ME314 Homework 5

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.

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

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)
```

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

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.zeros(1, len(q))
    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
    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]:
```
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


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
```

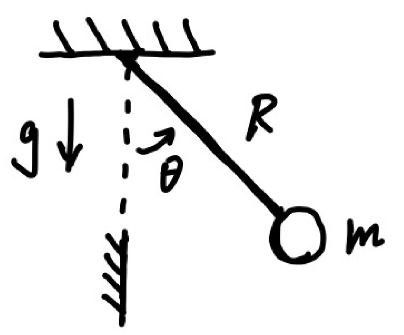
```python
    ============
    x_traj:
        simulated trajectory of x(t) from t=0 to tf
    """
    N = int((max(tspan)-min(tspan))/dt)
    x = np.copy(x0)
    tvec = np.linspace(min(tspan),max(tspan),N)
    xtraj = np.zeros((len(x0),N))

    for i in range(N):
        t = tvec[i]
        xtraj[:,i]=integrate(f,x,t,dt)
        x = np.copy(xtraj[:,i])
    return xtraj
```

In [57]:

```python
# def animate_double_pend(theta_array,L1=1,L2=1,T=10):
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
    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
    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])
    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=xx1, y=yy1,
               mode='markers', name='Pendulum 1 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='Double Pendulum Simulation',
                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.
    # 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)

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

    #######################
    # 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])
```

```python
    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)
```

In [7]:
```python
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>"))
```



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.

In [42]:
```python
# - define variables and constants
m, R, g = sym.symbols(r'm, R, g')
t = sym.symbols(r't')
theta = sym.Function(r'\theta')(t)
thetad = theta.diff(t)

# - define x and y as a function of theta
x = R * sym.sin(theta)
xd = x.diff(t)
y = -R * sym.cos(theta)
yd = y.diff(t)

# - make a substitution dict
subs_dict = {
    m : 1,
    R : 1,
    g : 9.8,
}

# - define state vector
q = sym.Matrix([theta])
qd = q.diff(t)
qdd = qd.diff(t)

# - define KE, U, and Lagrangian of system
KE = 0.5 * m * (xd**2 + yd**2)
U = m * g * y
lagrangian1 = KE - U
lagrangian1 = lagrangian1.simplify()

print("Lagrangian:")
display(lagrangian1)
```

Lagrangian:

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

In [43]:
```python
# - compute non-constrained EL
eqns = compute_EL(lagrangian1, q)
eqns_solved = solve_EL(eqns, qdd)

print("Euler-Lagrange equations:")
display(eqns)

print("Solved:")
for eq in eqns_solved:
    display(eq)
```

Euler-Lagrange equations:

$$\left[-1.0R^2m\frac{d^2}{dt^2}\theta(t) - Rgm\sin(\theta(t))\right] = [0]$$

Solved:

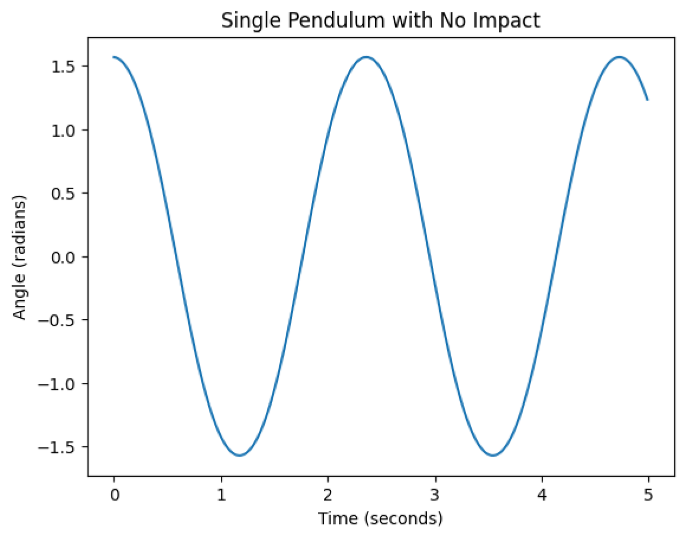
$$\frac{d^2}{dt^2}\theta(t) = -\frac{g\sin(\theta(t))}{R}$$

In [44]:
```python
# - make dxdt function
q_ext = sym.Matrix([theta, thetad])
thetadd_sym = eqns_solved[0].rhs.subs(subs_dict)
thetadd_np = sym.lambdify(q_ext, thetadd_sym)

# display(thetadd_sym)
# help(thetadd_np)

def dxdt_problem1(t, s):
    #derivatives of position and velocity are velocity and accel
    return np.array([s[1], thetadd_np(*s)])

# - define ICs
ICs = [np.pi/2, 0]
dt = 0.01
tspan = [0,5]

# - simulate system over times
traj_array = simulate(dxdt_problem1, ICs, tspan, dt, rk4)
print(len(traj_array))
```

```
2
```

In [45]:
```python
theta_array = traj_array[0]
t_array = np.arange(tspan[0], tspan[1], dt)

# - plot array over time
plt.plot(t_array, theta_array)
plt.xlabel("Time (seconds)")
plt.ylabel("Angle (radians)")
plt.title("Single Pendulum with No Impact")
```

Out[45]:
```
Text(0.5, 1.0, 'Single Pendulum with No Impact')
```



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:

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

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).

In [46]:
```python
#  - use setup provided in problem 1 - state, lagrangian
phi = theta
lamb = sym.symbols(r'\lambda')
q_sym, qd_sym = sym.symbols(r'q, \dot{q}')
q_subs = {theta: q_sym, thetad: qd_sym}

# - make equation dL/dqdot = lambda*dPhi/dq
dL_dqd_mat = lagrangian1.diff(qd)
dL_dqd = dL_dqd_mat[0]
dL_dqd_dot_qd = dL_dqd_mat.dot(qd)

lamb_dphi = sym.Matrix([lamb * phi.diff(a) for a in q])[0]
expr_a = dL_dqd.subs(q_subs)
expr_b = lamb_dphi.subs(q_subs)

# - make equation dL/dqd * q -L
expr_c = dL_dqd_dot_qd - lagrangian1
expr_d = expr_c.subs(q_subs)

print("dL_dqdot:")
display(expr_a)
print("Lambda * d(phi)/dq:")
display(expr_b)
print("dL_dqdot - L(q,qdot):")
display(expr_d)
```

```
dL_dqdot:
```
$1.0R^2\dot{q}m$
```
Lambda * d(phi)/dq:
```
λ
```
dL_dqdot - L(q,qdot):
```
$1.0R^2\dot{q}^2m - Rm\left(0.5R\dot{q}^2 + g\cos(q)\right)$

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.

In [47]:
```python
# - sub in dummy variables for the functions q(t)and qd(t)
qdtaup, qdtaum = sym.symbols(r'\dot{q}^{\tau+}, \dot{q}^{\tau-}')
qtaup, qtaum = sym.symbols(r'q^{\tau+}, q^{\tau-}')
qtaup_subs = {thetad:qdtaup}
qtaum_subs = {thetad:qdtaum}

#goal is to solve for qd(tau+) and q(tau+) as a function of
#pre-impact values
impact_a_LHS = dL_dqd.subs(qtaup_subs) - dL_dqd.subs(qtaum_subs)
impact_a = impact_a_LHS - lamb_dphi
impact_b = expr_c.subs(qtaup_subs) - expr_c.subs(qtaum_subs)
impact_b = impact_b.simplify()

expr_mat = sym.Matrix([impact_a, impact_b])
RHS = sym.zeros(len(expr_mat),1)
eqns = sym.Eq(expr_mat, RHS)

sol_vec = [qdtaup, lamb]
solns = sym.solve(eqns, sol_vec, dict=True)

print("Impact expressions to solve:")
display(eqns)

print("Solved:")
# - find solutions to qdot and lambda
#       - filter the solutions so that only the ones where lambda is
#               nonzero are valid

eqns_solved = []
for i, sol in enumerate(solns):

