HW5

November 1, 2023

1 HW 5

Qingyuan Chen

[1]:
```python
#IMPORT ALL NECESSARY PACKAGES AT THE TOP OF THE CODE
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt

# Custom display helpers
from IPython.display import Markdown

def md_print(md_str: str):
    display(Markdown(md_str))

def lax_eq(equation):
    return sym.latex(equation , mode='inline')
```

[2]:
```python
import sympy as sym
def get_eu_la(L: sym.Function, q: sym.Matrix, t: sym.symbols):
    """Generate euler lagrangian using sympy jacobian

    Args:
        L (sym.Function): Lagrangian equation
        q (sym.Matrix): matrix of system-var q
        t (sym.symbols): time symbol (needed for q.diff(t))
    """

    q_dot = q.diff(t)
    dL_dq = sym.simplify(sym.Matrix([L]).jacobian(q).T)
    dL_dq_dot = sym.simplify(sym.Matrix([L]).jacobian(q_dot).T)

    return sym.simplify(dL_dq_dot.diff(t) - dL_dq)


def solve_and_print(variables: sym.Matrix,
                    eu_la_eq: sym.Eq , quiet = False) -> list[dict[any]]:
    """Solve the given eu_la equation
```

```
    Args:
        variables (sym.Matrix): var to solve for
        eu_la_eq (sym.Eq): eu_la equation
        quiet (bool): turn off any printing if True

    Returns:
        list[dict[sym.Function]]: list of solution dicts (keyed with variables)
    """
    solution_dicts = sym.solve(eu_la_eq, variables, dict=True)
    i = 0
    print(f"Total of {len(solution_dicts)} solutions")
    for solution_dict in solution_dicts:
        i += 1
        if not quiet: md_print(f"solution : {i} / {len(solution_dicts)}")
        for var in variables:
            sol = solution_dict[var]
            if not quiet: md_print(f"{lax_eq(var)} = {lax_eq(sym.
 ↪simplify(sol))}")
    return solution_dicts

def lambdify_sys(var_list: list, function_dict: dict[any, sym.Function], keys␣
 ↪=None):
    lambda_dict ={}
    if keys is None:
        keys = function_dict.keys()
    for var in keys:
        acceleration_function = (function_dict[var])
        lambda_func = sym.lambdify(var_list, acceleration_function )
        lambda_dict[var] = lambda_func
    return lambda_dict

def make_system_equation_2(lambda_dict,lam_keys):
    def system_equation(state , lambda_dict ,lam_keys):
        '''
        argumetn:
        state -> array of 4 item, x,theta,xdot ,thetadot
        return -> array of 4 item, xdot, thetadot, xddot, thetaddot
        '''
        q_1 = state[0]
        v_1 = state[1]
        a_1 =  lambda_dict[lam_keys[0]](q_1,v_1)

        return np.array([v_1,a_1])

    return lambda state: system_equation(state , lambda_dict , lam_keys)
```

```python
def make_system_equation_6(lambda_dict,lam_keys):
    def system_equation(state , lambda_dict ,lam_keys):
        '''
        argumetn:
        state -> array of 4 item, x,theta,xdot ,thetadot
        return -> array of 4 item, xdot, thetadot, xddot, thetaddot
        '''
        q_1 = state[0]
        q_2 =state[1]
        q_3 =state[2]
        v_1 = state[3]
        v_2 = state[4]
        v_3 = state[5]
        a_1 =  lambda_dict[lam_keys[0]](q_1,q_2,q_3,v_1,v_2 ,v_3)
        a_2 = lambda_dict[lam_keys[1]](q_1,q_2,q_3,v_1,v_2,v_3)
        a_3 = lambda_dict[lam_keys[2]](q_1,q_2,q_3,v_1,v_2,v_3)

        return np.array([v_1, v_2, v_3, a_1, a_2, a_3])

    return lambda state: system_equation(state , lambda_dict , lam_keys)
```

[3]:
```python
# Integrate and simulate

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)
```

```python
    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 tvec , xtraj
```

[4]:
```python
# Simulation functions

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

```python
    """
    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='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
```

```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)
```

1.1 Problem 1

[5]:
```
# p1
t= sym.symbols('t')

theta = sym.Function(r'\theta')(t)
theta_dot = theta.diff(t)

q = sym.Matrix([theta])
q_dot = q.diff(t)
q_ddot = q_dot.diff(t)
R =1
m = 1
g=9.8

