Davidk_hw5_FA2024

November 6, 2024

1 ME314 Homework 5

1.0.1 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 editing
- 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.

[2]:
```
#Import cell
import sympy as sym
import numpy as np
```

[3]:
```
#############################################################################################
# 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)
```

Below are the help functions in previous homeworks, which you may need for this homework.

[4]:
```
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 animate_double_pend(theta_array,L1=1,L2=1,T=10):
    """
    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.
from plotly.offline import init_notebook_mode, iplot
from IPython.display import display, HTML, Markdown
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-latest.min.js?noext',
            },
          });
        </script>
        '''))
configure_plotly_browser_state()
init_notebook_mode(connected=False)

###############################################
# Getting data from pendulum angle trajectories.
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

####################################
# Using these to specify axis limits.
xm=np.min(xx1)-0.5
xM=np.max(xx1)+0.5
ym=np.min(yy1)-2.5
yM=np.max(yy1)+1.5

###########################
# Defining data dictionary.
# 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=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)
            ),
      ]

   ################################
   # 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='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.
   # This is what draws the lines from
   # joint to joint of the pendulum.
   frames=[dict(data=[dict(x=[0,xx1[k],xx2[k]],
                           y=[0,yy1[k],yy2[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)),
                      ]) for k in range(N)]

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

def animate_triple_pend(theta_array, L1=1, L2=1, L3=1, T=10):
    """
    Function to generate web-based animation of triple-pendulum system

    Parameters:
    ================================================
    theta_array:
        trajectory of theta1 and theta2, should be a NumPy array with
        shape of (3,N)
    L1:
        length of the first pendulum
    L2:
        length of the second pendulum
    L3:
        length of the third 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

    #######################
```

```python
# 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.
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])
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])
N = len(theta_array[0]) # Need this for specifying length of simulation

####################################
# Using these to specify axis limits.
xm=np.min(xx1)-0.5
xM=np.max(xx1)+0.5
ym=np.min(yy1)-2.5
yM=np.max(yy1)+1.5

###########################
# Defining data dictionary.
# 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="yellow", 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='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.
    # 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)
```

```
<>:2: SyntaxWarning: invalid escape sequence '\d'
<>:31: SyntaxWarning: invalid escape sequence '\d'
<>:2: SyntaxWarning: invalid escape sequence '\d'
<>:31: SyntaxWarning: invalid escape sequence '\d'
/tmp/ipykernel_208833/63316585.py:2: SyntaxWarning: invalid escape sequence '\d'
  """
/tmp/ipykernel_208833/63316585.py:31: SyntaxWarning: invalid escape sequence
'\d'
  """
```

[5]:
```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/
  ↪raw/master/singlepend.JPG' width=350' height='350'></table>"))
```

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

1.1 Problem 1 (5pts)

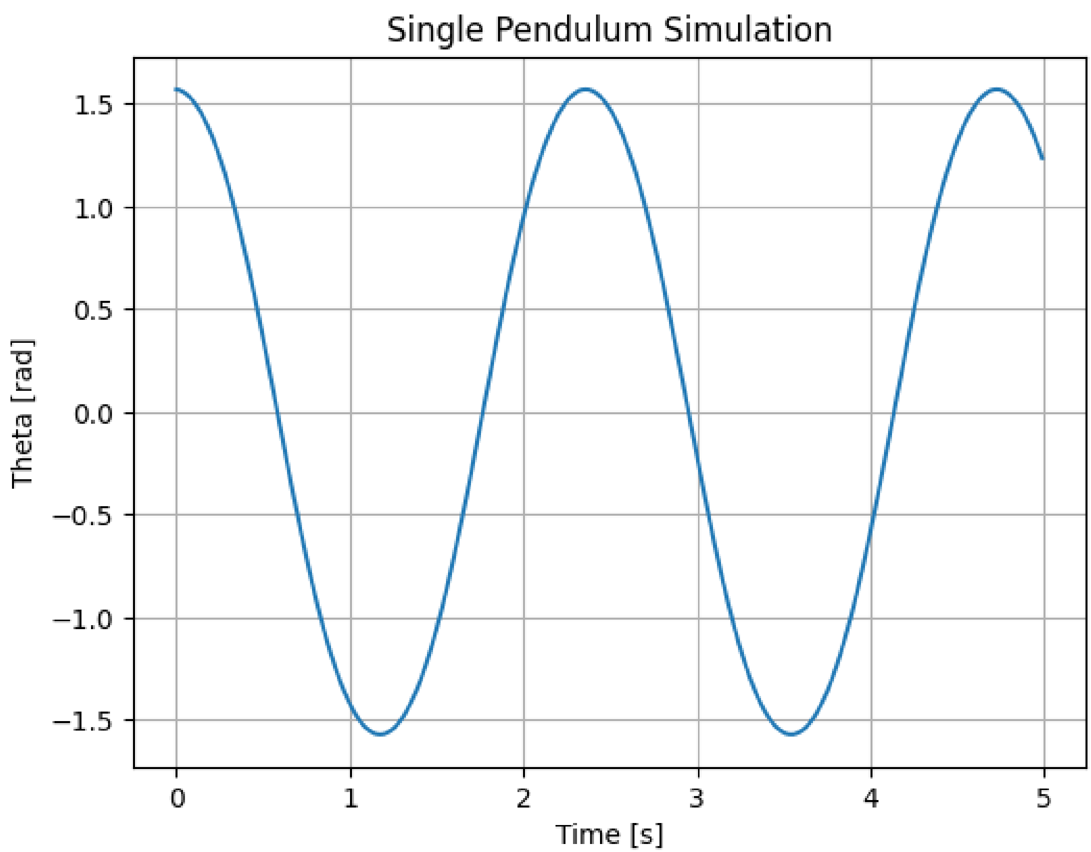
Consider the single pendulum showed above. Solve the Euler-Lagrange equations and simulate the system for $t \in [0, 5]$ with $dt = 0.01, R = 1, m = 1, g = 9.8$ given initial condition as $\theta = \frac{\pi}{2}, \dot{\theta} = 0$. Plot your simulation of the system (i.e. θ versus time). Note that in this problem there is no impact involved (ignore the wall at the bottom).

Turn in: A copy of the code used to solve the EL-equations and numerically simulate

the system. Also include code output, which should be the plot of the trajectory versus time.

[6]:
```
# Define symbols
t = sym.symbols('t')
theta = sym.Function('theta')(t)
R, m, g = sym.symbols('R m g')
theta_dot = sym.diff(theta, t)

# Define Lagrangian
T = 0.5 * m * (R * (sym.diff(theta)))**2  # Kinetic energy
V = m * g * R * (1 - sym.cos(theta))        # Potential energy
L = T - V                                   # Lagrangian
# Euler-Lagrange equation
dL_dtheta = sym.diff(L, theta)
dL_dtheta_dot = sym.diff(L, sym.diff(theta, t))
d_dt_dL_dtheta_dot = sym.diff(dL_dtheta_dot, t)
EL_eq = sym.Eq(d_dt_dL_dtheta_dot - dL_dtheta, 0)

# Solve Euler-Lagrange equation for theta''(t)
theta_ddot = sym.solve(EL_eq, sym.diff(theta, (t, 2)))[0]

# Convert to numerical function
theta_ddot = theta_ddot.subs({R: 1, m: 1, g: 9.8}).simplify()
theta_ddot_func = sym.lambdify([theta, theta_dot], theta_ddot)

# Parameters
dt = 0.01
tspan = [0, 5]
theta0 = np.pi / 2
theta_dot0 = 0.0

# Initial conditions
x0 = np.array([theta0, theta_dot0])

# Define the system of ODEs
def pendulum_ode(x):
    theta, theta_dot = x
    theta_ddot = theta_ddot_func(theta, theta_dot)
    return np.array([theta_dot, theta_ddot])

# Time vector
tvec = np.arange(tspan[0], tspan[1], dt)

sol = simulate(pendulum_ode, x0, tspan, dt, integrate)

# Integrate the ODE
```

```python
# Plot the results
import matplotlib.pyplot as plt

plt.plot(tvec, sol[0, :])
plt.xlabel('Time [s]')
plt.ylabel('Theta [rad]')
plt.title('Single Pendulum Simulation')
plt.grid(True)
plt.show()
```



1.2 Problem 2 (10pts)

Now, time for impact (i.e. don't ignore the vertical wall) ! As shown in the figure above, there is a wall such that the pendulum will hit it when $\theta = 0$. Recall that in the course notes, to solve the impact update rule, we have two set of equations:

$$\frac{\partial L}{\partial \dot{q}}\bigg|_{\tau^-}^{\tau^+} = \lambda \frac{\partial \phi}{\partial q}$$

$$\left[\frac{\partial L}{\partial \dot{q}} \cdot \dot{q} - L(q, \dot{q})\right]\Bigg|_{\tau^-}^{\tau^+} = 0$$

In this problem, you will need to symbolically compute the following three expressions contained the equations above:

$$\frac{\partial L}{\partial \dot{q}}, \quad \frac{\partial \phi}{\partial q}, \quad \frac{\partial L}{\partial \dot{q}} \cdot \dot{q} - L(q, \dot{q})$$

> Hint 1: The third expression is the Hamiltonian of the system.
>
> Hint 2: All three expressions can be considered as functions of q and $\dot{q}$. If you have previously defined q and $\dot{q}$ as SymPy's function objects, now you will need to substitute them with dummy symbols (using SymPy's substitute method)
>
> Hint 3: q and $\dot{q}$ should be two sets of separate symbols.

Turn in: A copy of code used to symbolically compute the three expressions, also include the outputs of your code, which should be the three expressions (make sure there is no SymPy Function(t) left in your solution output).

[7]:
```
q = sym.Function('q')(t)
q_dot = sym.diff(q, t)

L_q = L.subs({theta: q, theta_dot: q_dot})

dL_dq_dot = sym.diff(L_q, q_dot)

# since phi = q, dphi_dq = 1
dphi_dq = sym.diff(q, q)

Hamiltonian_expr = dL_dq_dot * q_dot - L_q

display(dL_dq_dot.simplify())
display(dphi_dq.simplify())
display(sym.Eq(sym.symbols('H'), Hamiltonian_expr))
```

$$1.0R^2m\frac{d}{dt}q(t)$$

1

$$H = 0.5R^2m\left(\frac{d}{dt}q(t)\right)^2 + Rgm\left(1 - \cos\left(q(t)\right)\right)$$

1.3 Problem 3 (10pts)

Now everything is ready for you to solve the impact update rules! To solve those equations, you will need to evaluate them right before and after the impact time at τ^- and τ^+.

> Hint 1: Here $\dot{q}(\tau^-)$ is actually same as the dummy symbol you defined in Problem 2 (why?), but you will need to define new dummy symbol for $\dot{q}(\tau^+)$. That is to say, $\frac{\partial L}{\partial \dot{q}}$ and $\frac{\partial L}{\partial \dot{q}} \cdot \dot{q} - L(q, \dot{q})$ evaluated at τ^- are those you already had in Problem 2, but you will need to substitute the dummy symbols of $\dot{q}(\tau^+)$ to evaluate them at τ^+.

Based on the information above, define the equations for impact update and solve them for impact update rules. After solving the impact update solution, numerically evaluate it as a function using

SymPy's lambdify method and test it with $\theta(\tau^-) = 0.01, \dot{\theta}(\tau^-) = 2$.