    #do some error checking - if lambda = 0, not valid
    if sol[lamb] == 0:
        continue

    for x in list(sol.keys()):
        sol_new = sol[x].simplify()
        eqn_solved = sym.Eq(x, sol_new)
        eqns_solved.append(eqn_solved)
```

```python
for eq in eqns_solved:
    display(eq)
```

Impact expressions to solve:

$$\begin{bmatrix} 1.0R^2\dot{q}^{\tau+}m - 1.0R^2\dot{q}^{\tau-}m - \lambda \\ 0.5R^2m\left(\left(\dot{q}^{\tau+}\right)^2 - \left(\dot{q}^{\tau-}\right)^2\right) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Solved:

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

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

In [48]:
```python
#lambdify and evaluate with given ICs
tau_state =  [0.01, 2] #q, qd at tau-

#make sympy expressions out of each q value
qdtaup_sy = eqns_solved[0].rhs
qtaup_sy = qtaum

impact_q = sym.Matrix([qtaum, qdtaum])
qdtaup_np = sym.lambdify(impact_q, qdtaup_sy)
qtaup_np = sym.lambdify(impact_q, qtaup_sy)

#test: what does impact update say about q^tau+ and q_dot^tau+?
a = qdtaup_np(*tau_state) #a = qdot, @ tau+, in numpy function form, as a function of ICs
b = qtaup_np(*tau_state) #b = q, @ tau+, in numpy function form, as a function of ICs

print(f"\nState before impact: \nTheta: {tau_state[0]} \nThetad: {tau_state[1]}")
print(f"\nState after impact: \nTheta: {b} \nThetad: {a}")

# print(help(qdtaup_np))
# print(help(qtaup_np))
```

```
State before impact:
Theta: 0.01
Thetad: 2

State after impact:
Theta: 0.01
Thetad: -2
```

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.

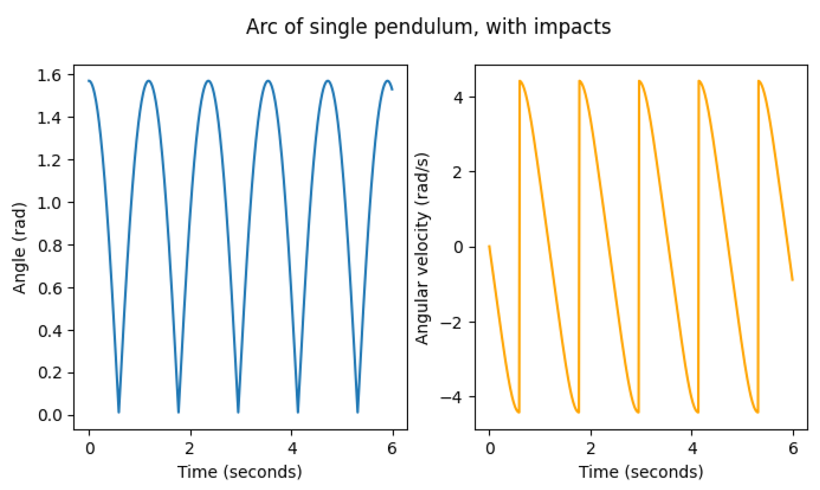
In [49]:
```python
# - define impact condition phi as point where theta equals zero
phi_f = sym.lambdify(sym.Matrix([theta, thetad]), phi)

# - define phi at initial timestep
phi_init = phi_f(*ICs)

#test out phi_f
print(phi_f(1.27, 8888))
print(phi_f(-1.27, 8888))

def impact_condition_p4(s):
    '''Checks for impact using impact condition s.'''
    return (phi_f(*s)/phi_init < 0)


# - define impact update function
def impact_update_p4(s):
    '''Applies the impact update and returns the state at tau+.'''
    return [qtaup_np(*s), qdtaup_np(*s)]
```

```
1.27
-1.27
```

In [50]:
```python
# - construct general loop structure of checking whether an impact
#       has occurred

def simulate_impact(t_span, dt, ICs, integrate, dxdt, impact_condition, impact_update):
    '''
    simulate(), but with an extra framework for detecting impact

    Inputs:
    - t_span: 2-elem array [to, tf]
    - dt: timestep, float
    - ICs: n-dim array with the initial state of system
    - integrate: type "function". for our integration scheme (usually RK4 or Euler)
    - dxdt: type "function". our derivative function, used to calculate next statew
    - impact_condition: type "function". takes in state s, returns True if particle passes through a boundary
    - impact_update: type "function". takes in s at tau-, returns the state of the system at tau+

    Returns:
    - traj_array: an nxm array, where m = length of the time vector and n = # of variables in state s
    '''

    #array indexing is necessary for altering next elements in array
    t_array = np.arange(t_span[0], t_span[1], dt)
    traj_array = np.zeros([len(ICs), len(t_array)])

    traj_array[:,0] = ICs

    for i, t in enumerate(t_array):

        if i == len(t_array) - 1:
            break #no going out of bounds!

        #get current value of s
        s = traj_array[:,i]

        #calculate s for next timestep
        s_next = integrate(dxdt, s, t, dt)

        #check if impact has occurred
        impact = impact_condition(s_next)

        #if impact has occurred, apply impact update
        if (impact):
            '''This is designed to alter the velocity of the particle
            just before impact. If we applied the impact update after impact
            (same position, changed velocity), there's a chance the particle
            would stay clipped into the wall.
            '''
            s_alt = impact_update(s)
            s_next = integrate(dxdt, s_alt, t, dt)

        #apply update to trajectory vector
        traj_array[:, i+1] = s_next

    return traj_array

tspan = [0,6]
traj = simulate_impact(tspan, dt, ICs, rk4, dxdt_problem1, impact_condition_p4, impact_update_p4)
```

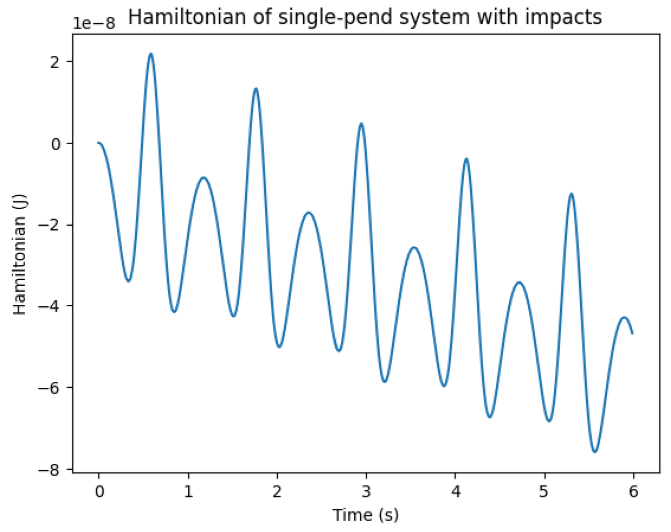
In [51]:
```python
angle_array = traj[0]
dtheta_dt = traj[1]
t_array = np.arange(tspan[0], tspan[1], dt)

fig, [ax1, ax2] = plt.subplots(1,2, figsize = (8,4))

ax1.plot(t_array, angle_array)
ax2.plot(t_array, dtheta_dt, color='orange')

ax1.set_xlabel("Time (seconds)")
ax1.set_ylabel("Angle (rad)")