# x points right,
# y points down
x = sym.sin(theta) *R
y = sym.cos(theta) *R

KE = 1/2 * m * (x.diff(t)**2 + y.diff(t)**2)
```

```python
PE = -m*g*y
L = sym.simplify(KE - PE)

p1_eu_la = get_eu_la(L , q ,t )
md_print("### Problem 1 solution")
p1_eu_la_sol = solve_and_print(q_ddot,
                               eu_la_eq=sym.Eq(p1_eu_la, sym.Matrix([0])),
                               quiet=False)[0]

lambda_dict = lambdify_sys([theta , theta.diff(t)] , p1_eu_la_sol , q_ddot)
p1_system_equation = make_system_equation_2(lambda_dict , q_ddot)

t_range = [0,5]
dt = 0.01
q0 = np.array([np.pi/2 , 0])
p1_tvec,p1_traj = simulate(p1_system_equation ,q0 , t_range , 0.01 , integrate)

plt.figure(1)
plt.plot(p1_tvec , p1_traj[0] , label = "theta")
plt.xlabel("time")
plt.ylabel("value")
plt.legend()
plt.plot()
```

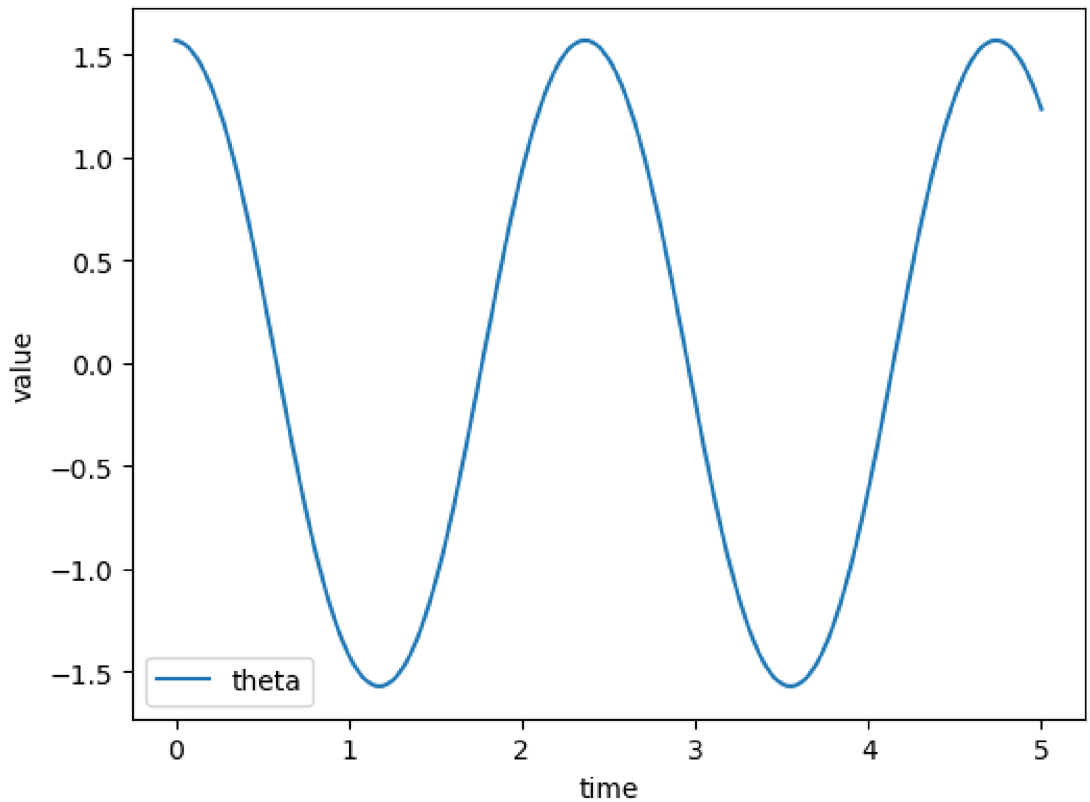
1.1.1 Problem 1 solution

```
Total of 1 solutions
```

solution : 1 / 1

$\frac{d^2}{dt^2}\theta(t) = -9.8\sin(\theta(t))$

```
[5]: []
```



1.2 Problem 2

[6]:
```
# p2

lamb = sym.symbols(r"\lambda")
# impact function
phi = theta - 0  # the same as x-0
# display(phi)