> Hint 2: In your equations and impact update solutions, there should be NO SymPy Function left (except for internal functions like sin or cos).
>
> Hint 3: You may wonder where are $q(\tau^-)$ and $q(\tau^+)$? The real question at hand is do we really need new dummy variables for them?
>
> Hint 4: The solution of the impact update rules, which is obtained by solving the equations for the dummy variables corresponds to $\dot{q}(\tau^+)$ and λ, can be a function of $q(\tau^-)$ or a function of $q(\tau^-)$ and $\dot{q}(\tau^-)$. While q will not be updated during impact, including it now (as an argument in your lambdify function) may help you to combine the function into simulation later.

Turn in: A copy of code used to symbolically solve for the impact update rules and evaluate them numerically. Also, include the outputs of your code, which should be the test output of your numerically evaluated impact update function.

[8]:
```
# create the q to be a matrix
q = sym.Matrix([theta])
q_dot = sym.Matrix([theta_dot])

q_plus, q_minus = sym.symbols(r'q^+ q^-')
q_dot_plus, q_dot_minus = sym.symbols(r'\dot{q}^+ \dot{q}^-')

subs_minus = {q[0]: q_minus, q_dot[0]: q_dot_minus}
subs_plus = {q[0]: q_minus, q_dot[0]: q_dot_plus}

p_minus = dL_dtheta_dot.subs(subs_minus)
p_plus = dL_dtheta_dot.subs(subs_plus)

lamb = sym.symbols('lambda')
momentum_eq = sym.Eq((p_plus - p_minus), lamb * dphi_dq)
Hamiltonian = dL_dtheta_dot * theta_dot - L


Hamiltonian_eq_plus = Hamiltonian.subs(subs_plus)
Hamiltonian_eq_minus = Hamiltonian.subs(subs_minus)

Hamiltonian = sym.Eq(Hamiltonian_eq_plus - Hamiltonian_eq_minus, 0)

display(Hamiltonian_eq_plus - Hamiltonian_eq_minus)

# solve for q_dot_plus and lambda
solutions = sym.solve([momentum_eq, Hamiltonian], [q_dot_plus, lamb])

print("Symbolic Solutions")
# display the solutions
display(sym.Eq(q_dot_plus, solutions[0][0]))
```

```python
display(sym.Eq(lamb, solutions[0][1]))
print()
print()
print()
# SUBSTITUTION numerical values

# Define the numerical values
R_val = 1
m_val = 1
g_val = 9.8
theta0_val = 0.01
theta_dot0_val = 2.0

# Substitute the numerical values
subs = {R: R_val, m: m_val, g: g_val, q_dot_minus: theta_dot0_val, q_minus:␣
↪theta0_val}

# Substitute the numerical values
q_dot_plus_val = solutions[0][0].subs(subs)
lambda_val = solutions[0][1].subs(subs)

print("Numerical Solutions")
display(sym.Eq(q_dot_plus, q_dot_plus_val))
display(sym.Eq(lamb, lambda_val))

qdot_lambdified = sym.lambdify([q_minus,q_dot_minus],q_dot_plus_val)
lam_lambdified = sym.lambdify([q_minus,q_dot_minus],lambda_val)

def impact_update(q, restitution=1.0):
    """
    Update the state `q` upon impact with a restitution factor.
    Reflects `thetadot` and applies restitution.
    """
    q[1] = -restitution * q[1]  # Reverse thetadot with restitution factor
    return q

s_test = np.array([0.01,2])
display(impact_update(x0))
```

$0.5R^2\left(\dot{q}^+\right)^2 m - 0.5R^2\left(\dot{q}^-\right)^2 m$

```
Symbolic Solutions
```

$\dot{q}^+ = -\dot{q}^-$

$\lambda = -2.0R^2\dot{q}^- m$

```
Numerical Solutions
```

$\dot{q}^{+} = -2.0$

$\lambda = -4.0$

```
array([ 1.57079633, -0.        ])
```

1.4 Problem 4 (20pts)

Finally, it's time to simulate the impact! To use impact update rules with our previous simulate function, there two more steps: 1. Write a function called ''impact_condition", which takes in $s = [q, \dot{q}]$ and returns **True** if s will cause an impact, otherwise the function will return **False**.

> Hint 1 : you need to use the constraint ϕ in this problem, and note that, since we are doing numerical evaluation, the impact condition will not be perfect, you will need to catch the change of sign at $\phi(s)$ or setup a threshold to decide the condition.

2. Now, with the ''impact_condition" function and the numerically evaluated impact update rule for $\dot{q}(\tau^{+})$ solved in last problem, find a way to combine them into the previous simulation function, thus it can simulate the impact. Pseudo-code for the simulate function can be found in lecture note 13.

Simulate the system with same parameters and initial condition in Problem 1 for the single pendulum hitting the wall for five times. Plot the trajectory and animate the simulation (you need to modify the animation function by yourself).

Turn in: A copy of the code used to simulate the system. You don't need to include the animation function, but please include other code (impact_condition, simulate, ets.) used for simulating impact. Also, include the plot and a video for animation. 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.

[9]:
```
phi = sym.sin(theta)
phi_func = sym.lambdify([theta],phi)

def impact_condition(q, threshold=1e-1):
    """
    Check if the pendulum is in an impact state, indicating a collision with the wall.
    Returns True if an impact occurs, otherwise False.
    """
    phi_val = phi_func(q[0])
    return -threshold < phi_val < threshold

def simulate_impact(f, x0, tspan, dt, integrate, max_impacts=5):
    """
    Simulate the pendulum system with impacts at the wall.

    Parameters:
```

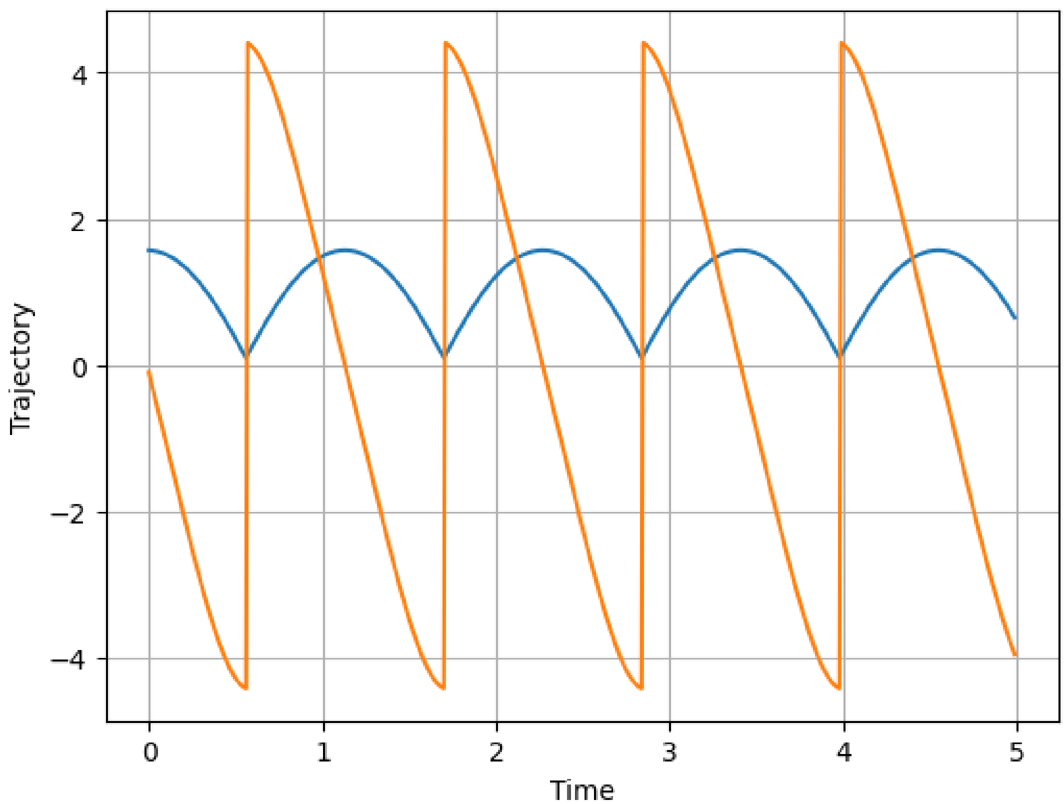
```python
        f: The ODE function representing the pendulum dynamics.
        x0: Initial state vector [theta, thetadot].
        tspan: Time span [t0, tf] for the simulation.
        dt: Time step for integration.
        integrate: Function to perform numerical integration (e.g., Euler).
        max_impacts: Maximum number of impacts to simulate.

    Returns:
        xtraj: Trajectory of the system.
    """
    N = int((max(tspan) - min(tspan)) / dt)
    x = np.copy(x0)
    xtraj = np.zeros((len(x0), N))
    impact_count = 0
    # Simulation loop
    for i in range(N):
        # Check for impact condition
        if impact_condition(x) and impact_count < max_impacts:
            x = impact_update(x)
            impact_count += 1
        xtraj[:, i] = integrate(f, x, dt)  # Integrate to find the next state
        x = np.copy(xtraj[:, i])  # Update current state

    return xtraj

# Generate a trajectory
dt = 0.01
traj = simulate_impact(pendulum_ode, x0, [0, 5], dt, integrate)
print('shape of traj: ', traj.shape)

# Plotting
time_steps = np.arange(traj.shape[1]) * dt
plt.plot(time_steps, traj[0:2].T)
plt.xlabel('Time')
plt.ylabel('Trajectory')
plt.grid(True)
plt.show()

def animate_single_pend(theta_array,L1=1,T=10):
    """
    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

    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, Markdown
    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])
    N = len(theta_array[0]) # Need this for specifying length of simulation

    ####################################
    # Using these to specify axis limits. (this needs to be adjusted too)
    xm=np.min(xx1)-0.5
    xM=np.max(xx1)+0.5
    ym=np.min(yy1)-2.5
    yM=np.max(yy1)+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=xx1, y=yy1,
           mode='markers', name='Pendulum 1 Traj',
           marker=dict(color="purple", 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),
            yaxis=dict(range=[ym, yM], autorange=False,␣
↪zeroline=False,scaleanchor = "x",dtick=1),
            title='Single 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]],
                        y=[0,yy1[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)

print('shape of trajectory: ', traj.shape)
animate_single_pend(traj,L1=1,T=10)
```

```
shape of traj:  (2, 500)
```




```
shape of trajectory:  (2, 500)
```

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

1.5 Problem 5 (10pts)

```
[10]: from IPython.core.display import HTML,  Markdown
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/
 ↪raw/master/tripend_constrained.JPG' width=500' height='450'></table>"))
```

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

We will now consider a constrained triple-pendulum system with the system configuration $q = [\theta_1, \theta_2, \theta_3]$. A constraint is such that *x coordinate* of the third pendulum (i.e. m_3) ONLY can not be smaller than 0 – there exist a vertical wall high enough for third pendulum impact. Note that there is no constraint on *y coordinate* – the top ceiling is infinitely high!

Similar to Problem 2, symbolically compute the following three expressions contained the equations above:

$$\frac{\partial L}{\partial \dot{q}}, \quad \frac{\partial \phi}{\partial q}, \quad \frac{\partial L}{\partial \dot{q}} \cdot \dot{q} - L(q, \dot{q})$$

Use $m_1 = m_2 = m_3 = 1$ and $R_1 = R_2 = R_3 = 1$ as numerical values in the equations (i.e. **do not** define $m_1, m_2, m_3, R_1, R_2, R_3$ as symbols).