ax2.set_xlabel("Time (seconds)")
ax2.set_ylabel("Angular velocity (rad/s)")
fig.suptitle("Arc of single pendulum, with impacts")
```

Out[51]:
```
Text(0.5, 0.98, 'Arc of single pendulum, with impacts')
```



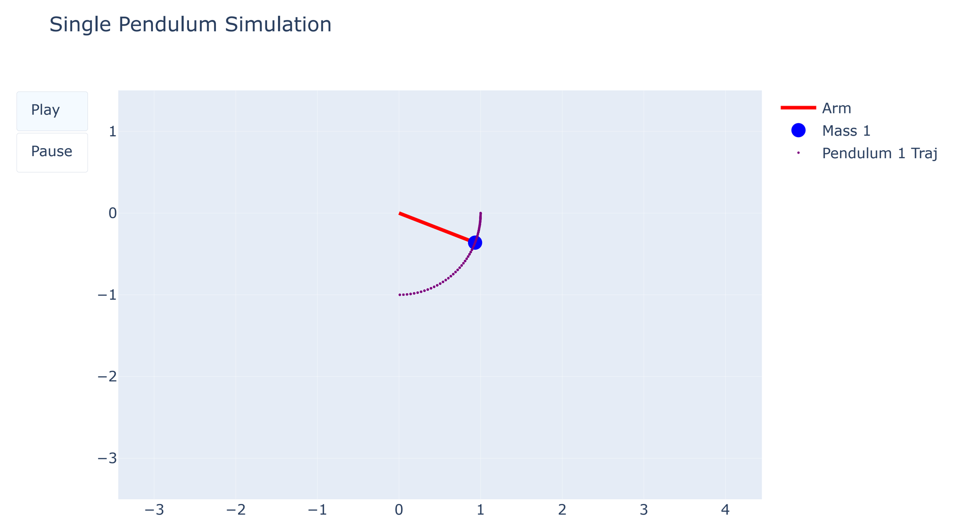
In [52]:
```python
#plot Hamiltonian of system over time
def H(s):
    #let R = 1 = M, g =9.81
    [theta, thetad] = s
    x = np.sin(theta)
    y = -np.cos(theta)
    xd = np.cos(theta) * thetad
    yd = np.sin(theta) * thetad

    KE = 0.5 * (xd**2 + yd**2)
    U = 9.8 * y
    return KE + U

ham_array = [H(s) for s in traj.T]

#plot
plt.figure()
plt.plot(t_array, ham_array)
plt.xlabel("Time (s)")
plt.ylabel("Hamiltonian (J)")
plt.title("Hamiltonian of single-pend system with impacts")
```

Out[52]:
```
Text(0.5, 1.0, 'Hamiltonian of single-pend system with impacts')
```



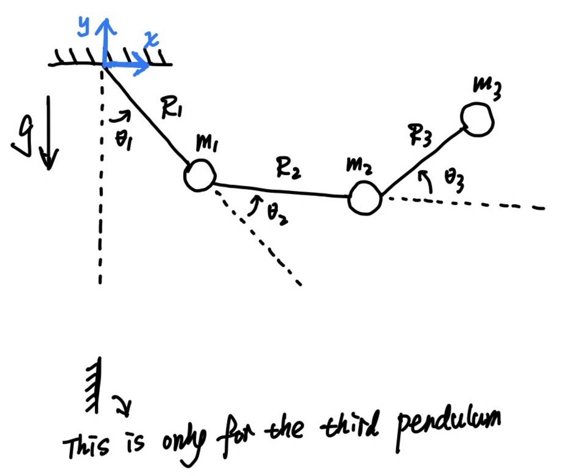
Although changes to the Hamiltonian are happening in an oscillating pattern that I don't like, the scale of changes to the Hamiltonian is on the order of 10^-8, even after impacts. The Hamiltonian suggests energy is being conserved even after elastic impacts, which is what we'd expect.

In [59]:
```python
theta_array = np.zeros([2,len(angle_array)])
theta_array[0,:] = angle_array

#change time based on how long the array with impacts becomes
animate_single_pend(theta_array, L1=1,T=6)
```



Problem 5 (10pts)

In [19]:
```python
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/raw/master/tripend_constrained.JPG' width=500' height='450'></table>"))
```



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!

In [70]:
```python
# - define variables and constants
t = sym.symbols(r't')
g = 9.81

theta1 = sym.Function(r'\theta_1')(t)
theta2 = sym.Function(r'\theta_2')(t)
theta3 = sym.Function(r'\theta_3')(t)

theta1d = theta1.diff(t)
theta2d = theta2.diff(t)
theta3d = theta3.diff(t)

# - define x and y as a function of theta
x1 = sym.sin(theta1)
y1 = -sym.cos(theta1)
x1d = x1.diff(t)
y1d = y1.diff(t)

x2 = x1 + sym.sin(theta1 + theta2)
y2 = y1 - sym.cos(theta1 + theta2)
x2d = x2.diff(t)
y2d = y2.diff(t)

x3 = x2 + sym.sin(theta1 + theta2 + theta3)
y3 = y2 - sym.cos(theta1 + theta2 + theta3)
x3d = x3.diff(t)
y3d = y3.diff(t)

# - define state vector
q = sym.Matrix([theta1, theta2, theta3])
qd = q.diff(t)
qdd = qd.diff(t)

# - define KE, U, and Lagrangian of system
#use g = 9.8, not a symbol, from the outset
KE1 = 0.5 * (x1d**2 + y1d**2)
U1 = g * y1

KE2 = 0.5 * (x2d**2 + y2d**2)
U2 = g * y2

KE3 = 0.5 * (x3d**2 + y3d**2)
U3 = g * y3
```

```python
lagrangian5 = (KE1 + KE2 + KE3) - (U1 + U2 + U3)
lagrangian5 = lagrangian5.simplify()

print("Lagrangian:")
display(lagrangian5)
```

Lagrangian:

$$19.62 \cos\left(\theta_1(t) + \theta_2(t)\right) + 1.0 \cos\left(\theta_2(t) + \theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0 \cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0 \cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 9.81 \cos\left(\theta_1(t) + \theta_2(t) + \theta_3(t)\right) + 29.43 \cos\left(\theta_1(t)\right) + 2.0 \cos\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 2.0 \cos\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0 \cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 2.0 \cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0 \cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 1.0 \cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0 \cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 3.0\left(\frac{d}{dt}\theta_1(t)\right)^2 + 3.0\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 1.5\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 0.5\left(\frac{d}{dt}\theta_3(t)\right)^2$$

In [71]:
```python
# - compute non-constrained EL
t0 = time.time()
eqns = compute_EL(lagrangian5, q)
eqns_new = eqns.simplify()

print("Euler-Lagrange equations, simplified:")
display(eqns_new)

tf = time.time()
print(f"Elapsed: {round(tf - t0,1)} seconds")
```

Euler-Lagrange equations, simplified:

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix}
2.0\left(\frac{d}{dt}\theta_2(t) + \frac{d}{dt}\theta_3(t)\right)\sin\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_1(t) + 1.0\left(\frac{d}{dt}\theta_2(t) + \frac{d}{dt}\theta_3(t)\right)\sin\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_2(t) + 1.0\left(\frac{d}{dt}\theta_2(t) + \frac{d}{dt}\theta_3(t)\right)\sin\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_3(t) - 19.62\sin\left(\theta_1(t) + \theta_2(t)\right) - 9.81\sin\left(\theta_1(t) + \theta_2(t) + \theta_3(t)\right) - 29.43\sin\left(\theta_1(t)\right) + 4.0\sin\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 2.0\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 2.0\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 2.0\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 1.0\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 - 2.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 1.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d^2}{dt^2}\theta_2(t) - 1.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d^2}{dt^2}\theta_3(t) - 4.0\cos\left(\theta_2(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 2.0\cos\left(\theta_2(t)\right)\frac{d^2}{dt^2}\theta_2(t) - 2.0\cos\left(\theta_3(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 2.0\cos\left(\theta_3(t)\right)\frac{d^2}{dt^2}\theta_2(t) - 1.0\cos\left(\theta_3(t)\right)\frac{d^2}{dt^2}\theta_3(t) - 6.0\frac{d^2}{dt^2}\theta_1(t) - 3.0\frac{d^2}{dt^2}\theta_2(t) - 1.0\frac{d^2}{dt^2}\theta_3(t) \\
-19.62\sin\left(\theta_1(t) + \theta_2(t)\right) - 1.0\sin\left(\theta_2(t) + \theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 9.81\sin\left(\theta_1(t) + \theta_2(t) + \theta_3(t)\right) - 2.0\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 2.0\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 2.0\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 1.0\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 - 1.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 2.0\cos\left(\theta_2(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 2.0\cos\left(\theta_3(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 2.0\cos\left(\theta_3(t)\right)\frac{d^2}{dt^2}\theta_2(t) - 1.0\cos\left(\theta_3(t)\right)\frac{d^2}{dt^2}\theta_3(t) - 3.0\frac{d^2}{dt^2}\theta_1(t) - 3.0\frac{d^2}{dt^2}\theta_2(t) - 1.0\frac{d^2}{dt^2}\theta_3(t) \\
-1.0\sin\left(\theta_2(t) + \theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 9.81\sin\left(\theta_1(t) + \theta_2(t) + \theta_3(t)\right) - 1.0\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 2.0\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) - 1.0\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 - 1.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 1.0\cos\left(\theta_3(t)\right)\frac{d^2}{dt^2}\theta_1(t) - 1.0\cos\left(\theta_3(t)\right)\frac{d^2}{dt^2}\theta_2(t) - 1.0\frac{d^2}{dt^2}\theta_1(t) - 1.0\frac{d^2}{dt^2}\theta_2(t) - 1.0\frac{d^2}{dt^2}\theta_3(t)
\end{bmatrix}$$

Elapsed: 6.2 seconds

In [72]:
```python
# - solve
to = time.time()