# dummy var
theta_minus, theta_dot_minus, theta_dot_plus = sym.symbols(r"\theta^- ,␣
↪\dot{\theta}^- , \dot{\theta}^+")
# display(theta_dot_minus)

# Construct the dl/dqdot with dummy tau plus/minus
dL_dqdot = sym.Matrix([L]).jacobian(q_dot)

sub_minus = {theta: theta_minus, theta_dot: theta_dot_minus}
sub_plus = {theta: theta_minus, theta_dot: theta_dot_plus}
dL_dqdot_minus = dL_dqdot.subs(sub_minus)
dL_dqdot_plus = dL_dqdot.subs(sub_plus )
```

```
md_print(f"### Problem 2 solution")
md_print(f"dL/dq_dot = \n\n {lax_eq(dL_dqdot[0].expand())}")

dphi_dq = sym.Matrix([phi]).jacobian(q)

md_print(f"dphi/dq = \n\n {lax_eq(dphi_dq)}")
# This is still needed because the other eqs doesn't have q anymore
dphi_dq_minus = dphi_dq.subs(sub_minus)
# display(dphi_dq)

H = (dL_dqdot * q_dot)[0] - L
H_plus = H.subs(sub_plus)
H_minus = H.subs(sub_minus)
md_print(f"H: (dL/dq_dot * qdot - L) = \n\n {lax_eq(H.expand())}")
```

1.2.1 Problem 2 solution

dL/dq_dot =

$1.0\frac{d}{dt}\theta(t)$

dphi/dq =

$\left[1\right]$

H: (dL/dq_dot * qdot - L) =

$-9.8\cos\left(\theta(t)\right) + 0.5\left(\frac{d}{dt}\theta(t)\right)^2$

1.3 Problem 3 solution

[7]:
```
# p3

impact_sys_lhs = sym.Matrix([dL_dqdot_plus[0] - dL_dqdot_minus[0] , H_plus -␣
↪H_minus] )
impact_sys_rhs = sym.Matrix([lamb * dphi_dq_minus[0] , 0 ])
impact_equation = sym.Eq(impact_sys_lhs, impact_sys_rhs)

display(impact_equation)

impact_q = [theta_dot_plus , lamb]
md_print("### Solutions for impact update")
impact_solved = solve_and_print(impact_q , impact_equation )

good_solution = None
for sol in impact_solved:
    if sol[lamb] !=0:
        good_solution = sol
```

```
if good_solution is None:
    raise RuntimeError("No impact solution")

# We actually don't care about lambdafy the lambda
impact_plus_lambda_dict = lambdify_sys([theta_minus, theta_dot_minus],␣
↪good_solution,
                                       [theta_dot_plus])

def p3_impact_update(states:list[float,float]):
    q_dot_plus = impact_plus_lambda_dict[theta_dot_plus](states[0] , states[1])
    return [states[0] ,q_dot_plus]