> Hint 1: As before, you will need to substitute q and $\dot{q}$ with dummy symbols.

Turn in: Include the code used to symbolically compute the three expressions, as well as code outputs - the resulting three expressions. Make sure there is no SymPy Function(t) left!

```
[11]: t, R1, R2, R3, m1, m2, m3, g = sym.symbols('t R1 R2 R3 m1 m2 m3 g')
theta1 = sym.Function('theta1')(t)
theta2 = sym.Function('theta2')(t)
theta3 = sym.Function('theta3')(t)

q = sym.Matrix([theta1, theta2, theta3])
q_dot = q.diff(t)
q_ddot = q_dot.diff(t)

# Define the positions of the masses
x1 = R1 * sym.sin(theta1)
y1 = -R1 * sym.cos(theta1)
x2 = x1 + R2 * sym.sin(theta1 + theta2)
y2 = y1 - R2 * sym.cos(theta1 + theta2)
x3 = x2 + R3 * sym.sin(theta1 + theta2 + theta3)
y3 = y2 - R3 * sym.cos(theta1 + theta2 + theta3)

# Define the velocities of the masses
x1_dot = x1.diff(t)
y1_dot = y1.diff(t)
x2_dot = x2.diff(t)
y2_dot = y2.diff(t)
x3_dot = x3.diff(t)
```

```python
y3_dot = y3.diff(t)

# Define the kinetic energy
T1 = 0.5 * m1 * (x1_dot**2 + y1_dot**2)
T2 = 0.5 * m2 * (x2_dot**2 + y2_dot**2)
T3 = 0.5 * m3 * (x3_dot**2 + y3_dot**2)

T = T1 + T2 + T3

# Define the potential energy
V1 = m1 * g * y1
V2 = m2 * g * y2
V3 = m3 * g * y3

V = V1 + V2 + V3

# Define the Lagrangian
L = T - V

# Define the Euler-Lagrange equation
EL_eq = sym.diff(L, q) - sym.diff(sym.diff(L, q_dot), t)

# substitute the numerical values
subs = {R1: 1, R2: 1, R3: 1, m1: 1, m2: 1, m3: 1, g: 9.8}

# substitute the numerical values
EL_eq = EL_eq.subs(subs).simplify()

# solve for q_ddot
q_ddot_sol = sym.solve(EL_eq, q_ddot, dict=True)[0]
```

[12]:
```python
theta1_sol = q_ddot_sol[theta1.diff(t, 2)]
theta2_sol = q_ddot_sol[theta2.diff(t, 2)]
theta3_sol = q_ddot_sol[theta3.diff(t, 2)]

# convert to numerical function
theta1_func = sym.lambdify([theta1, theta2, theta3, theta1.diff(t), theta2.
 ↪diff(t), theta3.diff(t)], theta1_sol)
theta2_func = sym.lambdify([theta1, theta2, theta3, theta1.diff(t), theta2.
 ↪diff(t), theta3.diff(t)], theta2_sol)
theta3_func = sym.lambdify([theta1, theta2, theta3, theta1.diff(t), theta2.
 ↪diff(t), theta3.diff(t)], theta3_sol)

# define the system of ODEs
def triple_pendulum_ode(x):
    return np.
 ↪array([x[3],x[4],x[5],theta1_func([*x]),theta2_func([*x]),theta3_func([*x])])
```

2 Compute DL/dq_dot, dphi/dq, and hamiltonian

[13]:
```
# Compute DL/Dq_dot

# Compute dphi/dq
phi = x3

# sub phi
phi = phi.subs({m1: 1, m2: 1, m3: 1, R1: 1, R2: 1, R3: 1, g: 9.8})

# Compute dphi/dq
phi_func = sym.lambdify([theta1, theta2, theta3, q_dot[0], q_dot[1], q_dot[2]],␣
↪phi)

# Compute dphi/dq
dphi_dq = sym.Matrix([phi]).jacobian(q)

theta1, theta2, theta3 = sym.symbols(r'\theta_1 \theta_2 \theta_3')
theta1_dot, theta2_dot, theta3_dot = sym.symbols(r'\dot{\theta}_1␣
↪\dot{\theta}_2 \dot{\theta}_3')

dL_dq_dot = sym.diff(L, q_dot)
dL_dq_dot = dL_dq_dot.subs({
    # sub theta as a function of time to just a plain symbol
    sym.Function('theta1')(t): theta1,
    sym.Function('theta2')(t): theta2,
    sym.Function('theta3')(t): theta3,
    sym.diff(sym.Function('theta1')(t), t): theta1_dot,
    sym.diff(sym.Function('theta2')(t), t): theta2_dot,
    sym.diff(sym.Function('theta3')(t), t): theta3_dot
})

dphi_dq = dphi_dq.subs({
    # sub theta as a function of time to just a plain symbol
    sym.Function('theta1')(t): theta1,
    sym.Function('theta2')(t): theta2,
    sym.Function('theta3')(t): theta3,
    sym.diff(sym.Function('theta1')(t), t): theta1_dot,
    sym.diff(sym.Function('theta2')(t), t): theta2_dot,
    sym.diff(sym.Function('theta3')(t), t): theta3_dot
})

Hamiltonian_expr = sym.Matrix([dL_dq_dot]).dot(q_dot) - L

Hamiltonian_pr2 = Hamiltonian_expr.subs({
    # sub theta as a function of time to just a plain symbol
    sym.Function('theta1')(t): theta1,
```

```python
    sym.Function('theta2')(t): theta2,
    sym.Function('theta3')(t): theta3,
    sym.diff(sym.Function('theta1')(t), t): theta1_dot,
    sym.diff(sym.Function('theta2')(t), t): theta2_dot,
    sym.diff(sym.Function('theta3')(t), t): theta3_dot
})

#display(Markdown('$\\frac{\\partial L}{\\partial \\dot{q}} = $'))

display(Markdown('$Value$ $of$ $\\frac{\partial L}{ \partial \\dot{q}}$'))
display(Markdown(f'$\\frac{{\\partial L}}{{\\partial \\dot{{\\theta_{1}}}}} = {dL_dq_dot[0].expand()}$'))
display(Markdown(f'$\\frac{{\\partial L}}{{\\partial \\dot{{\\theta_{2}}}}} = {dL_dq_dot[1].expand()}$'))
display(Markdown(f'$\\frac{{\\partial L}}{{\\partial \\dot{{\\theta_{3}}}}} = {dL_dq_dot[2].expand()}$'))

display(Markdown('$Value$ $of$ $\\frac{\partial \phi}{ \partial q}$'))
display(Markdown(f'$\\frac{{\\partial \\phi}}{{\\partial \\theta_{1}}} = {dphi_dq[0].expand()}$'))
display(Markdown(f'$\\frac{{\\partial \\phi}}{{\\partial \\theta_{2}}} = {dphi_dq[1].expand()}$'))
display(Markdown(f'$\\frac{{\\partial \\phi}}{{\\partial \\theta_{3}}} = {dphi_dq[2].expand()}$'))

display(Markdown('$Hamiltonian:$'))
display(Markdown(f'$H = {sym.latex(Hamiltonian_pr2.expand())}$'))
```

```
<>:53: SyntaxWarning:

invalid escape sequence '\p'

<>:58: SyntaxWarning:

invalid escape sequence '\p'

<>:53: SyntaxWarning:

invalid escape sequence '\p'

<>:58: SyntaxWarning:

invalid escape sequence '\p'

/tmp/ipykernel_208833/3294819495.py:53: SyntaxWarning:

invalid escape sequence '\p'
```

```
/tmp/ipykernel_208833/3294819495.py:58: SyntaxWarning:

invalid escape sequence '\p'
```

$Value\ of\ \frac{\partial L}{\partial \dot{q}}$

$\frac{\partial L}{\partial \dot{\theta_1}} = 1.0*R1**2*\dot{\theta}_1*m1*sin(\theta_1)**2+1.0*R1**2*\dot{\theta}_1*m1*cos(\theta_1)**2+1.0*R1**2*\dot{\theta}_1*m2*sin(\theta_1)**2+1.0*R1**2*\dot{\theta}_1*m2*cos(\theta_1)**2+1.0*R1**2*\dot{\theta}_1*m3*sin(\theta_1)**2+1.0*R1**2*\dot{\theta}_1*m3*cos(\theta_1)**2+2.0*R1*R2*\dot{\theta}_1*m2*sin(\theta_1)*sin(\theta_1+\theta_2)+2.0*R1*R2*\dot{\theta}_1*m2*cos(\theta_1)*cos(\theta_1+\theta_2)+2.0*R1*R2*\dot{\theta}_1*m3*sin(\theta_1)*sin(\theta_1+\theta_2)+2.0*R1*R2*\dot{\theta}_1*m3*cos(\theta_1)*cos(\theta_1+\theta_2)+1.0*R1*R2*\dot{\theta}_2*m2*sin(\theta_1)*sin(\theta_1+\theta_2)+1.0*R1*R2*\dot{\theta}_2*m2*cos(\theta_1)*cos(\theta_1+\theta_2)+1.0*R1*R2*\dot{\theta}_2*m3*sin(\theta_1)*sin(\theta_1+\theta_2)+1.0*R1*R2*\dot{\theta}_2*m3*cos(\theta_1)*cos(\theta_1+\theta_2)+2.0*R1*R3*\dot{\theta}_1*m3*sin(\theta_1)*sin(\theta_1+\theta_2+\theta_3)+2.0*R1*R3*\dot{\theta}_1*m3*cos(\theta_1)*cos(\theta_1+\theta_2+\theta_3)+1.0*R1*R3*\dot{\theta}_2*m3*sin(\theta_1)*sin(\theta_1+\theta_2+\theta_3)+1.0*R1*R3*\dot{\theta}_2*m3*cos(\theta_1)*cos(\theta_1+\theta_2+\theta_3)+1.0*R1*R3*\dot{\theta}_3*m3*sin(\theta_1)*sin(\theta_1+\theta_2+\theta_3)+1.0*R1*R3*\dot{\theta}_3*m3*cos(\theta_1)*cos(\theta_1+\theta_2+\theta_3)+1.0*R2**2*\dot{\theta}_1*m2*sin(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_1*m2*cos(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_1*m3*sin(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_1*m3*cos(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_2*m2*sin(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_2*m2*cos(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_2*m3*sin(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_2*m3*cos(\theta_1+\theta_2)**2+2.0*R2*R3*\dot{\theta}_1*m3*sin(\theta_1+\theta_2)*sin(\theta_1+\theta_2+\theta_3)+2.0*R2*R3*\dot{\theta}_1*m3*cos(\theta_1+\theta_2)*cos(\theta_1+\theta_2+\theta_3)+2.0*R2*R3*\dot{\theta}_2*m3*sin(\theta_1+\theta_2)*sin(\theta_1+\theta_2+\theta_3)+2.0*R2*R3*\dot{\theta}_2*m3*cos(\theta_1+\theta_2)*cos(\theta_1+\theta_2+\theta_3)+1.0*R2*R3*\dot{\theta}_3*m3*sin(\theta_1+\theta_2)*sin(\theta_1+\theta_2+\theta_3)+1.0*R2*R3*\dot{\theta}_3*m3*cos(\theta_1+\theta_2)*cos(\theta_1+\theta_2+\theta_3)+1.0*R3**2*\dot{\theta}_1*m3*sin(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_1*m3*cos(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_2*m3*sin(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_2*m3*cos(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_3*m3*sin(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_3*m3*cos(\theta_1+\theta_2+\theta_3)**2$

