left(x^2 + y^2 + z^2 + \varepsilon^2(y^2 + z^2)\right)$$
$$-mg(z + \varepsilon y)$$

$$L_{\varphi,\varepsilon} = L + \varepsilon^2\left(\frac{1}{2}m(\dot{y}^2 + \dot{z}^2) - \frac{1}{2}k(y^2 + z^2)\right) - mg\varepsilon y$$

Check if $\frac{d}{d\varepsilon} L_{\varphi,\varepsilon}\Big|_{\varepsilon=0} = 0$:

$$\frac{d}{d\varepsilon} L_{\varphi,\varepsilon}\Big|_{\varepsilon=0} = \varepsilon\left(m(\dot{y}^2 + \dot{z}^2) - k(y^2 + z^2)\right) - mgy\Big|_{\varepsilon=0}$$
$$= \underline{\underline{mgy}} \neq 0.$$

The transformation R_φ, rotation about the x axis, is not locally invariant. R_ψ is also not locally invariant as its Lagrangian $L_{\psi,\varepsilon}$ has term $L_{\psi,\varepsilon} = \ldots - mg(z - \varepsilon x)$. $\frac{d}{d\varepsilon} L_{\psi,\varepsilon}\Big|_{\varepsilon=0}$ has term $\underline{\underline{mgx}} \neq 0$ that leaves $L_{\psi,\varepsilon}$ <u>not</u> locally invariant.

2.2)

Linearized transformation for $R_\Theta \cdot q$:

$$q_\varepsilon = q + \varepsilon G(q) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \varepsilon \begin{bmatrix} -y \\ x \\ 0 \end{bmatrix};$$

$L(q_{\varepsilon}, \dot{q}_\varepsilon) - L(q, \dot{q}) = 0$; see calculations in 2.1 for derivation of that result.

2.3) Conserved Quantity via Noether:

$$P_{\text{conserved}} = \frac{\partial L}{\partial \dot{Q}} G(q)$$

$$\frac{\partial L}{\partial \dot{Q}} = \begin{bmatrix} m\dot{x} \\ m\dot{y} \\ m\dot{z} \end{bmatrix}^T, \quad G(q) = \begin{bmatrix} -y \\ x \\ 0 \end{bmatrix} \quad (q_\varepsilon = q + \varepsilon G(q))$$

$$P_{\text{cons}} = \begin{bmatrix} m\dot{x} & m\dot{y} & m\dot{z} \end{bmatrix} \begin{bmatrix} -y \\ x \\ 0 \end{bmatrix}$$

$\boxed{P_{\text{cons}} = m(x\dot{y} - y\dot{x})}$ = angular momentum about the z axis.

The rationale behind conservation of this value is that there are no forces or torques, conservative or non-conservative, that act on the particle about the z axis.

3) Find conserved momentum in cart-pendulum system using Noether:

1) Find a transformation $G(q)$ that leaves $L(q, \dot{q})$ locally invariant

2) Test local invariance with $L(q + \varepsilon Q(q))$ or $L(R(q, \varepsilon))$; global invariance with general, non-small approximations

3) Find $p_{cons} = \frac{\partial L}{\partial \dot{q}} G(q)$

1) State vector: $[x_m, \theta]^T$

Transformation $G(q)$ that leaves $L(q, \dot{q})$ locally invariant: $[1 \quad 0]^T$

2) Test local invariance:

$$L = \frac{1}{2} M \dot{x}_m^2 - mgR\cos\theta + \frac{1}{2} mR^2\dot{\theta}^2$$
$$+ mR\cos(\theta)\dot{\theta}\dot{x}_m + \frac{1}{2} m\dot{x}_m^2$$

$$\frac{dL}{d\dot{x}} = (M+m)\dot{x}_m + mR\cos(\theta)\dot{\theta}$$

$$\frac{dL}{d\dot{\theta}} = mR^2\dot{\theta} + mR\cos\theta(\dot{x}_m)$$

3) 2. $q_\varepsilon = q + \varepsilon G(q)$

$$= \begin{bmatrix} x_m \\ \theta \end{bmatrix} + \varepsilon \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x + \varepsilon \\ \theta \end{bmatrix}$$

$L_\varepsilon = L(q_\varepsilon, \dot{q}_\varepsilon)$

$$L_\varepsilon = \frac{1}{2} M \dot{x}_m^2 - mgR\cos\theta + \frac{1}{2} m R^2 \dot{\theta}^2 + mR\cos(\theta)\dot{\theta}\dot{x}_m + \frac{1}{2} m \dot{x}_m^2 \rightarrow L_\varepsilon - L = 0$$

The system is both locally and globally invariant due to $G(q)$ as the Lagrangian does not depend on ε.

3. Find conserved quantity P_{cons}:

$$P_{cons} = \frac{\partial L}{\partial \dot{q}} G(q)$$

$$= \begin{bmatrix} (M+m)\dot{x}_m + mR\cos(\theta)\dot{\theta} \\ mR^2\dot{\theta} + mR\cos\theta(\dot{x}_m) \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\boxed{P_{cons} = (M+m)\dot{x}_m + mR\cos(\theta)\dot{\theta}}$$