# testing

md_print(f"""
### numerically evaluate the impact update
[ $q(\\tau^+)$ , $\\dot{{q}}(\\tau^-)$ ]
{p3_impact_update([0.01 , 2])}
""")
```

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

1.3.1 Solutions for impact update

Total of 2 solutions

solution : 1 / 2

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

$\lambda = -2.0\dot{\theta}^{-}$

solution : 2 / 2

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

$\lambda = 0$

1.3.2 numerically evaluate the impact update

[$q(\tau^+)$, $\dot{q}(\tau^-)$] [0.01, -2]

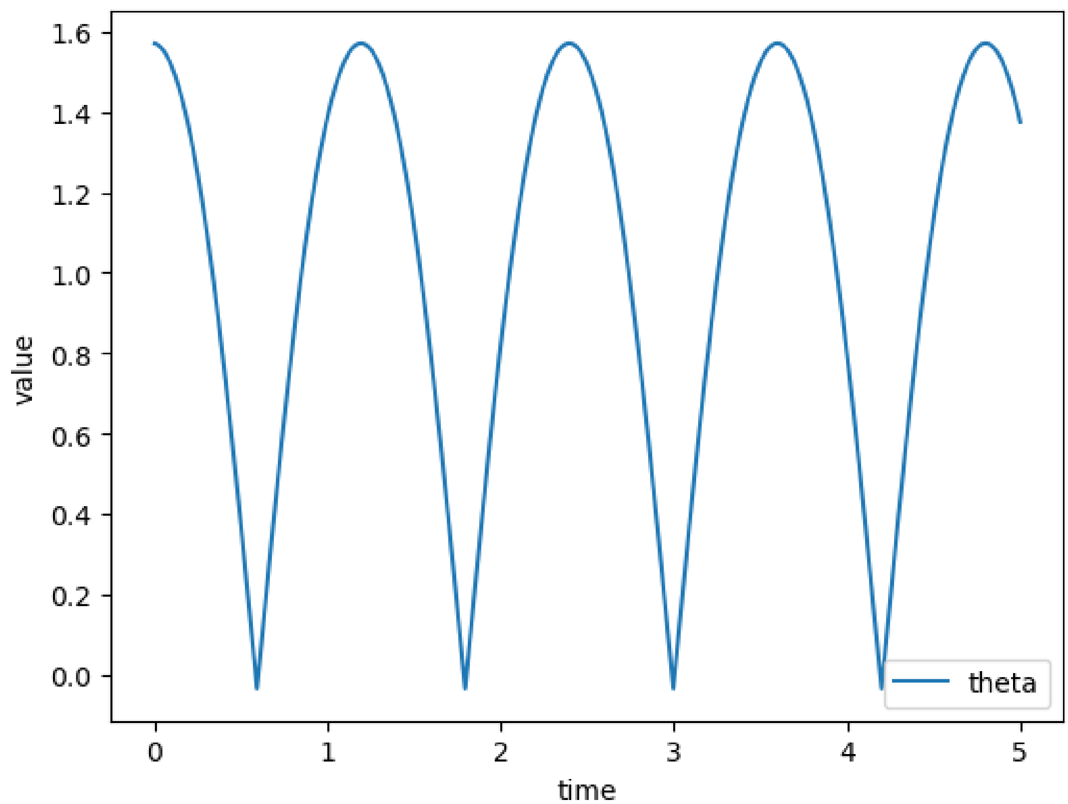
1.4 Problem 4 solution

[8]:
```
phi_lambda = sym.lambdify([theta , theta_dot] , phi)


def p4_get_constrain_phi_value(s)->float:
    return phi_lambda(*s)
def p4_impact_condition(s:list[float,float] , initial_sign)->bool:
```

```python
    """Check if impact has happened
    """
    # if phi has changed sign.
    return p4_get_constrain_phi_value(s) * initial_sign <0




def p4_simulate_impact(f, x0, tspan, dt, integrate):
    """
    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))
    phi_0 = p4_get_constrain_phi_value(x0)

    for i in range(N):
        if p4_impact_condition( x , phi_0):
            x = p3_impact_update(x)

        xtraj[:,i]=integrate(f,x,dt)
        x = np.copy(xtraj[:,i])
    return tvec , xtraj


p4_tvec,p4_traj = p4_simulate_impact(p1_system_equation ,q0 , t_range , 0.01 ,
 ↪integrate)
```

```
plt.figure(1)
plt.plot(p4_tvec , p4_traj[0] , label = "theta")
plt.xlabel("time")
plt.ylabel("value")
plt.legend()
plt.plot()

animate_single_pend(p4_traj,L1=1,T=5)
```

<IPython.core.display.HTML object>



1.5 Problem 5

[9]:
```
# p5

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

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

R1=1
R2=1
R3=1
m1= 1
m2= 1
m3= 1
g=9.8


x1 = sym.sin(theta1)*R1
y1 = - sym.cos(theta1)*R1
x2 = x1 + sym.sin(theta1+theta2)*R2
y2 = y1 - sym.cos(theta1+theta2)*R2
x3 = x2 + sym.sin(theta1+theta2+theta3)*R2
y3 = y2 - sym.cos(theta1+theta2+theta3)*R2

KE = 1 / 2 * m1 * (x1.diff(t)**2 + y1.diff(t)**2) + 1 / 2 * m1 * (
    x2.diff(t)**2 + y2.diff(t)**2) + 1 / 2 * m1 * (x3.diff(t)**2 + y3.
 ↪diff(t)**2)

PE = m1*g*y1 + m2*g*y2 + m3*g*y3

L = sym.simplify(KE - PE)

print("solved_eu_la:")
p5_eu_la = get_eu_la(L , q ,t )
p5_eu_la_sol = solve_and_print(q_ddot,
                               eu_la_eq=sym.Eq(p5_eu_la, sym.Matrix([0,0,0])),
                               quiet=False)[0]

p5_lambda_dict = lambdify_sys([*q, *q_dot] , p5_eu_la_sol , q_ddot)
p5_system_equation = make_system_equation_6(p5_lambda_dict , q_ddot)
```