$\frac{\partial L}{\partial \dot{\theta_2}} = 1.0*R1*R2*\dot{\theta}_1*m2*sin(\theta_1)*sin(\theta_1+\theta_2)+1.0*R1*R2*\dot{\theta}_1*m2*cos(\theta_1)*cos(\theta_1+\theta_2)+1.0*R1*R2*\dot{\theta}_1*m3*sin(\theta_1)*sin(\theta_1+\theta_2)+1.0*R1*R2*\dot{\theta}_1*m3*cos(\theta_1)*cos(\theta_1+\theta_2)+1.0*R1*R3*\dot{\theta}_1*m3*sin(\theta_1)*sin(\theta_1+\theta_2+\theta_3)+1.0*R1*R3*\dot{\theta}_1*m3*cos(\theta_1)*cos(\theta_1+\theta_2+\theta_3)+1.0*R2**2*\dot{\theta}_1*m2*sin(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_1*m2*cos(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_1*m3*sin(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_1*m3*cos(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_2*m2*sin(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_2*m2*cos(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_2*m3*sin(\theta_1+\theta_2)**2+1.0*R2**2*\dot{\theta}_2*m3*cos(\theta_1+\theta_2)**2+2.0*R2*R3*\dot{\theta}_1*m3*sin(\theta_1+\theta_2)*sin(\theta_1+\theta_2+\theta_3)+2.0*R2*R3*\dot{\theta}_1*m3*cos(\theta_1+\theta_2)*cos(\theta_1+\theta_2+\theta_3)+2.0*R2*R3*\dot{\theta}_2*m3*sin(\theta_1+\theta_2)*sin(\theta_1+\theta_2+\theta_3)+2.0*R2*R3*\dot{\theta}_2*m3*cos(\theta_1+\theta_2)*cos(\theta_1+\theta_2+\theta_3)+1.0*R2*R3*\dot{\theta}_3*m3*sin(\theta_1+\theta_2)*sin(\theta_1+\theta_2+\theta_3)+1.0*R2*R3*\dot{\theta}_3*m3*cos(\theta_1+\theta_2)*cos(\theta_1+\theta_2+\theta_3)+1.0*R3**2*\dot{\theta}_1*m3*sin(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_1*m3*cos(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_2*m3*sin(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_2*m3*cos(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_3*m3*sin(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_3*m3*cos(\theta_1+\theta_2+\theta_3)**2$

$\frac{\partial L}{\partial \dot{\theta_3}} = 1.0*R1*R3*\dot{\theta}_1*m3*sin(\theta_1)*sin(\theta_1+\theta_2+\theta_3)+1.0*R1*R3*\dot{\theta}_1*m3*cos(\theta_1)*cos(\theta_1+\theta_2+\theta_3)+1.0*R2*R3*\dot{\theta}_1*m3*sin(\theta_1+\theta_2)*sin(\theta_1+\theta_2+\theta_3)+1.0*R2*R3*\dot{\theta}_1*m3*cos(\theta_1+\theta_2)*cos(\theta_1+\theta_2+\theta_3)+1.0*R2*R3*\dot{\theta}_2*m3*sin(\theta_1+\theta_2)*sin(\theta_1+\theta_2+\theta_3)+1.0*R2*R3*\dot{\theta}_2*m3*cos(\theta_1+\theta_2)*cos(\theta_1+\theta_2+\theta_3)+1.0*R3**2*\dot{\theta}_1*m3*sin(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_1*m3*cos(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_2*m3*sin(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_2*m3*cos(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_3*m3*sin(\theta_1+\theta_2+\theta_3)**2+1.0*R3**2*\dot{\theta}_3*m3*cos(\theta_1+\theta_2+\theta_3)**2$

$Value\ of\ \frac{\partial \phi}{\partial q}$

$\frac{\partial \phi}{\partial \theta_1} = cos(\theta_1) + cos(\theta_1+\theta_2) + cos(\theta_1+\theta_2+\theta_3)$

$\frac{\partial \phi}{\partial \theta_2} = cos(\theta_1 + \theta_2) + cos(\theta_1 + \theta_2 + \theta_3)$

$\frac{\partial \phi}{\partial \theta_3} = cos(\theta_1 + \theta_2 + \theta_3)$

$Hamiltonian:$

$$H = 0.5R_1^2\dot{\theta}_1^2 m_1 \sin^2(\theta_1) + 0.5R_1^2\dot{\theta}_1^2 m_1 \cos^2(\theta_1) + 0.5R_1^2\dot{\theta}_1^2 m_2 \sin^2(\theta_1) + 0.5R_1^2\dot{\theta}_1^2 m_2 \cos^2(\theta_1) + 0.5R_1^2\dot{\theta}_1^2 m_3 \sin^2(\theta_1) + 0.5R_1^2\dot{\theta}_1^2 m_3 \cos^2(\theta_1) + 1.0R_1R_2\dot{\theta}_1^2 m_2 \sin(\theta_1)\sin(\theta_1+\theta_2) + 1.0R_1R_2\dot{\theta}_1^2 m_2 \cos(\theta_1)\cos(\theta_1+\theta_2) + 1.0R_1R_2\dot{\theta}_1^2 m_3 \sin(\theta_1)\sin(\theta_1+\theta_2) + 1.0R_1R_2\dot{\theta}_1^2 m_3 \cos(\theta_1)\cos(\theta_1+\theta_2) + 1.0R_1R_2\dot{\theta}_1\dot{\theta}_2 m_2 \sin(\theta_1)\sin(\theta_1+\theta_2) + 1.0R_1R_2\dot{\theta}_1\dot{\theta}_2 m_2 \cos(\theta_1)\cos(\theta_1+\theta_2) + 1.0R_1R_2\dot{\theta}_1\dot{\theta}_2 m_3 \sin(\theta_1)\sin(\theta_1+\theta_2) + 1.0R_1R_2\dot{\theta}_1\dot{\theta}_2 m_3 \cos(\theta_1)\cos(\theta_1+\theta_2) + 1.0R_1R_3\dot{\theta}_1^2 m_3 \sin(\theta_1)\sin(\theta_1+\theta_2+\theta_3) + 1.0R_1R_3\dot{\theta}_1^2 m_3 \cos(\theta_1)\cos(\theta_1+\theta_2+\theta_3) + 1.0R_1R_3\dot{\theta}_1\dot{\theta}_2 m_3 \sin(\theta_1)\sin(\theta_1+\theta_2+\theta_3) + 1.0R_1R_3\dot{\theta}_1\dot{\theta}_2 m_3 \cos(\theta_1)\cos(\theta_1+\theta_2+\theta_3) + 1.0R_1R_3\dot{\theta}_1\dot{\theta}_3 m_3 \sin(\theta_1)\sin(\theta_1+\theta_2+\theta_3) + 1.0R_1R_3\dot{\theta}_1\dot{\theta}_3 m_3 \cos(\theta_1)\cos(\theta_1+\theta_2+\theta_3) - R_1 g m_1 \cos(\theta_1) - R_1 g m_2 \cos(\theta_1) - R_1 g m_3 \cos(\theta_1) + 0.5R_2^2\dot{\theta}_1^2 m_2 \sin^2(\theta_1+\theta_2) + 0.5R_2^2\dot{\theta}_1^2 m_2 \cos^2(\theta_1+\theta_2) + 0.5R_2^2\dot{\theta}_1^2 m_3 \sin^2(\theta_1+\theta_2) + 0.5R_2^2\dot{\theta}_1^2 m_3 \cos^2(\theta_1+\theta_2) + 1.0R_2^2\dot{\theta}_1\dot{\theta}_2 m_2 \sin^2(\theta_1+\theta_2) + 1.0R_2^2\dot{\theta}_1\dot{\theta}_2 m_2 \cos^2(\theta_1+\theta_2) + 1.0R_2^2\dot{\theta}_1\dot{\theta}_2 m_3 \sin^2(\theta_1+\theta_2) + 1.0R_2^2\dot{\theta}_1\dot{\theta}_2 m_3 \cos^2(\theta_1+\theta_2) + 0.5R_2^2\dot{\theta}_2^2 m_2 \sin^2(\theta_1+\theta_2) + 0.5R_2^2\dot{\theta}_2^2 m_2 \cos^2(\theta_1+\theta_2) + 0.5R_2^2\dot{\theta}_2^2 m_3 \sin^2(\theta_1+\theta_2) + 0.5R_2^2\dot{\theta}_2^2 m_3 \cos^2(\theta_1+\theta_2) + 1.0R_2R_3\dot{\theta}_1^2 m_3 \sin(\theta_1+\theta_2)\sin(\theta_1+\theta_2+\theta_3) + 1.0R_2R_3\dot{\theta}_1^2 m_3 \cos(\theta_1+\theta_2)\cos(\theta_1+\theta_2+\theta_3) + 2.0R_2R_3\dot{\theta}_1\dot{\theta}_2 m_3 \sin(\theta_1+\theta_2)\sin(\theta_1+\theta_2+\theta_3) + 2.0R_2R_3\dot{\theta}_1\dot{\theta}_2 m_3 \cos(\theta_1+\theta_2)\cos(\theta_1+\theta_2+\theta_3) + 1.0R_2R_3\dot{\theta}_1\dot{\theta}_3 m_3 \sin(\theta_1+\theta_2)\sin(\theta_1+\theta_2+\theta_3) + 1.0R_2R_3\dot{\theta}_1\dot{\theta}_3 m_3 \cos(\theta_1+\theta_2)\cos(\theta_1+\theta_2+\theta_3) + 1.0R_2R_3\dot{\theta}_2^2 m_3 \sin(\theta_1+\theta_2)\sin(\theta_1+\theta_2+\theta_3) + 1.0R_2R_3\dot{\theta}_2^2 m_3 \cos(\theta_1+\theta_2)\cos(\theta_1+\theta_2+\theta_3) + 1.0R_2R_3\dot{\theta}_2\dot{\theta}_3 m_3 \sin(\theta_1+\theta_2)\sin(\theta_1+\theta_2+\theta_3) + 1.0R_2R_3\dot{\theta}_2\dot{\theta}_3 m_3 \cos(\theta_1+\theta_2)\cos(\theta_1+\theta_2+\theta_3) - R_2 g m_2 \cos(\theta_1+\theta_2) - R_2 g m_3 \cos(\theta_1+\theta_2) + 0.5R_3^2\dot{\theta}_1^2 m_3 \sin^2(\theta_1+\theta_2+\theta_3) + 0.5R_3^2\dot{\theta}_1^2 m_3 \cos^2(\theta_1+\theta_2+\theta_3) + 1.0R_3^2\dot{\theta}_1\dot{\theta}_2 m_3 \sin^2(\theta_1+\theta_2+\theta_3) + 1.0R_3^2\dot{\theta}_1\dot{\theta}_2 m_3 \cos^2(\theta_1+\theta_2+\theta_3) + 1.0R_3^2\dot{\theta}_1\dot{\theta}_3 m_3 \sin^2(\theta_1+\theta_2+\theta_3) + 1.0R_3^2\dot{\theta}_1\dot{\theta}_3 m_3 \cos^2(\theta_1+\theta_2+\theta_3) + 0.5R_3^2\dot{\theta}_2^2 m_3 \sin^2(\theta_1+\theta_2+\theta_3) + 0.5R_3^2\dot{\theta}_2^2 m_3 \cos^2(\theta_1+\theta_2+\theta_3) + 1.0R_3^2\dot{\theta}_2\dot{\theta}_3 m_3 \sin^2(\theta_1+\theta_2+\theta_3) + 1.0R_3^2\dot{\theta}_2\dot{\theta}_3 m_3 \cos^2(\theta_1+\theta_2+\theta_3) + 0.5R_3^2\dot{\theta}_3^2 m_3 \sin^2(\theta_1+\theta_2+\theta_3) + 0.5R_3^2\dot{\theta}_3^2 m_3 \cos^2(\theta_1+\theta_2+\theta_3) - R_3 g m_3 \cos(\theta_1+\theta_2+\theta_3)$$

2.1 Problem 6 (10pts)

Similar to Problem 3, now you need to define dummy symbols for $\dot{q}(\tau^+)$, define the equations for impact update rules. Note that you don't need to solve the equations in this problem - in fact it's very time consuming to solve the analytical solution, and we will use a trick to get around it later!