eqns_solved = solve_EL(eqns, qdd)
print(f"Solved at time {round(time.time() - to,1)} seconds")
```

Solved at time 33.6 seconds

In [73]:
```python
print("Solved equations: ")

eqns_new = []
for eq in eqns_solved:
    eq_new = sym.trigsimp(eq)
    display(eq_new)
    eqns_new.append(eq_new)

tf = time.time()
print(f"Elapsed: {round(tf - to,1)} seconds")
```

Solved equations:

$$\frac{d^2}{dt^2}\theta_1(t) = \frac{
4.905\sin\left(\theta_1(t) + 2\theta_2(t)\right) + 1.22625\sin\left(\theta_1(t) - 2\theta_3(t)\right) + 1.22625\sin\left(\theta_1(t) + 2\theta_3(t)\right) + 0.25\sin\left(\theta_2(t) - \theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2
+ 0.5\sin\left(\theta_2(t) - \theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 0.5\sin\left(\theta_2(t) - \theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 0.25\sin\left(\theta_2(t) - \theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 0.5\sin\left(\theta_2(t) - \theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 0.25\sin\left(\theta_2(t) - \theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 + 0.25\sin\left(\theta_2(t) + \theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 0.5\sin\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 0.5\sin\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 0.25\sin\left(\theta_2(t) + \theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 0.5\sin\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 0.25\sin\left(\theta_2(t) + \theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 - 12.2625\sin\left(\theta_1(t)\right) + 1.0\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 2.0\sin\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 0.5\sin\left(2\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2
}{-0.5\cos\left(2\theta_2(t)\right) - 0.25\cos\left(2\theta_3(t)\right) + 1.25}$$

$$\frac{d^2}{dt^2}\theta_2(t) = \frac{
-9.81\sin\left(\theta_1(t) + \theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\cos\left(\theta_2(t)\right) - 9.81\sin\left(\theta_1(t) + \theta_2(t)\right)\cos\left(\theta_2(t)\right) + 2.0\sin\left(\theta_2(t) - \theta_3(t)\right)\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0\sin\left(\theta_2(t) - \theta_3(t)\right)\cos\left(\theta_2(t) + \theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 - 0.375\sin\left(\theta_2(t) - \theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2
- 0.75\sin\left(\theta_2(t) - \theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) - 0.75\sin\left(\theta_2(t) - \theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) - 0.375\sin\left(\theta_2(t) - \theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2
- 0.75\sin\left(\theta_2(t) - \theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) - 0.375\sin\left(\theta_2(t) - \theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 - 0.5\sin\left(\theta_2(t) + \theta_3(t)\right)\sin^2\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2
- 1.0\sin\left(\theta_2(t) + \theta_3(t)\right)\sin^2\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) - 1.0\sin\left(\theta_2(t) + \theta_3(t)\right)\sin^2\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) - 0.5\sin\left(\theta_2(t) + \theta_3(t)\right)\sin^2\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 - 1.0\sin\left(\theta_2(t) + \theta_3(t)\right)\sin^2\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) - 0.5\sin\left(\theta_2(t) + \theta_3(t)\right)\sin^2\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 + 14.715\sin\left(\theta_2(t) + \theta_3(t)\right)\cos\left(\theta_1(t)\right)\cos\left(\theta_3(t)\right) - 0.125\sin\left(\theta_2(t) + 3\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 0.25\sin\left(\theta_2(t) + 3\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) - 0.25\sin\left(\theta_2(t) + 3\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) - 0.125\sin\left(\theta_2(t) + 3\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 - 0.25\sin\left(\theta_2(t) + 3\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) - 0.125\sin\left(\theta_2(t) + 3\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 + 14.715\sin\left(\theta_1(t)\right)\sin^2\left(\theta_3(t)\right) + 14.715\sin\left(\theta_1(t)\right)
+ 2.0\sin^3\left(\theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 1.0\sin^3\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 1.0\sin^2\left(\theta_2(t)\right)\sin^3\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 2.0\sin^2\left(\theta_2(t)\right)\sin^3\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) - 2.0\sin^2\left(\theta_2(t)\right)\sin^3\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) - 1.0\sin^2\left(\theta_2(t)\right)\sin^3\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2
- 2.0\sin^2\left(\theta_2(t)\right)\sin^3\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) - 1.0\sin^2\left(\theta_2(t)\right)\sin^3\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 - 2.0\sin^2\left(\theta_2(t)\right)\sin\left(\theta_3(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.25\sin^2\left(\theta_2(t)\right)\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 2.5\sin^2\left(\theta_2(t)\right)\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 2.5\sin^2\left(\theta_2(t)\right)\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 1.25\sin^2\left(\theta_2(t)\right)\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 2.5\sin^2\left(\theta_2(t)\right)\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t)
+ 1.25\sin^2\left(\theta_2(t)\right)\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 + 9.81\sin\left(\theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\cos\left(\theta_1(t) + \theta_2(t)\right) + 1.0\sin\left(\theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 2.0\sin\left(\theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 2.0\sin\left(\theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 1.0\sin\left(\theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 2.0\sin\left(\theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 1.0\sin\left(\theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 - 2.0\sin\left(\theta_2(t)\right)\sin^2\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 0.25\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t) + 3\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 0.5\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t) + 3\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t)
+ 0.5\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t) + 3\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 0.25\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t) + 3\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 0.5\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t) + 3\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 0.25\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t) + 3\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 - 29.43\sin\left(\theta_2(t)\right)\cos\left(\theta_1(t)\right) - 0.75\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 1.5\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) - 1.5\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) - 0.75\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 - 1.5\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t)
- 0.75\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 - 2.0\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 4.0\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) - 2.0\sin\left(\theta_2(t)\right)\cos\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 - 1.75\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 - 2.0\sin\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t)
- 1.0\sin\left(\theta_2(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 - 0.25\sin\left(3\theta_2(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 0.5\sin^3\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0\sin^3\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t)
+ 1.0\sin^3\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 0.5\sin^3\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0\sin^3\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 0.5\sin^3\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2
- 9.81\sin\left(\theta_3(t)\right)\cos\left(\theta_2(t) + \theta_3(t)\right)\cos\left(\theta_1(t)\right) - 0.5\sin\left(\theta_3(t)\right)\cos\left(3\theta_2(t) + \theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0\sin\left(\theta_3(t)\right)\cos\left(\theta_2(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 0.5\sin\left(\theta_3(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 3.0\sin\left(\theta_3(t)\right)\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.5\sin\left(\theta_3(t)\right)\cos\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 0.125\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2 + 0.25\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 0.25\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t)