```
solved_eu_la:
Total of 1 solutions
```

solution : 1 / 1

$$\frac{d^2}{dt^2}\theta_1(t) = \frac{-49.0\sin(\theta_1(t)+2\theta_2(t))-12.25\sin(\theta_1(t)-2\theta_3(t))-12.25\sin(\theta_1(t)+2\theta_3(t))-2.5\sin(\theta_2(t)-\theta_3(t))\left(\frac{d}{dt}\theta_1(t)\right)^2-5.0\sin(\theta_2(t)-\theta_3(t))\frac{d}{dt}\theta_1(t)}{}$$

$$\frac{d^2}{dt^2}\theta_2(t) = \frac{-2.5(-\sin(\theta_2(t)-2\theta_3(t))+\sin(\theta_2(t)+2\theta_3(t)))\left(\frac{d}{dt}\theta_1(t)\right)^2-5.0(-\sin(\theta_2(t)-2\theta_3(t))+\sin(\theta_2(t)+2\theta_3(t)))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t)-2.5(-\sin(\theta_2(t)-2}{}$$

$$\frac{d^2}{dt^2}\theta_3(t) = \frac{5.0(\sin(\theta_2(t)-\theta_3(t))+\sin(3\theta_2(t)+\theta_3(t)))\left(\frac{d}{dt}\theta_1(t)\right)^2-2.5(\sin(\theta_2(t)+2\theta_3(t))+\sin(3\theta_2(t)+2\theta_3(t)))\left(\frac{d}{dt}\theta_1(t)\right)^2-5.0(\sin(2\theta_2(t)-\theta_3(t))+\sin(2}{}$$

[10]:

```python
# p5 continue
lamb = sym.symbols(r"\lambda")

phi = x3 -0
display(phi)
theta1_dot_plus, theta2_dot_plus, theta3_dot_plus = sym.
↪symbols(r"\dot{\theta}_1^+ , \dot{\theta}_2^+ , \dot{\theta}_3^+")
theta1_minus, theta2_minus, theta3_minus = sym.symbols(r"\theta_1^- ,␣
↪\theta_2^- , \theta_3^-")
theta1_dot_minus, theta2_dot_minus, theta3_dot_minus = sym.
↪symbols(r"\dot{\theta}_1^- , \dot{\theta}_2^- , \dot{\theta}_3^-")

theta_minus, theta_dot_minus, theta_dot_plus = sym.symbols(r"\theta^- ,␣
↪\dot{\theta}^- , \dot{\theta}^+")

subs_minus = {
    theta1: theta1_minus ,
    theta2: theta2_minus ,
    theta3: theta3_minus ,
    theta1.diff(t): theta1_dot_minus ,
    theta2.diff(t): theta2_dot_minus ,
    theta3.diff(t): theta3_dot_minus ,
}

subs_plus = {
    theta1: theta1_minus ,
    theta2: theta2_minus ,
    theta3: theta3_minus ,
    theta1.diff(t): theta1_dot_plus ,
    theta2.diff(t): theta2_dot_plus ,
    theta3.diff(t): theta3_dot_plus ,
}

dL_dqdot = sym.Matrix([L]).jacobian(q_dot)
dL_dqdot_minus = dL_dqdot.subs(subs_minus)
dL_dqdot_plus = dL_dqdot.subs(subs_plus )
md_print("### Problem 5 solution")
md_print(f"**dL/dq_dot**")

for i , func in zip(range(len(dL_dqdot)),dL_dqdot):
    print(f"for theta_{i}")
    display(func)

dphi_dq = sym.Matrix([phi]).jacobian(q)
md_print(f"**dphi/dq**")
for i , func in zip(range(len(dphi_dq)),dphi_dq):
    print(f"for theta_{i}")
```

```
    display(func)



dphi_dq_minus = dphi_dq.subs(subs_minus)