Turn in: Include a copy of the code used to define the equations for impact update and the code output (i.e. print out of the equations).

[14]:
```
# Use problem 3 to solve for q_dot_plus and lambda

th1_minus, th2_minus, th3_minus = sym.symbols(' _1^-  _2^-  _3^-')

# define theta_dot_plus and theta_dot_minus, make the dot be on the top and not␣
↪a subscipt
th1_dot_plus, th2_dot_plus, th3_dot_plus = sym.symbols(r'\dot{\theta}_1^+␣
↪\dot{\theta}_2^+ \dot{\theta}_3^+')
th1_dot_minus, th2_dot_minus, th3_dot_minus = sym.symbols(r'\dot{\theta}_1^-␣
↪\dot{\theta}_2^- \dot{\theta}_3^-')
```

```python
# no th1_plus, th2_plus, th3_plus because the velocty changes but the position␣
 ↪does not after and before some time step dt
lamb = sym.symbols('lambda')

subs_minus = {theta1: th1_minus, theta2: th2_minus, theta3: th3_minus,
                theta1_dot: th1_dot_minus, theta2_dot: th2_dot_minus,␣
 ↪theta3_dot: th3_dot_minus}

subs_plus = {theta1: th1_minus, theta2: th2_minus, theta3: th3_minus,
                theta1_dot: th1_dot_plus, theta2_dot: th2_dot_plus, theta3_dot:␣
 ↪th3_dot_plus}

p_minus = dL_dq_dot.subs(subs_minus)
p_plus = dL_dq_dot.subs(subs_plus)

# SUB PHI_q
dphi_dq_minus = dphi_dq.subs(subs_minus)

Hamiltonian_eq_plus = Hamiltonian_pr2.subs(subs_plus)
Hamiltonian_eq_minus = Hamiltonian_pr2.subs(subs_minus)

impact_eq_l = sym.Matrix(
    [
        p_plus[0] - p_minus[0],
        p_plus[1] - p_minus[1],
        p_plus[2] - p_minus[2],
        Hamiltonian_eq_plus - Hamiltonian_eq_minus
    ]
)

impact_eq_r = sym.Matrix([
    lamb * dphi_dq_minus[0],
    lamb * dphi_dq_minus[1],
    lamb * dphi_dq_minus[2],
    0
])

impact_eq = sym.Eq(impact_eq_l, impact_eq_r)
impact_eq = impact_eq.subs(subs)


display(Markdown('$Equations$ $of$ $impact$ $update$'))
lhs = impact_eq_l
rhs = impact_eq_r

lhs = lhs.subs(subs)
```

```python
rhs = rhs.subs(subs)

impact_eqs = [
    sym.Eq((p_plus[0] - p_minus[0]).expand(), (lamb * dphi_dq_minus[0]).
 ↪expand()),
    sym.Eq((p_plus[1] - p_minus[1]).expand(), (lamb * dphi_dq_minus[1]).
 ↪expand()),
    sym.Eq((p_plus[2] - p_minus[2]).expand(), (lamb * dphi_dq_minus[2]).
 ↪expand()),
    sym.Eq(Hamiltonian_eq_plus - Hamiltonian_eq_minus, 0)
]

# Display each impact equation
for eq in impact_eqs:
    display(eq.expand())
```

Equations of impact update

$$
1.0R_{1}^{2}\dot{\theta}_{1}^{+}m_{1}\sin^{2}(\theta_{1}^{-}) + 1.0R_{1}^{2}\dot{\theta}_{1}^{+}m_{1}\cos^{2}(\theta_{1}^{-}) + 1.0R_{1}^{2}\dot{\theta}_{1}^{+}m_{2}\sin^{2}(\theta_{1}^{-}) + 1.0R_{1}^{2}\dot{\theta}_{1}^{+}m_{2}\cos^{2}(\theta_{1}^{-}) + 1.0R_{1}^{2}\dot{\theta}_{1}^{+}m_{3}\sin^{2}(\theta_{1}^{-}) + 1.0R_{1}^{2}\dot{\theta}_{1}^{+}m_{3}\cos^{2}(\theta_{1}^{-}) - 1.0R_{1}^{2}\dot{\theta}_{1}^{-}m_{1}\sin^{2}(\theta_{1}^{-}) - 1.0R_{1}^{2}\dot{\theta}_{1}^{-}m_{1}\cos^{2}(\theta_{1}^{-}) - 1.0R_{1}^{2}\dot{\theta}_{1}^{-}m_{2}\sin^{2}(\theta_{1}^{-}) - 1.0R_{1}^{2}\dot{\theta}_{1}^{-}m_{2}\cos^{2}(\theta_{1}^{-}) - 1.0R_{1}^{2}\dot{\theta}_{1}^{-}m_{3}\sin^{2}(\theta_{1}^{-}) - 1.0R_{1}^{2}\dot{\theta}_{1}^{-}m_{3}\cos^{2}(\theta_{1}^{-}) + 2.0R_{1}R_{2}\dot{\theta}_{1}^{+}m_{2}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}) + 2.0R_{1}R_{2}\dot{\theta}_{1}^{+}m_{2}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}) + 2.0R_{1}R_{2}\dot{\theta}_{1}^{+}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}) + 2.0R_{1}R_{2}\dot{\theta}_{1}^{+}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}) - 2.0R_{1}R_{2}\dot{\theta}_{1}^{-}m_{2}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}) - 2.0R_{1}R_{2}\dot{\theta}_{1}^{-}m_{2}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}) - 2.0R_{1}R_{2}\dot{\theta}_{1}^{-}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}) - 2.0R_{1}R_{2}\dot{\theta}_{1}^{-}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{1}R_{2}\dot{\theta}_{2}^{+}m_{2}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{1}R_{2}\dot{\theta}_{2}^{+}m_{2}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{1}R_{2}\dot{\theta}_{2}^{+}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{1}R_{2}\dot{\theta}_{2}^{+}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{1}R_{2}\dot{\theta}_{2}^{-}m_{2}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{1}R_{2}\dot{\theta}_{2}^{-}m_{2}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{1}R_{2}\dot{\theta}_{2}^{-}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{1}R_{2}\dot{\theta}_{2}^{-}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}) + 2.0R_{1}R_{3}\dot{\theta}_{1}^{+}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 2.0R_{1}R_{3}\dot{\theta}_{1}^{+}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 2.0R_{1}R_{3}\dot{\theta}_{1}^{-}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 2.0R_{1}R_{3}\dot{\theta}_{1}^{-}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 1.0R_{1}R_{3}\dot{\theta}_{2}^{+}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 1.0R_{1}R_{3}\dot{\theta}_{2}^{+}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 1.0R_{1}R_{3}\dot{\theta}_{2}^{-}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 1.0R_{1}R_{3}\dot{\theta}_{2}^{-}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 1.0R_{1}R_{3}\dot{\theta}_{3}^{+}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 1.0R_{1}R_{3}\dot{\theta}_{3}^{+}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 1.0R_{1}R_{3}\dot{\theta}_{3}^{-}m_{3}\sin(\theta_{1}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 1.0R_{1}R_{3}\dot{\theta}_{3}^{-}m_{3}\cos(\theta_{1}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 1.0R_{2}^{2}\dot{\theta}_{1}^{+}m_{2}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{2}^{2}\dot{\theta}_{1}^{+}m_{2}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{2}^{2}\dot{\theta}_{1}^{+}m_{3}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{2}^{2}\dot{\theta}_{1}^{+}m_{3}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{2}^{2}\dot{\theta}_{1}^{-}m_{2}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{2}^{2}\dot{\theta}_{1}^{-}m_{2}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{2}^{2}\dot{\theta}_{1}^{-}m_{3}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{2}^{2}\dot{\theta}_{1}^{-}m_{3}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{2}^{2}\dot{\theta}_{2}^{+}m_{2}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{2}^{2}\dot{\theta}_{2}^{+}m_{2}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{2}^{2}\dot{\theta}_{2}^{+}m_{3}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}) + 1.0R_{2}^{2}\dot{\theta}_{2}^{+}m_{3}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{2}^{2}\dot{\theta}_{2}^{-}m_{2}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{2}^{2}\dot{\theta}_{2}^{-}m_{2}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{2}^{2}\dot{\theta}_{2}^{-}m_{3}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}) - 1.0R_{2}^{2}\dot{\theta}_{2}^{-}m_{3}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}) + 2.0R_{2}R_{3}\dot{\theta}_{1}^{+}m_{3}\sin(\theta_{1}^{-}+\theta_{2}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 2.0R_{2}R_{3}\dot{\theta}_{1}^{+}m_{3}\cos(\theta_{1}^{-}+\theta_{2}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 2.0R_{2}R_{3}\dot{\theta}_{1}^{-}m_{3}\sin(\theta_{1}^{-}+\theta_{2}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 2.0R_{2}R_{3}\dot{\theta}_{1}^{-}m_{3}\cos(\theta_{1}^{-}+\theta_{2}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 2.0R_{2}R_{3}\dot{\theta}_{2}^{+}m_{3}\sin(\theta_{1}^{-}+\theta_{2}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 2.0R_{2}R_{3}\dot{\theta}_{2}^{+}m_{3}\cos(\theta_{1}^{-}+\theta_{2}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 2.0R_{2}R_{3}\dot{\theta}_{2}^{-}m_{3}\sin(\theta_{1}^{-}+\theta_{2}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 2.0R_{2}R_{3}\dot{\theta}_{2}^{-}m_{3}\cos(\theta_{1}^{-}+\theta_{2}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 1.0R_{2}R_{3}\dot{\theta}_{3}^{+}m_{3}\sin(\theta_{1}^{-}+\theta_{2}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 1.0R_{2}R_{3}\dot{\theta}_{3}^{+}m_{3}\cos(\theta_{1}^{-}+\theta_{2}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 1.0R_{2}R_{3}\dot{\theta}_{3}^{-}m_{3}\sin(\theta_{1}^{-}+\theta_{2}^{-})\sin(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 1.0R_{2}R_{3}\dot{\theta}_{3}^{-}m_{3}\cos(\theta_{1}^{-}+\theta_{2}^{-})\cos(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 1.0R_{3}^{2}\dot{\theta}_{1}^{+}m_{3}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) + 1.0R_{3}^{2}\dot{\theta}_{1}^{+}m_{3}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 1.0R_{3}^{2}\dot{\theta}_{1}^{-}m_{3}\sin^{2}(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) - 1.0R_{3}^{2}\dot{\theta}_{1}^{-}m_{3}\cos^{2}(\theta_{1}^{-}+\theta_{2}^{-}+\theta_{3}^{-}) +
$$