+ 0.125\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 0.25\sin\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 0.125\sin\left(\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2 + 0.125\sin\left(3\theta_3(t)\right)\left(\frac{d}{dt}\theta_1(t)\right)^2
+ 0.25\sin\left(3\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 0.25\sin\left(3\theta_3(t)\right)\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 0.125\sin\left(3\theta_3(t)\right)\left(\frac{d}{dt}\theta_2(t)\right)^2 + 0.25\sin\left(3\theta_3(t)\right)\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 0.125\sin\left(3\theta_3(t)\right)\left(\frac{d}{dt}\theta_3(t)\right)^2
}{1.0\sin^2\left(\theta_2(t)\right) + 0.5\sin^2\left(\theta_3(t)\right) + 0.5}$$

$$\frac{d^{2}}{dt^{2}}\theta_{3}(t) = \frac{\begin{gathered}19.62\sin\left(\theta_{1}(t)+\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right) - 2.0\sin\left(\theta_{2}(t)+\theta_{3}(t)\right)\sin^{2}\left(\theta_{2}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 14.715\sin\left(\theta_{2}(t)+\theta_{3}(t)\right)\cos\left(\theta_{1}(t)\right)\cos \\ \left(\theta_{2}(t)\right) - 0.5\sin\left(3\theta_{2}(t)+\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} + 19.62\sin\left(\theta_{1}(t)\right)\sin^{2}\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)+\theta_{3}(t)\right) - 19.62\sin\left(\theta_{1}(t)\right)\sin \\ \left(\theta_{2}(t)\right)\sin\left(\theta_{3}(t)\right) - 19.62\sin\left(\theta_{1}(t)\right)\cos\left(\theta_{2}(t)+\theta_{3}(t)\right) - 2.0\sin^{3}\left(\theta_{2}(t)\right)\sin^{2}\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 19.62\sin^{3}\left(\theta_{2}(t)\right)\cos \\ \left(\theta_{1}(t)+\theta_{2}(t)\right) + 1.0\sin^{3}\left(\theta_{2}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} + 1.0\sin^{2}\left(\theta_{2}(t)\right)\sin^{3}\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} + 2.0\sin^{2}\left(\theta_{2}(t)\right)\sin^{3} \\ \left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) + 2.0\sin^{2}\left(\theta_{2}(t)\right)\sin^{3}\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{3}(t) + 1.0\sin^{2}\left(\theta_{2}(t)\right)\sin^{3}\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} + 2.0\sin^{2} \\ \left(\theta_{2}(t)\right)\sin^{3}\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{2}(t)\frac{d}{dt}\theta_{3}(t) + 1.0\sin^{2}\left(\theta_{2}(t)\right)\sin^{3}\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{3}(t)\right)^{2} + 2.0\sin^{2}\left(\theta_{2}(t)\right)\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{2}(t)\right)\cos \\ \left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 1.25\sin^{2}\left(\theta_{2}(t)\right)\sin\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 2.5\sin^{2}\left(\theta_{2}(t)\right)\sin\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) - 2.5\sin^{2}\left(\theta_{2}(t)\right)\sin \\ \left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{3}(t) - 1.25\sin^{2}\left(\theta_{2}(t)\right)\sin\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} - 2.5\sin^{2}\left(\theta_{2}(t)\right)\sin\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{2}(t)\frac{d}{dt}\theta_{3}(t) - 1.25\sin^{2} \\ \left(\theta_{2}(t)\right)\sin\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{3}(t)\right)^{2} - 1.0\sin\left(\theta_{2}(t)\right)\sin^{2}\left(\theta_{3}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 2.0\sin\left(\theta_{2}(t)\right)\sin^{2}\left(\theta_{3}(t)\right)\cos \\ \left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) - 2.0\sin\left(\theta_{2}(t)\right)\sin^{2}\left(\theta_{3}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{3}(t) - 1.0\sin\left(\theta_{2}(t)\right)\sin^{2} \\ \left(\theta_{3}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} - 2.0\sin\left(\theta_{2}(t)\right)\sin^{2}\left(\theta_{3}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{2}(t)\frac{d}{dt}\theta_{3}(t) - 1.0\sin\left(\theta_{2}(t)\right)\sin^{2} \\ \left(\theta_{3}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{3}(t)\right)^{2} + 2.0\sin\left(\theta_{2}(t)\right)\sin^{2}\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 19.62\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{1}(t)-\theta_{2}(t)\right)\cos \\ \left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right) - 0.25\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)+3\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 0.5\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)+3\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) \\ - 0.5\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)+3\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{3}(t) - 0.25\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)+3\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} - 0.5\sin\left(\theta_{2}(t)\right)\cos \\ \left(\theta_{2}(t)+3\theta_{3}(t)\right)\frac{d}{dt}\theta_{2}(t)\frac{d}{dt}\theta_{3}(t) - 0.25\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)+3\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{3}(t)\right)^{2} + 29.43\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{1}(t)\right) + 0.75\sin \\ \left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} + 1.5\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) + 1.5\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)\right)\cos \\ \left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) + 0.75\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} + 1.5\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{2}(t)\frac{d}{dt}\theta_{3}(t) \\ + 0.75\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{3}(t)\right)^{2} + 1.0\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} + 2.0\sin\left(\theta_{3}(t)\right)\cos \\ \left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) + 1.0\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} + 1.5\sin\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} + 0.75\sin \\ \left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} + 0.25\sin\left(3\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 0.5\sin^{3}\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 1.0\sin^{3}\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) - 1.0\sin^{3} \\ \left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{3}(t) - 0.5\sin^{3}\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} - 1.0\sin^{3}\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{2}(t)\frac{d}{dt}\theta_{3}(t) - 0.5\sin^{3}\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{3}(t)\right)^{2} + 9.81\sin \\ \left(\theta_{3}(t)\right)\cos\left(\theta_{2}(t)+\theta_{3}(t)\right)\cos\left(\theta_{1}(t)\right) + 0.5\sin\left(\theta_{3}(t)\right)\cos\left(3\theta_{2}(t)+\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 9.81\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{1}(t)\right)\cos\left(\theta_{2}(t)\right) \\ - 1.0\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{2}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 0.5\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{2}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 1.0\sin\left(\theta_{3}(t)\right)\cos \\ \left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 2.0\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) - 1.0\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{3}(t) - 1.0\sin\left(\theta_{3}(t)\right)\cos \\ \left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} - 1.0\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{2}(t)\frac{d}{dt}\theta_{3}(t) - 0.5\sin\left(\theta_{3}(t)\right)\cos\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{3}(t)\right)^{2} - 1.125\sin \\ \left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 2.25\sin\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) - 0.25\sin\left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{3}(t) - 1.125\sin\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} - 0.25\sin \\ \left(\theta_{3}(t)\right)\frac{d}{dt}\theta_{2}(t)\frac{d}{dt}\theta_{3}(t) - 0.125\sin\left(\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{3}(t)\right)^{2} - 0.125\sin\left(3\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{1}(t)\right)^{2} - 0.25\sin\left(3\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{2}(t) \\ - 0.25\sin\left(3\theta_{3}(t)\right)\frac{d}{dt}\theta_{1}(t)\frac{d}{dt}\theta_{3}(t) - 0.125\sin\left(3\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{2}(t)\right)^{2} - 0.25\sin\left(3\theta_{3}(t)\right)\frac{d}{dt}\theta_{2}(t)\frac{d}{dt}\theta_{3}(t) - 0.125\sin \\ \left(3\theta_{3}(t)\right)\left(\frac{d}{dt}\theta_{3}(t)\right)^{2}\end{gathered}}{1.0\sin^{2}\left(\theta_{2}(t)\right) + 0.5\sin^{2}\left(\theta_{3}(t)\right) + 0.5}$$

Elapsed: 453.3 seconds

In [74]:

```python
# - make dxdt function
theta1dd_sy = eqns_new[0]
theta2dd_sy = eqns_new[1]
theta3dd_sy = eqns_new[2]

q_ext = sym.Matrix(
    [theta1, theta2, theta3, theta1d, theta2d, theta3d]
)

theta1dd_np = sym.lambdify(q_ext, theta1dd_sy.rhs)
theta2dd_np = sym.lambdify(q_ext, theta2dd_sy.rhs)
theta3dd_np = sym.lambdify(q_ext, theta3dd_sy.rhs)

def dxdt_problem5(t, s):
    '''let state s be a 1x6 vector with t1, t2, t3, t1d, t2d, t3d
    '''
    s = s.tolist()
    s = [float(x) for x in s]
    return np.array([s[3], s[4], s[5], theta1dd_np(*s), theta2dd_np(*s), theta3dd_np(*s)])

t_span = [0,2]
dt = 0.01
ICs = [np.pi/3, np.pi/3, -np.pi/3, 0, 0, 0]
```

In [75]:

```python
# - define impact condition phi
phi = x3

#define function to return the 3 expressions of interest:
#dL_dqdot, dL/dqdot * qdot - L, dphi/dq

def impact_symbolic_eqs(phi, lagrangian, q, q_subs):
    '''Takes the impact condition phi, Lagrangian L, and state vector
    q, and returns the expressions we use to evaluate for impact
    '''
    qd = q.diff(t)

    #define dL_dqdot before substitution
    L_mat = sym.Matrix([lagrangian])
    dL_dqd = L_mat.jacobian(qd)

    #define dPhi/dq before substitution
    phi_mat = sym.Matrix([phi])
    dphi_dq = phi_mat.jacobian(q)

    #define third expression
    dL_dqd_dot_qd = dL_dqd.dot(qd)
    expr = dL_dqd_dot_qd - lagrangian

    expr_a = dL_dqd.subs(q_subs)
    expr_b = dphi_dq.subs(q_subs)
    expr_c = expr.subs(q_subs)

    return [expr_a, expr_b, expr_c]

#create symbolic substitutions for each element in state array
q_ext = sym.Matrix(
    [theta1d, theta2d, theta3d, theta1, theta2, theta3]
)

def gen_sym_subs(q, q_ext):
    '''
    Makes three sets of symbolic variables for use in the impact equations.
    Inputs:
    - q: our state vector. ex: [theta1 theta2 theta3]
    - q_ext: our state vector, plus velocities. must have velocities first.
        ex: [theta1d theta2d theta3d theta1 theta2 theta3]

    Returns:
    - q_subs: a dictionary of state variables and their "q_1" and "qd_1"
        representations for use in calculation of the impact symbolic equations
    - q_taup_subs: a dictionary that can replace "q_1" and "qd_1" with
        "q_1^{tau+}" and "qd_1^{tau+}" for solving for the impact update
    - q_taum_subs: "same as above, but for tau-minus
    '''

    #create symbolic substitutions for each element in state array
    sym_q_only = [sym.symbols(f"q_{i+1}") for i in range(len(q))]
    sym_qd = [sym.symbols(f"qd_{i+1}") for i in range(len(q))]
    sym_q = sym_qd + sym_q_only
    q_subs = {q_ext[i]: sym_q[i] for i in range(len(q_ext))}

    # - Define substitution dicts for q at tau+ and q at tau-,
    qd_taup_vars = [sym.symbols(f"qd_{i+1}^+") for i in range(len(q))]
    qd_taum_vars = [sym.symbols(f"qd_{i+1}^-") for i in range(len(q))]

    q_taup_subs = {sym_q[i] : qd_taup_vars[i] for i in range(len(q))}
    q_taum_subs = {sym_q[i] : qd_taum_vars[i] for i in range(len(q))}

    return q_subs, q_taup_subs, q_taum_subs

q_subs, q_taup_subs, q_taum_subs = gen_sym_subs(q, q_ext)
expr_a, expr_b, expr_c = impact_symbolic_eqs(phi, lagrangian5, q, q_subs)

print("Symbolic expression of dL/dQdot:")
display(expr_a)
print("Symbolic expression of dPhi/dq:")
display(expr_b)
print("Symbolic expression of dL/dQdot * Qdot - L(q,qdot):")
display(expr_c)
```

Symbolic expression of dL/dQdot:

$$\left[\begin{matrix}\begin{gathered}4.0qd_{1}\cos\left(q_{2}\right)+2.0qd_{1}\cos\left(q_{3}\right)+2.0qd_{1}\cos\left(q_{2}+q_{3}\right)+6.0qd_{1}+2.0qd_{2}\cos\left(q_{2}\right)+2.0qd_{2}\cos\left(q_{3}\right)+1.0qd_{2}\cos\left(q_{2}+q_{3}\right)\\+3.0qd_{2}+1.0qd_{3}\cos\left(q_{3}\right)+1.0qd_{3}\cos\left(q_{2}+q_{3}\right)+1.0qd_{3}\end{gathered} & 2.0qd_{1}\cos\left(q_{2}\right)+2.0qd_{1}\cos\left(q_{3}\right)+1.0qd_{1}\cos\left(q_{2}+q_{3}\right)+3.0qd_{1}+2.0qd_{2}\cos\left(q_{3}\right)+3.0qd_{2}+1.0qd_{3}\cos\left(q_{3}\right)+1.0qd_{3} & 1.0qd_{1}\cos\left(q_{3}\right)+1.0qd_{1}\cos\left(q_{2}+q_{3}\right)+1.0qd_{1}+1.0qd_{2}\cos\left(q_{3}\right)+1.0qd_{2}+1.0qd_{3}\end{matrix}\right]$$

Symbolic expression of dPhi/dq:

$$\left[\begin{matrix}\cos\left(q_{1}\right)+\cos\left(q_{1}+q_{2}\right)+\cos\left(q_{1}+q_{2}+q_{3}\right) & \cos\left(q_{1}+q_{2}\right)+\cos\left(q_{1}+q_{2}+q_{3}\right) & \cos\left(q_{1}+q_{2}+q_{3}\right)\end{matrix}\right]$$

Symbolic expression of dL/dQdot * Qdot - L(q,qdot):

$$
\begin{aligned}
&-2.0qd_1^2\cos(q_2) - 1.0qd_1^2\cos(q_3) - 1.0qd_1^2\cos(q_2+q_3) - 3.0qd_1^2 - 2.0qd_1qd_2\cos(q_2) - 2.0qd_1qd_2\cos(q_3) - 1.0qd_1qd_2\cos \\
&(q_2+q_3) - 3.0qd_1qd_2 - 1.0qd_1qd_3\cos(q_3) - 1.0qd_1qd_3\cos(q_2+q_3) - 1.0qd_1qd_3 + qd_1 \\
&\cdot(4.0qd_1\cos(q_2) + 2.0qd_1\cos(q_3) + 2.0qd_1\cos(q_2+q_3) + 6.0qd_1 + 2.0qd_2\cos(q_2) + 2.0qd_2\cos(q_3) + 1.0qd_2\cos(q_2+q_3) \\
&+ 3.0qd_2 + 1.0qd_3\cos(q_3) + 1.0qd_3\cos(q_2+q_3) + 1.0qd_3) - 1.0qd_2^2\cos(q_3) - 1.5qd_2^2 - 1.0qd_2qd_3\cos(q_3) - 1.0qd_2qd_3 \\
&+ qd_2\cdot(2.0qd_1\cos(q_2) + 2.0qd_1\cos(q_3) + 1.0qd_1\cos(q_2+q_3) + 3.0qd_1 + 2.0qd_2\cos(q_3) + 3.0qd_2 + 1.0qd_3\cos(q_3) + 1.0qd_3) \\
&- 0.5qd_3^2 + qd_3\cdot(1.0qd_1\cos(q_3) + 1.0qd_1\cos(q_2+q_3) + 1.0qd_1 + 1.0qd_2\cos(q_3) + 1.0qd_2 + 1.0qd_3) - 29.43\cos(q_1) \\
&- 19.62\cos(q_1+q_2) - 9.81\cos(q_1+q_2+q_3)
\end{aligned}
$$

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).

In [76]:
```python
# subtract the values of elements at tau- from values at tau+
lamb_dphi_dq = lamb * expr_b

dL_dqdot_eqn = \
    expr_a.subs(q_taup_subs) \
    - expr_a.subs(q_taum_subs) \
    - lamb_dphi_dq

hamiltonian_eqn = \
    expr_c.subs(q_taup_subs) \
    - expr_c.subs(q_taum_subs) \

hamiltonian_eqn = hamiltonian_eqn.simplify()

print("dL_dqdot difference:")
display(dL_dqdot_eqn)
print("Hamiltonian difference:")
display(hamiltonian_eqn)
```