H = (dL_dqdot * q_dot)[0] - L
H_plus = H.subs(subs_plus)
H_minus = H.subs(subs_minus)

md_print(f"**H: (dL/dq_dot * qdot - L)**: \n\n {lax_eq(sym.expand( H))}")
```

$\sin\left(\theta_1(t) + \theta_2(t)\right) + \sin\left(\theta_1(t) + \theta_2(t) + \theta_3(t)\right) + \sin\left(\theta_1(t)\right)$

1.5.1 Problem 5 solution

dL/dq_dot

`for theta_0`

$2.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_1(t) + 1.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_2(t) + 1.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_3(t) + 4.0\cos\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t) + 2.0\cos\left(\theta_2(t)\right)\frac{d}{dt}\theta_2(t) + 2.0\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t) + 2.0\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t) + 1.0\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_3(t) + 6.0\frac{d}{dt}\theta_1(t) + 3.0\frac{d}{dt}\theta_2(t) + 1.0\frac{d}{dt}\theta_3(t)$

`for theta_1`

$1.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_1(t) + 2.0\cos\left(\theta_2(t)\right)\frac{d}{dt}\theta_1(t) + 2.0\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t) + 2.0\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t) + 1.0\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_3(t) + 3.0\frac{d}{dt}\theta_1(t) + 3.0\frac{d}{dt}\theta_2(t) + 1.0\frac{d}{dt}\theta_3(t)$

`for theta_2`

$1.0\cos\left(\theta_2(t) + \theta_3(t)\right)\frac{d}{dt}\theta_1(t) + 1.0\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_1(t) + 1.0\cos\left(\theta_3(t)\right)\frac{d}{dt}\theta_2(t) + 1.0\frac{d}{dt}\theta_1(t) + 1.0\frac{d}{dt}\theta_2(t) + 1.0\frac{d}{dt}\theta_3(t)$

dphi/dq

`for theta_0`

$\cos\left(\theta_1(t) + \theta_2(t)\right) + \cos\left(\theta_1(t) + \theta_2(t) + \theta_3(t)\right) + \cos\left(\theta_1(t)\right)$

`for theta_1`

$\cos\left(\theta_1(t) + \theta_2(t)\right) + \cos\left(\theta_1(t) + \theta_2(t) + \theta_3(t)\right)$

`for theta_2`

$\cos\left(\theta_1(t) + \theta_2(t) + \theta_3(t)\right)$

H: (dL/dq_dot * qdot - L):

$-19.6\cos(\theta_1(t)+\theta_2(t)) + 1.0\cos(\theta_2(t)+\theta_3(t))\left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0\cos(\theta_2(t)+\theta_3(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0\cos(\theta_2(t)+\theta_3(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) - 9.8\cos(\theta_1(t)+\theta_2(t)+\theta_3(t)) - 29.4\cos(\theta_1(t)) + 2.0\cos(\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2 + 2.0\cos(\theta_2(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0\cos(\theta_3(t))\left(\frac{d}{dt}\theta_1(t)\right)^2 + 2.0\cos(\theta_3(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0\cos(\theta_3(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 1.0\cos(\theta_3(t))\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0\cos(\theta_3(t))\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 3.0\left(\frac{d}{dt}\theta_1(t)\right)^2 + 3.0\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t) + 1.0\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_3(t) + 1.5\left(\frac{d}{dt}\theta_2(t)\right)^2 + 1.0\frac{d}{dt}\theta_2(t)\frac{d}{dt}\theta_3(t) + 0.5\left(\frac{d}{dt}\theta_3(t)\right)^2$

1.6 Problem 6

[15]:
```
impact_sys_lhs = sym.Matrix([
    dL_dqdot_plus[0] - dL_dqdot_minus[0] ,
    dL_dqdot_plus[1] - dL_dqdot_minus[1],
    dL_dqdot_plus[2] - dL_dqdot_minus[2],
    H_plus - H_minus] )
impact_sys_rhs = sym.Matrix([
    lamb * dphi_dq_minus[0] ,
    lamb * dphi_dq_minus[1] ,
    lamb * dphi_dq_minus[2] ,
    0 ])



impact_equation = sym.Eq(impact_sys_lhs, impact_sys_rhs)

display(sym.expand(impact_equation))
md_print("### Problem 6 solution \n\n printed in an expanded version")
for i in range(4):
    print(f"equation {i}")
    md_print(f"{lax_eq(impact_equation.lhs[i].expand())} =
 ↪{lax_eq(impact_equation.rhs[i].expand())}")
```

$4.0\dot{\theta}_1^+$

$2.0\left(\dot{\theta}_1^+\right)^2\cos(\theta_2^-) + 1.0\left(\dot{\theta}_1^+\right)^2\cos(\theta_3^-) + 1.0\left(\dot{\theta}_1^+\right)^2\cos(\theta_2^- + \theta_3^-) + 3.0\left(\dot{\theta}_1^+\right)^2 + 2.0\dot{\theta}_1^+\dot{\theta}_2^+\cos(\theta_2^-) + 2.0\dot{\theta}_1^+\dot{\theta}_2^+\cos(\theta_3^-)$

$$\begin{bmatrix}\lambda\cos(\theta_1^-) + \lambda\cos(\theta_1^- + \theta_2^-) + \lambda\cos(\theta_1^- + \theta_2^- + \theta_3^-)\\ \lambda\cos(\theta_1^- + \theta_2^-) + \lambda\cos(\theta_1^- + \theta_2^- + \theta_3^-)\\ \lambda\cos(\theta_1^- + \theta_2^- + \theta_3^-)\\ 0\end{bmatrix}$$

1.6.1 Problem 6 solution

printed in an expanded version