$$1.0R_3^2\dot{\theta}_2^+m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_2^+m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_2^-m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_2^-m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_3^+m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_3^+m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_3^-m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_3^-m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)=\lambda\cos\left(\theta_1^-\right)+\lambda\cos\left(\theta_1^-+\theta_2^-\right)+\lambda\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)$$

$$1.0R_1R_2\dot{\theta}_1^+m_2\sin\left(\theta_1^-\right)\sin\left(\theta_1^-+\theta_2^-\right)+1.0R_1R_2\dot{\theta}_1^+m_2\cos\left(\theta_1^-\right)\cos\left(\theta_1^-+\theta_2^-\right)+1.0R_1R_2\dot{\theta}_1^+m_3\sin\left(\theta_1^-\right)\sin\left(\theta_1^-+\theta_2^-\right)+1.0R_1R_2\dot{\theta}_1^+m_3\cos\left(\theta_1^-\right)\cos\left(\theta_1^-+\theta_2^-\right)-1.0R_1R_2\dot{\theta}_1^-m_2\sin\left(\theta_1^-\right)\sin\left(\theta_1^-+\theta_2^-\right)-1.0R_1R_2\dot{\theta}_1^-m_2\cos\left(\theta_1^-\right)\cos\left(\theta_1^-+\theta_2^-\right)-1.0R_1R_2\dot{\theta}_1^-m_3\sin\left(\theta_1^-\right)\sin\left(\theta_1^-+\theta_2^-\right)-1.0R_1R_2\dot{\theta}_1^-m_3\cos\left(\theta_1^-\right)\cos\left(\theta_1^-+\theta_2^-\right)+1.0R_1R_3\dot{\theta}_1^+m_3\sin\left(\theta_1^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_1R_3\dot{\theta}_1^+m_3\cos\left(\theta_1^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_1R_3\dot{\theta}_1^-m_3\sin\left(\theta_1^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_1R_3\dot{\theta}_1^-m_3\cos\left(\theta_1^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_2^2\dot{\theta}_1^+m_2\sin^2\left(\theta_1^-+\theta_2^-\right)+1.0R_2^2\dot{\theta}_1^+m_2\cos^2\left(\theta_1^-+\theta_2^-\right)+1.0R_2^2\dot{\theta}_1^+m_3\sin^2\left(\theta_1^-+\theta_2^-\right)+1.0R_2^2\dot{\theta}_1^+m_3\cos^2\left(\theta_1^-+\theta_2^-\right)-1.0R_2^2\dot{\theta}_1^-m_2\sin^2\left(\theta_1^-+\theta_2^-\right)-1.0R_2^2\dot{\theta}_1^-m_2\cos^2\left(\theta_1^-+\theta_2^-\right)-1.0R_2^2\dot{\theta}_1^-m_3\sin^2\left(\theta_1^-+\theta_2^-\right)-1.0R_2^2\dot{\theta}_1^-m_3\cos^2\left(\theta_1^-+\theta_2^-\right)+1.0R_2^2\dot{\theta}_2^+m_2\sin^2\left(\theta_1^-+\theta_2^-\right)+1.0R_2^2\dot{\theta}_2^+m_2\cos^2\left(\theta_1^-+\theta_2^-\right)+1.0R_2^2\dot{\theta}_2^+m_3\sin^2\left(\theta_1^-+\theta_2^-\right)+1.0R_2^2\dot{\theta}_2^+m_3\cos^2\left(\theta_1^-+\theta_2^-\right)-1.0R_2^2\dot{\theta}_2^-m_2\sin^2\left(\theta_1^-+\theta_2^-\right)-1.0R_2^2\dot{\theta}_2^-m_2\cos^2\left(\theta_1^-+\theta_2^-\right)-1.0R_2^2\dot{\theta}_2^-m_3\sin^2\left(\theta_1^-+\theta_2^-\right)-1.0R_2^2\dot{\theta}_2^-m_3\cos^2\left(\theta_1^-+\theta_2^-\right)+2.0R_2R_3\dot{\theta}_1^+m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+2.0R_2R_3\dot{\theta}_1^+m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-2.0R_2R_3\dot{\theta}_1^-m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-2.0R_2R_3\dot{\theta}_1^-m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+2.0R_2R_3\dot{\theta}_2^+m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+2.0R_2R_3\dot{\theta}_2^+m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-2.0R_2R_3\dot{\theta}_2^-m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-2.0R_2R_3\dot{\theta}_2^-m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_2R_3\dot{\theta}_3^+m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_2R_3\dot{\theta}_3^+m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_2R_3\dot{\theta}_3^-m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_2R_3\dot{\theta}_3^-m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_1^+m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_1^+m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_1^-m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_1^-m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_2^+m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_2^+m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_2^-m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_2^-m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_3^+m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_3^+m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_3^-m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_3^-m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)=\lambda\cos\left(\theta_1^-+\theta_2^-\right)+\lambda\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)$$

$$1.0R_1R_3\dot{\theta}_1^+m_3\sin\left(\theta_1^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_1R_3\dot{\theta}_1^+m_3\cos\left(\theta_1^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_1R_3\dot{\theta}_1^-m_3\sin\left(\theta_1^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_1R_3\dot{\theta}_1^-m_3\cos\left(\theta_1^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_2R_3\dot{\theta}_1^+m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_2R_3\dot{\theta}_1^+m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_2R_3\dot{\theta}_1^-m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_2R_3\dot{\theta}_1^-m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_2R_3\dot{\theta}_2^+m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_2R_3\dot{\theta}_2^+m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_2R_3\dot{\theta}_2^-m_3\sin\left(\theta_1^-+\theta_2^-\right)\sin\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_2R_3\dot{\theta}_2^-m_3\cos\left(\theta_1^-+\theta_2^-\right)\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_1^+m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_1^+m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_1^-m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_1^-m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_2^+m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_2^+m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_2^-m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_2^-m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_3^+m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)+1.0R_3^2\dot{\theta}_3^+m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_3^-m_3\sin^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)-1.0R_3^2\dot{\theta}_3^-m_3\cos^2\left(\theta_1^-+\theta_2^-+\theta_3^-\right)=\lambda\cos\left(\theta_1^-+\theta_2^-+\theta_3^-\right)$$

$$0.5R_1^2\left(\dot{\theta}_1^+\right)^2m_1\sin^2\left(\theta_1^-\right)+0.5R_1^2\left(\dot{\theta}_1^+\right)^2m_1\cos^2\left(\theta_1^-\right)+0.5R_1^2\left(\dot{\theta}_1^+\right)^2m_2\sin^2\left(\theta_1^-\right)+0.5R_1^2\left(\dot{\theta}_1^+\right)^2m_2\cos^2\left(\theta_1^-\right)+0.5R_1^2\left(\dot{\theta}_1^+\right)^2m_3\sin^2\left(\theta_1^-\right)+0.5R_1^2\left(\dot{\theta}_1^+\right)^2m_3\cos^2\left(\theta_1^-\right)-0.5R_1^2\left(\dot{\theta}_1^-\right)^2m_1\sin^2\left(\theta_1^-\right)-0.5R_1^2\left(\dot{\theta}_1^-\right)^2m_1\cos^2\left(\theta_1^-\right)-0.5R_1^2\left(\dot{\theta}_1^-\right)^2m_2\sin^2\left(\theta_1^-\right)-0.5R_1^2\left(\dot{\theta}_1^-\right)^2m_2\cos^2\left(\theta_1^-\right)-0.5R_1^2\left(\dot{\theta}_1^-\right)^2m_3\sin^2\left(\theta_1^-\right)-0.5R_1^2\left(\dot{\theta}_1^-\right)^2m_3\cos^2\left(\theta_1^-\right)+1.0R_1R_2\left(\dot{\theta}_1^+\right)^2m_2\sin\left(\theta_1^-\right)\sin\left(\theta_1^-+\theta_2^-\right)+1.0R_1R_2\left(\dot{\theta}_1^+\right)^2m_2\cos\left(\theta_1^-\right)\cos\left(\theta_1^-+\theta_2^-\right)+$$