```
dL_dqdot difference:
```

$$
\left[\begin{matrix} -\lambda(\cos(q_1) + \cos(q_1+q_2) + \cos(q_1+q_2+q_3)) + 4.0qd_1^+\cos(q_2) + 2.0qd_1^+\cos(q_3) + 2.0qd_1^+\cos(q_2+q_3) + 6.0qd_1^+ - 4.0qd_1^-\cos(q_2) - 2.0qd_1^-\cos(q_3) - 2.0qd_1^-\cos(q_2+q_3) - 6.0qd_1^- + 2.0qd_2^+\cos(q_2) + 2.0qd_2^+\cos(q_3) + 1.0qd_2^+\cos(q_2+q_3) + 3.0qd_2^+ - 2.0qd_2^-\cos(q_2) - 2.0qd_2^-\cos(q_3) - 1.0qd_2^-\cos(q_2+q_3) - 3.0qd_2^- + 1.0qd_3^+\cos(q_3) + 1.0qd_3^+\cos \ldots & -\lambda(\cos(q_1+q_2) + \cos(q_1+q_2+q_3)) + 2.0qd_1^+\cos(q_2) + 2.0qd_1^+\cos(q_3) + 1.0qd_1^+\cos(q_2+q_3) + 3.0qd_1^+ - 2.0qd_1^-\cos(q_2) - 2.0qd_1^-\cos(q_3) - 1.0qd_1^-\cos(q_2+q_3) - 3.0qd_1^- + 2.0qd_2^+\cos(q_3) + 3.0qd_2^+ - 2.0qd_2^-\cos(q_3) - 3.0qd_2^- + 1.0qd_3^+\cos(q_3) + 1.0qd_3^+ - 1.0qd_3^-\cos(q_3) - 1.0qd_3^- & -\lambda\cos(q_1+q_2+q_3) + 1.0qd_1^+\cos(q_3) + 1.0qd_1^+\cos(q_2+q_3) + 1.0qd_1^+ - 1.0qd_1^-\cos(q_3) - 1.0qd_1^-\cos(q_2+q_3) - 1.0qd_1^- + 1.0qd_2^+\cos(q_3) + 1.0qd_2^+ - 1.0qd_2^-\cos(q_3) - 1.0qd_2^- + 1.0qd_3^+ - 1.0qd_3^- \end{matrix}\right]
$$

```
Hamiltonian difference:
```

$$
\begin{aligned}
&2.0\left(qd_1^+\right)^2\cos(q_2) + 1.0\left(qd_1^+\right)^2\cos(q_3) + 1.0\left(qd_1^+\right)^2\cos(q_2+q_3) + 3.0\left(qd_1^+\right)^2 + 2.0qd_1^+qd_2^+\cos(q_2) + 2.0qd_1^+qd_2^+\cos(q_3) \\
&+ 1.0qd_1^+qd_2^+\cos(q_2+q_3) + 3.0qd_1^+qd_2^+ + 1.0qd_1^+qd_3^+\cos(q_3) + 1.0qd_1^+qd_3^+\cos(q_2+q_3) + 1.0qd_1^+qd_3^+ - 2.0\left(qd_1^-\right)^2\cos(q_2) \\
&- 1.0\left(qd_1^-\right)^2\cos(q_3) - 1.0\left(qd_1^-\right)^2\cos(q_2+q_3) - 3.0\left(qd_1^-\right)^2 - 2.0qd_1^-qd_2^-\cos(q_2) - 2.0qd_1^-qd_2^-\cos(q_3) - 1.0qd_1^-qd_2^-\cos \\
&(q_2+q_3) - 3.0qd_1^-qd_2^- - 1.0qd_1^-qd_3^-\cos(q_3) - 1.0qd_1^-qd_3^-\cos(q_2+q_3) - 1.0qd_1^-qd_3^- + 1.0\left(qd_2^+\right)^2\cos(q_3) + 1.5\left(qd_2^+\right)^2 \\
&+ 1.0qd_2^+qd_3^+\cos(q_3) + 1.0qd_2^+qd_3^+ - 1.0\left(qd_2^-\right)^2\cos(q_3) - 1.5\left(qd_2^-\right)^2 - 1.0qd_2^-qd_3^-\cos(q_3) - 1.0qd_2^-qd_3^- + 0.5\left(qd_3^+\right)^2 \\
&- 0.5\left(qd_3^-\right)^2
\end{aligned}
$$

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.

In [88]:
```python
def impact_update_triple_pend(s):
    '''
    Applies the impact update equation, using the two elastic impact equations,
    by plugging in the current values of the state and solving the equations.

    dependencies:
    - dL_dqdot_eqn
    - hamiltonian_eqn
    - variable named "lamb" in main function

    State s contains the current values of the extended state vector.
    Ex: [theta1 theta2 theta3 theta1d theta2d theta3d]
    '''
    q_taum_vars = [sym.symbols(f"q_{i+1}") for i in range(len(q))]
    qd_taum_vars = [sym.symbols(f"qd_{i+1}^-") for i in range(len(q))]
    qd_taup_vars = [sym.symbols(f"qd_{i+1}^+") for i in range(len(q))]

    q_vars = q_taum_vars + qd_taum_vars

    state_subs = {q_vars[i] : s[i] for i in range(len(s))}
    dL_dqdot_new = dL_dqdot_eqn.subs(state_subs)
    hamiltonian_new = hamiltonian_eqn.subs(state_subs)
    hamiltonian_new = hamiltonian_new.simplify()

    #set up equations to be solved
    #insert the hamiltonian at the (n)th row in 0-based indexing, i.e. add onto end of matrix
    eqns_matrix = dL_dqdot_new.T.row_insert(\
        len(q), sym.Matrix([hamiltonian_new]))

    print("Matrix of equations to solve:")
    display(eqns_matrix)

    #solve for the values of qdot and lambda
    sol_vars = qd_taup_vars
    sol_vars.append(lamb)

    solns = sym.solve(eqns_matrix, sol_vars, dict = True)
    eqns_solved = []
    state_new = s[0:3]

    for i, sol in enumerate(solns):

        #do some error checking - if lambda = 0, not valid
        if np.isclose( float(sol[lamb]) , 0):
            continue

        print(sol[lamb])

        for x in list(sol.keys()):

            if x == lamb: continue
            state_new = np.append(state_new, sol[x])

    return state_new
```

In [89]:
```python
s0 = [0,0,0,0,0,-1]
ans = impact_update_triple_pend(s0)
print("Test of impact update:")
print(ans) #expected: +1
```

```
Matrix of equations to solve:
```

$$
\left[\begin{matrix} -3\lambda + 14.0qd_1^+ + 8.0qd_2^+ + 3.0qd_3^+ + 3.0 \\ -2\lambda + 8.0qd_1^+ + 5.0qd_2^+ + 2.0qd_3^+ + 2.0 \\ -\lambda + 3.0qd_1^+ + 2.0qd_2^+ + 1.0qd_3^+ + 1.0 \\ 7.0\left(qd_1^+\right)^2 + 8.0qd_1^+qd_2^+ + 3.0qd_1^+qd_3^+ + 2.5\left(qd_2^+\right)^2 + 2.0qd_2^+qd_3^+ + 0.5\left(qd_3^+\right)^2 - 0.5 \end{matrix}\right]
$$