```
equation 0
```

$4.0\dot{\theta}_1^+\cos(\theta_2^-) + 2.0\dot{\theta}_1^+\cos(\theta_3^-) + 2.0\dot{\theta}_1^+\cos(\theta_2^- + \theta_3^-) + 6.0\dot{\theta}_1^+ - 4.0\dot{\theta}_1^-\cos(\theta_2^-) - 2.0\dot{\theta}_1^-\cos(\theta_3^-) - 2.0\dot{\theta}_1^-\cos(\theta_2^- + \theta_3^-) - 6.0\dot{\theta}_1^- + 2.0\dot{\theta}_2^+\cos(\theta_2^-) + 2.0\dot{\theta}_2^+\cos(\theta_3^-) + 1.0\dot{\theta}_2^+\cos(\theta_2^- + \theta_3^-) + 3.0\dot{\theta}_2^+ -$

$2.0\dot{\theta}_2^- \cos(\theta_2^-) - 2.0\dot{\theta}_2^- \cos(\theta_3^-) - 1.0\dot{\theta}_2^- \cos(\theta_2^- + \theta_3^-) - 3.0\dot{\theta}_2^- + 1.0\dot{\theta}_3^+ \cos(\theta_3^-) + 1.0\dot{\theta}_3^+ \cos(\theta_2^- + \theta_3^-) + 1.0\dot{\theta}_3^+ - 1.0\dot{\theta}_3^- \cos(\theta_3^-) - 1.0\dot{\theta}_3^- \cos(\theta_2^- + \theta_3^-) - 1.0\dot{\theta}_3^- = \lambda \cos(\theta_1^-) + \lambda \cos(\theta_1^- + \theta_2^-) + \lambda \cos(\theta_1^- + \theta_2^- + \theta_3^-)$

```
equation 1
```

$2.0\dot{\theta}_1^+ \cos(\theta_2^-) + 2.0\dot{\theta}_1^+ \cos(\theta_3^-) + 1.0\dot{\theta}_1^+ \cos(\theta_2^- + \theta_3^-) + 3.0\dot{\theta}_1^+ - 2.0\dot{\theta}_1^- \cos(\theta_2^-) - 2.0\dot{\theta}_1^- \cos(\theta_3^-) - 1.0\dot{\theta}_1^- \cos(\theta_2^- + \theta_3^-) - 3.0\dot{\theta}_1^- + 2.0\dot{\theta}_2^+ \cos(\theta_3^-) + 3.0\dot{\theta}_2^+ - 2.0\dot{\theta}_2^- \cos(\theta_3^-) - 3.0\dot{\theta}_2^- + 1.0\dot{\theta}_3^+ \cos(\theta_3^-) + 1.0\dot{\theta}_3^+ - 1.0\dot{\theta}_3^- \cos(\theta_3^-) - 1.0\dot{\theta}_3^- = \lambda \cos(\theta_1^- + \theta_2^-) + \lambda \cos(\theta_1^- + \theta_2^- + \theta_3^-)$

```
equation 2
```

$1.0\dot{\theta}_1^+ \cos(\theta_3^-) + 1.0\dot{\theta}_1^+ \cos(\theta_2^- + \theta_3^-) + 1.0\dot{\theta}_1^+ - 1.0\dot{\theta}_1^- \cos(\theta_3^-) - 1.0\dot{\theta}_1^- \cos(\theta_2^- + \theta_3^-) - 1.0\dot{\theta}_1^- + 1.0\dot{\theta}_2^+ \cos(\theta_3^-) + 1.0\dot{\theta}_2^+ - 1.0\dot{\theta}_2^- \cos(\theta_3^-) - 1.0\dot{\theta}_2^- + 1.0\dot{\theta}_3^+ - 1.0\dot{\theta}_3^- = \lambda \cos(\theta_1^- + \theta_2^- + \theta_3^-)$