$$1.0R_1R_2\left(\dot{\theta}_1^+\right)^2 m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-) + 1.0R_1R_2\left(\dot{\theta}_1^+\right)^2 m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-) +$$
$$1.0R_1R_2\dot{\theta}_1^+\dot{\theta}_2^+ m_2\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-) + 1.0R_1R_2\dot{\theta}_1^+\dot{\theta}_2^+ m_2\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-) +$$
$$1.0R_1R_2\dot{\theta}_1^+\dot{\theta}_2^+ m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-) + 1.0R_1R_2\dot{\theta}_1^+\dot{\theta}_2^+ m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-) -$$
$$1.0R_1R_2\left(\dot{\theta}_1^-\right)^2 m_2\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-) - 1.0R_1R_2\left(\dot{\theta}_1^-\right)^2 m_2\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-) -$$
$$1.0R_1R_2\left(\dot{\theta}_1^-\right)^2 m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-) - 1.0R_1R_2\left(\dot{\theta}_1^-\right)^2 m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-) -$$
$$1.0R_1R_2\dot{\theta}_1^-\dot{\theta}_2^- m_2\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-) - 1.0R_1R_2\dot{\theta}_1^-\dot{\theta}_2^- m_2\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-) -$$
$$1.0R_1R_2\dot{\theta}_1^-\dot{\theta}_2^- m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-) - 1.0R_1R_2\dot{\theta}_1^-\dot{\theta}_2^- m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-) +$$
$$1.0R_1R_3\left(\dot{\theta}_1^+\right)^2 m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-) + 1.0R_1R_3\left(\dot{\theta}_1^+\right)^2 m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-) +$$
$$1.0R_1R_3\dot{\theta}_1^+\dot{\theta}_2^+ m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-) + 1.0R_1R_3\dot{\theta}_1^+\dot{\theta}_2^+ m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-) +$$
$$1.0R_1R_3\dot{\theta}_1^+\dot{\theta}_3^+ m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-) + 1.0R_1R_3\dot{\theta}_1^+\dot{\theta}_3^+ m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-) -$$
$$1.0R_1R_3\left(\dot{\theta}_1^-\right)^2 m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-) - 1.0R_1R_3\left(\dot{\theta}_1^-\right)^2 m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-) -$$
$$1.0R_1R_3\dot{\theta}_1^-\dot{\theta}_2^- m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-) - 1.0R_1R_3\dot{\theta}_1^-\dot{\theta}_2^- m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-) -$$
$$1.0R_1R_3\dot{\theta}_1^-\dot{\theta}_3^- m_3\sin(\theta_1^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-) - 1.0R_1R_3\dot{\theta}_1^-\dot{\theta}_3^- m_3\cos(\theta_1^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-) +$$
$$0.5R_2^2\left(\dot{\theta}_1^+\right)^2 m_2\sin^2(\theta_1^-+\theta_2^-) + 0.5R_2^2\left(\dot{\theta}_1^+\right)^2 m_2\cos^2(\theta_1^-+\theta_2^-) + 0.5R_2^2\left(\dot{\theta}_1^+\right)^2 m_3\sin^2(\theta_1^-+\theta_2^-) +$$
$$0.5R_2^2\left(\dot{\theta}_1^+\right)^2 m_3\cos^2(\theta_1^-+\theta_2^-) + 1.0R_2^2\dot{\theta}_1^+\dot{\theta}_2^+ m_2\sin^2(\theta_1^-+\theta_2^-) + 1.0R_2^2\dot{\theta}_1^+\dot{\theta}_2^+ m_2\cos^2(\theta_1^-+\theta_2^-) +$$
$$1.0R_2^2\dot{\theta}_1^+\dot{\theta}_2^+ m_3\sin^2(\theta_1^-+\theta_2^-) + 1.0R_2^2\dot{\theta}_1^+\dot{\theta}_2^+ m_3\cos^2(\theta_1^-+\theta_2^-) - 0.5R_2^2\left(\dot{\theta}_1^-\right)^2 m_2\sin^2(\theta_1^-+\theta_2^-) -$$
$$0.5R_2^2\left(\dot{\theta}_1^-\right)^2 m_2\cos^2(\theta_1^-+\theta_2^-) - 0.5R_2^2\left(\dot{\theta}_1^-\right)^2 m_3\sin^2(\theta_1^-+\theta_2^-) - 0.5R_2^2\left(\dot{\theta}_1^-\right)^2 m_3\cos^2(\theta_1^-+\theta_2^-) -$$
$$1.0R_2^2\dot{\theta}_1^-\dot{\theta}_2^- m_2\sin^2(\theta_1^-+\theta_2^-) - 1.0R_2^2\dot{\theta}_1^-\dot{\theta}_2^- m_2\cos^2(\theta_1^-+\theta_2^-) - 1.0R_2^2\dot{\theta}_1^-\dot{\theta}_2^- m_3\sin^2(\theta_1^-+\theta_2^-) -$$
$$1.0R_2^2\dot{\theta}_1^-\dot{\theta}_2^- m_3\cos^2(\theta_1^-+\theta_2^-) + 0.5R_2^2\left(\dot{\theta}_2^+\right)^2 m_2\sin^2(\theta_1^-+\theta_2^-) + 0.5R_2^2\left(\dot{\theta}_2^+\right)^2 m_2\cos^2(\theta_1^-+\theta_2^-) +$$
$$0.5R_2^2\left(\dot{\theta}_2^+\right)^2 m_3\sin^2(\theta_1^-+\theta_2^-) + 0.5R_2^2\left(\dot{\theta}_2^+\right)^2 m_3\cos^2(\theta_1^-+\theta_2^-) - 0.5R_2^2\left(\dot{\theta}_2^-\right)^2 m_2\sin^2(\theta_1^-+\theta_2^-) -$$
$$0.5R_2^2\left(\dot{\theta}_2^-\right)^2 m_2\cos^2(\theta_1^-+\theta_2^-) - 0.5R_2^2\left(\dot{\theta}_2^-\right)^2 m_3\sin^2(\theta_1^-+\theta_2^-) - 0.5R_2^2\left(\dot{\theta}_2^-\right)^2 m_3\cos^2(\theta_1^-+\theta_2^-) +$$
$$1.0R_2R_3\left(\dot{\theta}_1^+\right)^2 m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)+1.0R_2R_3\left(\dot{\theta}_1^+\right)^2 m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)+$$
$$2.0R_2R_3\dot{\theta}_1^+\dot{\theta}_2^+ m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)+2.0R_2R_3\dot{\theta}_1^+\dot{\theta}_2^+ m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)+$$
$$1.0R_2R_3\dot{\theta}_1^+\dot{\theta}_3^+ m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)+1.0R_2R_3\dot{\theta}_1^+\dot{\theta}_3^+ m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)-$$
$$1.0R_2R_3\left(\dot{\theta}_1^-\right)^2 m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)-1.0R_2R_3\left(\dot{\theta}_1^-\right)^2 m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)-$$
$$2.0R_2R_3\dot{\theta}_1^-\dot{\theta}_2^- m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)-2.0R_2R_3\dot{\theta}_1^-\dot{\theta}_2^- m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)-$$
$$1.0R_2R_3\dot{\theta}_1^-\dot{\theta}_3^- m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)-1.0R_2R_3\dot{\theta}_1^-\dot{\theta}_3^- m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)+$$
$$1.0R_2R_3\left(\dot{\theta}_2^+\right)^2 m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)+1.0R_2R_3\left(\dot{\theta}_2^+\right)^2 m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)+$$
$$1.0R_2R_3\dot{\theta}_2^+\dot{\theta}_3^+ m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)+1.0R_2R_3\dot{\theta}_2^+\dot{\theta}_3^+ m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)-$$
$$1.0R_2R_3\left(\dot{\theta}_2^-\right)^2 m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)-1.0R_2R_3\left(\dot{\theta}_2^-\right)^2 m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)-$$
$$1.0R_2R_3\dot{\theta}_2^-\dot{\theta}_3^- m_3\sin(\theta_1^-+\theta_2^-)\sin(\theta_1^-+\theta_2^-+\theta_3^-)-1.0R_2R_3\dot{\theta}_2^-\dot{\theta}_3^- m_3\cos(\theta_1^-+\theta_2^-)\cos(\theta_1^-+\theta_2^-+\theta_3^-)+$$
$$0.5R_3^2\left(\dot{\theta}_1^+\right)^2 m_3\sin^2(\theta_1^-+\theta_2^-+\theta_3^-) + 0.5R_3^2\left(\dot{\theta}_1^+\right)^2 m_3\cos^2(\theta_1^-+\theta_2^-+\theta_3^-) +$$
$$1.0R_3^2\dot{\theta}_1^+\dot{\theta}_2^+ m_3\sin^2(\theta_1^-+\theta_2^-+\theta_3^-) + 1.0R_3^2\dot{\theta}_1^+\dot{\theta}_2^+ m_3\cos^2(\theta_1^-+\theta_2^-+\theta_3^-) +$$
$$1.0R_3^2\dot{\theta}_1^+\dot{\theta}_3^+ m_3\sin^2(\theta_1^-+\theta_2^-+\theta_3^-) + 1.0R_3^2\dot{\theta}_1^+\dot{\theta}_3^+ m_3\cos^2(\theta_1^-+\theta_2^-+\theta_3^-) -$$
$$0.5R_3^2\left(\dot{\theta}_1^-\right)^2 m_3\sin^2(\theta_1^-+\theta_2^-+\theta_3^-) - 0.5R_3^2\left(\dot{\theta}_1^-\right)^2 m_3\cos^2(\theta_1^-+\theta_2^-+\theta_3^-) -$$
$$1.0R_3^2\dot{\theta}_1^-\dot{\theta}_2^- m_3\sin^2(\theta_1^-+\theta_2^-+\theta_3^-) - 1.0R_3^2\dot{\theta}_1^-\dot{\theta}_2^- m_3\cos^2(\theta_1^-+\theta_2^-+\theta_3^-) -$$
$$1.0R_3^2\dot{\theta}_1^-\dot{\theta}_3^- m_3\sin^2(\theta_1^-+\theta_2^-+\theta_3^-) - 1.0R_3^2\dot{\theta}_1^-\dot{\theta}_3^- m_3\cos^2(\theta_1^-+\theta_2^-+\theta_3^-) +$$
$$0.5R_3^2\left(\dot{\theta}_2^+\right)^2 m_3\sin^2(\theta_1^-+\theta_2^-+\theta_3^-) + 0.5R_3^2\left(\dot{\theta}_2^+\right)^2 m_3\cos^2(\theta_1^-+\theta_2^-+\theta_3^-) +$$

$1.0R_3^2\dot{\theta}_2^+\dot{\theta}_3^+ m_3 \sin^2(\theta_1^- + \theta_2^- + \theta_3^-) + 1.0R_3^2\dot{\theta}_2^+\dot{\theta}_3^+ m_3 \cos^2(\theta_1^- + \theta_2^- + \theta_3^-) - 0.5R_3^2\left(\dot{\theta}_2^-\right)^2 m_3 \sin^2(\theta_1^- + \theta_2^- + \theta_3^-) - 0.5R_3^2\left(\dot{\theta}_2^-\right)^2 m_3 \cos^2(\theta_1^- + \theta_2^- + \theta_3^-) - 1.0R_3^2\dot{\theta}_2^-\dot{\theta}_3^- m_3 \sin^2(\theta_1^- + \theta_2^- + \theta_3^-) - 1.0R_3^2\dot{\theta}_2^-\dot{\theta}_3^- m_3 \cos^2(\theta_1^- + \theta_2^- + \theta_3^-) + 0.5R_3^2\left(\dot{\theta}_3^+\right)^2 m_3 \sin^2(\theta_1^- + \theta_2^- + \theta_3^-) + 0.5R_3^2\left(\dot{\theta}_3^+\right)^2 m_3 \cos^2(\theta_1^- + \theta_2^- + \theta_3^-) - 0.5R_3^2\left(\dot{\theta}_3^-\right)^2 m_3 \sin^2(\theta_1^- + \theta_2^- + \theta_3^-) - 0.5R_3^2\left(\dot{\theta}_3^-\right)^2 m_3 \cos^2(\theta_1^- + \theta_2^- + \theta_3^-) = 0$

2.2 Problem 7 (15pts)

Since solving the analytical symbolic solution of the impact update rules for the triple-pendulum system is too slow, here we will solve it along within the simulation. The idea is, when the impact happens, substitute the numerical values of q and $\dot{q}$ at that moment into the equations you got in Problem 6, thus you will just need to solve a set equations with most terms being numerical values (which is very fast).

The first thing is to write a function called "impact_update_triple_pend". This function at least takes in the current state of the system $s(t^-) = [q(t^-), \dot{q}(t^-)]$ or $\dot{q}(t^-)$, inside the function you need to substitute in $q(t^-)$ and $\dot{q}(t^-)$, solve for and return $s(t^+) = [q(t^+), \dot{q}(t^+)]$ or $\dot{q}(t^+)$ (which should be numerical values now). This function will replace lambdify, and you can use SymPy's "sym.N()" or "expr.evalf()" methods to convert SymPy expressions into numerical values. Test your function with $\theta_1(\tau^-) = \theta_2(\tau^-) = \theta_3(\tau^-) = 0$ and $\dot{\theta}_1(\tau^-) = \dot{\theta}_2(\tau^-) = \dot{\theta}_3(\tau^-) = -1$.

Turn in: A copy of your "impact_update_triple_pend" function, and the test result of your function.