```
2.00000000000000
Test of impact update:
[0 0 0 0.0 0.0 1.00000000000000]
```

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!

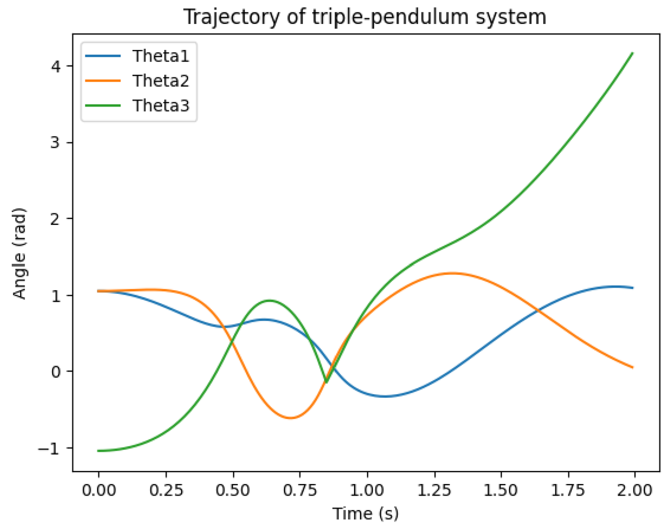
Turn in: A copy of code for the impact update function and simulate function, as well as code output including the plot of simulated trajectory and the animation. The video should be uploaded separately from the .pdf file through Canvas, and it should be in ".mp4" format. You can use screen capture or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

In [79]:
```python
#define the state of the constraint function before impact

# display(phi)
q_ext = [theta1, theta2, theta3, theta1d, theta2d, theta3d]
phi_np = sym.lambdify(q_ext, phi)
# help(phi_np)

def impact_condition_triple_pend(s):
    '''Use the "threshold" method for detecting collisions.
    '''
    thres = 0.1
    return( phi_np(*s) < thres and phi_np(*s) > -thres )

# apply impact condition and update to the simulate() function
t_span = [0,2]
dt = 0.01
ICs = [np.pi/3, np.pi/3, -np.pi/3, 0, 0, 0]

traj_array = simulate_impact(t_span, dt, ICs, rk4, dxdt_problem5, impact_condition_triple_pend, impact_update_triple_pend)
```

In [92]:
```python
#plot trajectory
t_array = np.arange(t_span[0], t_span[1], dt)
plt.figure()
plt.plot(t_array, traj_array[0], label="Theta1")
plt.plot(t_array, traj_array[1], label="Theta2")
plt.plot(t_array, traj_array[2], label="Theta3")

plt.xlabel("Time (s)")
plt.ylabel("Angle (rad)")
```

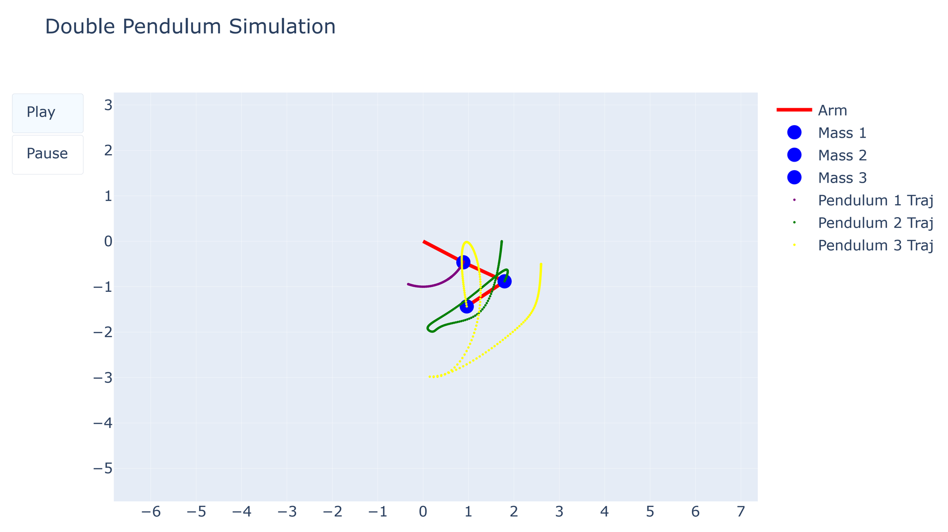
```
plt.title("Trajectory of triple-pendulum system")
plt.legend()
```

Out[92]: `<matplotlib.legend.Legend at 0x2249dd7a470>`



In [80]:
```
# animate the trajectory
theta_array = traj_array[0:3]
animate_triple_pend(theta_array)
```

3



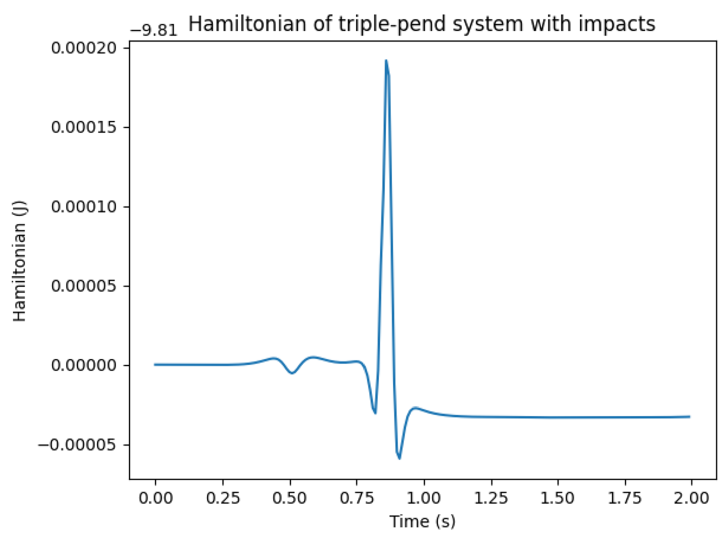
Problem 9 (5pts)

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.

In [82]:
```
def H(s):
    '''Let R1 = R2 = R3 = 1, m1 = m2 = m3 = 1, g = 9.8
    '''
    [t1, t2, t3, t1d, t2d, t3d] = s

    x1 = np.sin(t1)
    y1 = -np.cos(t1)
    x1d = np.cos(t1) * t1d
    y1d = np.sin(t1) * t1d

    x2 = x1 + np.sin(t1 + t2)
    y2 = y1 - np.cos(t1 + t2)
    x2d = x1d + np.cos(t1 + t2) * (t1d + t2d)
    y2d = y1d + np.sin(t1 + t2) * (t1d + t2d)

    x3 = x2 + np.sin(t1 + t2 + t3)
    y3 = y2 - np.cos(t1 + t2 + t3)
    x3d = x2d + np.cos(t1 + t2 + t3) * (t1d + t2d + t3d)
    y3d = y2d + np.sin(t1 + t2 + t3) * (t1d + t2d + t3d)

    KE1 = 0.5 * (x1d**2 + y1d**2)
    KE2 = 0.5 * (x2d**2 + y2d**2)
    KE3 = 0.5 * (x3d**2 + y3d**2)

    U1 = 9.81 * y1
    U2 = 9.81 * y2
    U3 = 9.81 * y3

    return (KE1 + KE2 + KE3) + (U1 + U2 + U3)

t_array = np.arange(t_span[0], t_span[1], dt)
ham_array = [H(s) for s in traj_array.T]

#plot
plt.figure()
plt.plot(t_array, ham_array)
plt.xlabel("Time (s)")
plt.ylabel("Hamiltonian (J)")
plt.title("Hamiltonian of triple-pend system with impacts")
```

Out[82]: `Text(0.5, 1.0, 'Hamiltonian of triple-pend system with impacts')`



Variation in the Hamiltonian spikes at the point of impact, on the scale of 2 * 10^-4. After the impact, the Hamiltonian has decreased by about 2 * 10^-5. The variation in the Hamiltonian post-impact is small enough where we can consider it a result of the numerical integration behavior of the system.

In [91]:
```
#values of spikes
max = np.max(ham_array) + 9.81
min = np.min(ham_array) + 9.81

print(f"Max and min error: {max} {min}")
```

```
Max and min error: 0.00019161474707196646 -5.907130546667361e-05
```