[15]:
```
def impact_update_triple_pend(s):
    """
    Update the state vector `s` upon impact in a triple pendulum system,␣
↪applying reflection
    and restitution to `thetadot` values.

    Parameters
    ----------
    s : list or np.array
        State vector `[theta1, theta2, theta3, theta1_dot, theta2_dot,␣
↪theta3_dot]`
        where `theta` values represent angular positions and `theta_dot` values
        represent angular velocities.

    Returns
    -------
    np.array
        Updated state vector `[theta1, theta2, theta3, theta1_dot, theta2_dot,␣
↪theta3_dot]`
        after applying the impact equations.

    Notes
    -----
```

```
    The function uses the symbolic impact equations to solve for the␣
↪post-impact angular
    velocities (`theta_dot_plus`). The pre-impact angles (`theta`) remain the␣
↪same.
    """
    # Define substitution dictionary with input state
    values = {
        th1_minus: s[0], th2_minus: s[1], th3_minus: s[2],
        th1_dot_minus: s[3], th2_dot_minus: s[4], th3_dot_minus: s[5]
    }

    # Solve the impact equations for post-impact velocities and the lambda␣
↪multiplier
    impact_solutions = sym.solve(impact_eq.subs(values), [th1_dot_plus,␣
↪th2_dot_plus, th3_dot_plus, lamb], dict=True)

    # Use the second solution as the valid one, and extract angular velocity␣
↪components
    # We want the second solution because it is the one with the correct signs
    soln = impact_solutions[1]
    th1_plus = float(soln[th1_dot_plus])
    th2_plus = float(soln[th2_dot_plus])
    th3_plus = float(soln[th3_dot_plus])

    # Return the updated state vector with original angles and updated angular␣
↪velocities
    return np.array([s[0], s[1], s[2], th1_plus, th2_plus, th3_plus])
```

[16]:
```
VALS = np.array([0,0,0,-1,-1,-1])
sol = impact_update_triple_pend(VALS)
display(Markdown(r'$Updated$ $state$ $vector$ $after$ $impact:$'))
display(Markdown(f'$\\theta_1 = {sol[0]}$'))
display(Markdown(f'$\\theta_2 = {sol[1]}$'))
display(Markdown(f'$\\theta_3 = {sol[2]}$'))
display(Markdown(f'$\\dot{{\\theta}}_1^+ = {sol[3]}$'))
display(Markdown(f'$\\dot{{\\theta}}_2^+ = {sol[4]}$'))
display(Markdown(f'$\\dot{{\\theta}}_3^+ = {sol[5]}$'))
```

$Updated$ $state$ $vector$ $after$ $impact:$

$\theta_1 = 0.0$

$\theta_2 = 0.0$

$\theta_3 = 0.0$

$\dot{\theta}_1^+ = -1.0$

$\dot{\theta}_2^+ = -1.0$

$\dot{\theta}_3^+ = 11.0$

2.3 Problem 8 (15pts)

Similar to the single-pendulum system, you will still want to implement a function named "impact_condition_triple_pend" to indicate the moment when impact happens. Again, you need to use the constraint ϕ. After obtaining the impact condition function, simulate the triple-pendulum system with impact for $t \in [0, 2], dt = 0.01$ with initial condition $\theta_1 = \frac{\pi}{3}, \theta_2 = \frac{\pi}{3}, \theta_3 = -\frac{\pi}{3}$ and $\dot{\theta}_1 = \dot{\theta}_2 = \dot{\theta}_3 = 0$. Plot the simulated trajectory versus time and animate your simulated trajectory.

Hint 1: You will need to modify the simulate function!

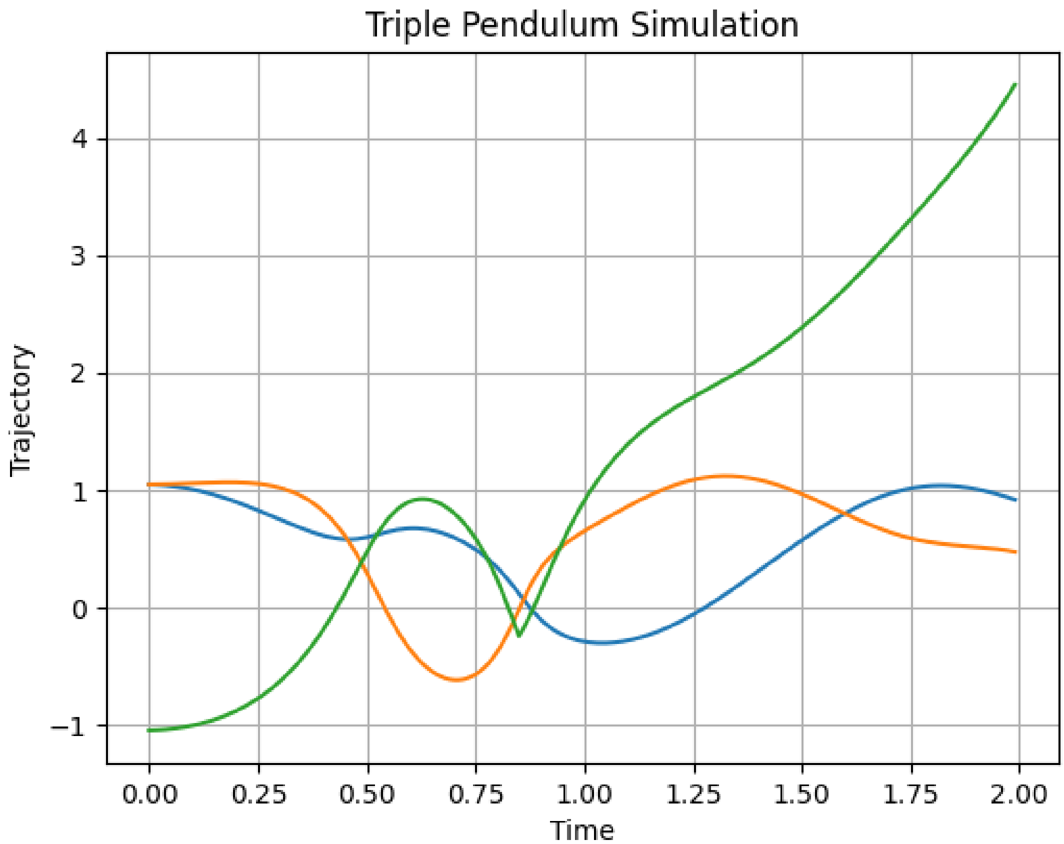
Turn in: A copy of code for the impact update function and simulate function, as well as code output including the plot of simulated trajectory and the animation. The video should be uploaded separately from the .pdf file through Canvas, and it should be in ".mp4" format. You can use screen capture or record the screen directly with your phone.

[17]:
```
def triple_pendulum_ode_v2(x):
    return np.array([
        x[3], x[4], x[5],
        theta1_func(x[0], x[1], x[2], x[3], x[4], x[5]),
        theta2_func(x[0], x[1], x[2], x[3], x[4], x[5]),
        theta3_func(x[0], x[1], x[2], x[3], x[4], x[5])
    ])
```

[25]:
```
def impact_condition_triple_pend(x,threshold = 1e-1):
    """
    Checks if the impact condition is met based on the phi value.

    Parameters:
    ================================================
    x : np.array
        State vector [theta1, theta2, theta3, theta1_dot, theta2_dot,␣
 ↪theta3_dot].
    threshold : float, optional
        Threshold for detecting impact, default is 1e-1.

    Returns:
    ================================================
    bool : True if the phi value is within the threshold range, else False.
    """
    phi_val = phi_func(*x)

    if phi_val > -threshold and phi_val < threshold:
        return True
    else:
        return False
```

```
def simulate_impact_triple_pend(f,x0,tspan,dt,integrate):
    N = int((max(tspan)-min(tspan))/dt)
    x = np.copy(x0)
    xtraj = np.zeros((len(x0),N))

    for i in range(N):
        if impact_condition_triple_pend(x) is True:
            x = impact_update_triple_pend(x)
            xtraj[:,i] = integrate(f,x,dt)
        else:
            xtraj[:,i] = integrate(f,x,dt)
        x = np.copy(xtraj[:,i])

    return xtraj

s0 = np.array([
    np.pi/3,
    np.pi/3,
    -np.pi/3,
    0,
    0,
    0
])

traj = simulate_impact_triple_pend(triple_pendulum_ode_v2, s0, [0, 2], 0.01,␣
 ↪integrate)
print('shape of traj: ', traj.shape)

plt.plot(np.arange(200)*0.01,traj[0:3].T)
plt.grid(True)
plt.xlabel('Time')
plt.ylabel('Trajectory')
plt.title('Triple Pendulum Simulation')
plt.show()
```

```
shape of traj:  (6, 200)
```



[26]:
```
animate_triple_pend(traj,L1=1,L2=1,L3=1,T=10)
```

<IPython.core.display.HTML object>

2.4 Problem 9 (5pts)

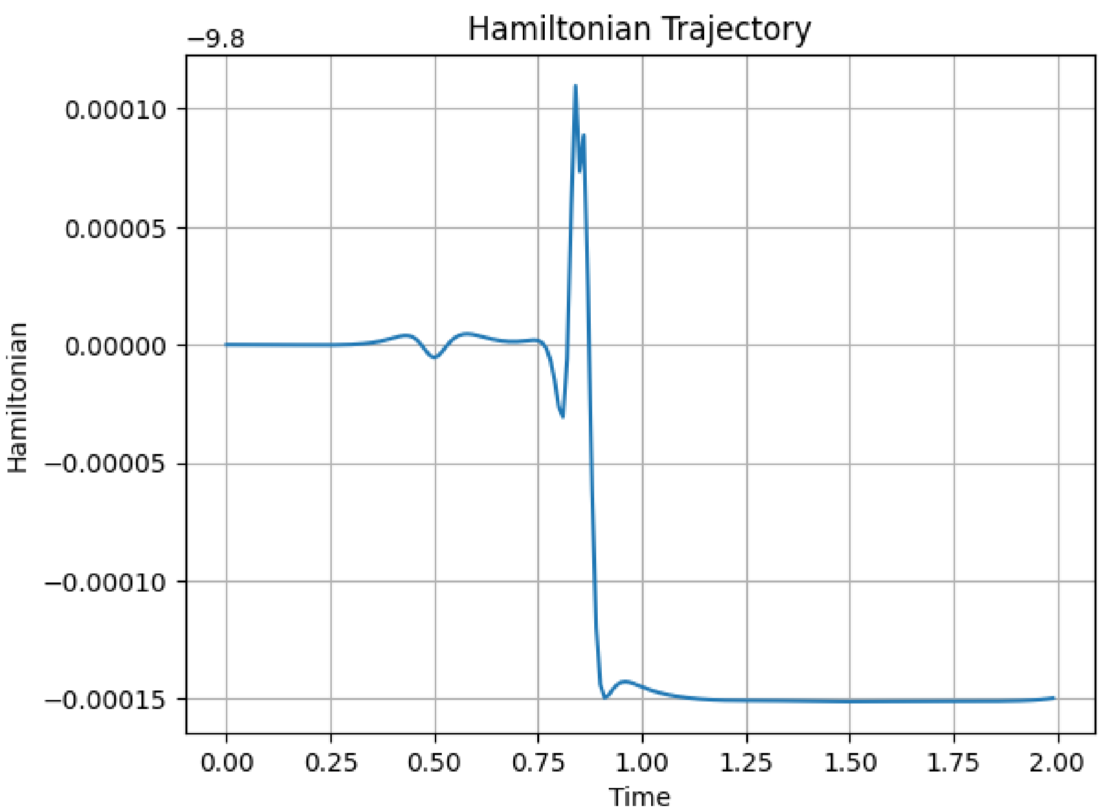
Compute and plot the Hamiltonian of the simulated trajectory for the triple-pendulum system with impact.

Turn in: A copy of code used to compute the Hamiltonian, also include the code output, which should the plot of the Hamiltonian versus time.

[27]:
```
# Lambdify hamiltonian
hamiltonian = sym.lambdify([theta1, theta2, theta3, theta1_dot, theta2_dot,␣
↪theta3_dot], Hamiltonian_pr2.subs(subs))
```

[29]:
```
# compute trajectory
dt = 0.01

# define the hamiltonian trajectory
```

```
traj = simulate_impact_triple_pend(triple_pendulum_ode_v2, s0, [0, 2], dt,␣
    ↪integrate)

# compute the hamiltonian for each point in the trajectory
hamiltonian_traj = np.array([hamiltonian(*s) for s in traj.T])

plt.plot(np.arange(200)*0.01, hamiltonian_traj)
plt.xlabel('Time')
plt.ylabel('Hamiltonian')
plt.title('Hamiltonian of the simulated trajectory')
plt.grid(True)
plt.show()
```




```
[ ]:
```