```
equation 3
```

$2.0\left(\dot{\theta}_1^+\right)^2 \cos(\theta_2^-) + 1.0\left(\dot{\theta}_1^+\right)^2 \cos(\theta_3^-) + 1.0\left(\dot{\theta}_1^+\right)^2 \cos(\theta_2^- + \theta_3^-) + 3.0\left(\dot{\theta}_1^+\right)^2 + 2.0\dot{\theta}_1^+\dot{\theta}_2^+ \cos(\theta_2^-) + 2.0\dot{\theta}_1^+\dot{\theta}_2^+ \cos(\theta_3^-) + 1.0\dot{\theta}_1^+\dot{\theta}_2^+ \cos(\theta_2^- + \theta_3^-) + 3.0\dot{\theta}_1^+\dot{\theta}_2^+ + 1.0\dot{\theta}_1^+\dot{\theta}_3^+ \cos(\theta_3^-) + 1.0\dot{\theta}_1^+\dot{\theta}_3^+ \cos(\theta_2^- + \theta_3^-) + 1.0\dot{\theta}_1^+\dot{\theta}_3^+ - 2.0\left(\dot{\theta}_1^-\right)^2 \cos(\theta_2^-) - 1.0\left(\dot{\theta}_1^-\right)^2 \cos(\theta_3^-) - 1.0\left(\dot{\theta}_1^-\right)^2 \cos(\theta_2^- + \theta_3^-) - 3.0\left(\dot{\theta}_1^-\right)^2 - 2.0\dot{\theta}_1^-\dot{\theta}_2^- \cos(\theta_2^-) - 2.0\dot{\theta}_1^-\dot{\theta}_2^- \cos(\theta_3^-) - 1.0\dot{\theta}_1^-\dot{\theta}_2^- \cos(\theta_2^- + \theta_3^-) - 3.0\dot{\theta}_1^-\dot{\theta}_2^- - 1.0\dot{\theta}_1^-\dot{\theta}_3^- \cos(\theta_3^-) - 1.0\dot{\theta}_1^-\dot{\theta}_3^- \cos(\theta_2^- + \theta_3^-) - 1.0\dot{\theta}_1^-\dot{\theta}_3^- + 1.0\left(\dot{\theta}_2^+\right)^2 \cos(\theta_3^-) + 1.5\left(\dot{\theta}_2^+\right)^2 + 1.0\dot{\theta}_2^+\dot{\theta}_3^+ \cos(\theta_3^-) + 1.0\dot{\theta}_2^+\dot{\theta}_3^+ - 1.0\left(\dot{\theta}_2^-\right)^2 \cos(\theta_3^-) - 1.5\left(\dot{\theta}_2^-\right)^2 - 1.0\dot{\theta}_2^-\dot{\theta}_3^- \cos(\theta_3^-) - 1.0\dot{\theta}_2^-\dot{\theta}_3^- + 0.5\left(\dot{\theta}_3^+\right)^2 - 0.5\left(\dot{\theta}_3^-\right)^2 = 0$

1.7 Problem 7

[12]:
```
# p7

def impact_update_trip_pend(states):
    """
    states include theta1-3 and theta_dot 1-3
    """
    subs_dict = {
        theta1_minus : states[0] ,
        theta2_minus : states[1] ,
        theta3_minus : states[2] ,
        theta1_dot_minus : states[3] ,
        theta2_dot_minus : states[4] ,
        theta3_dot_minus : states[5]
        }
    num_impact_eq = impact_equation.subs(subs_dict)

    # solve_and_print(num_impact_eq , [lamb, theta1_dot_plus, theta2_dot_plus,␣
↪theta3_dot_plus])
    solved_dicts = sym.solve(num_impact_eq,
                             [lamb, theta1_dot_plus, theta2_dot_plus,␣
↪theta3_dot_plus],
                             dict=True)
```

```
    lambda0 = solved_dicts[0][lamb]
    lambda1 = solved_dicts[1][lamb]

    better_solution = None
    # Instead of =0 checking on lambda, pick the bigger one
    if abs(lambda0) > abs(lambda1):
        better_solution = solved_dicts[0]
    else :
        better_solution = solved_dicts[1]
    theta1_plus_num = better_solution[theta1_dot_plus].evalf()
    theta2_plus_num  = better_solution[theta2_dot_plus].evalf()
    theta3_plus_num  = better_solution[theta3_dot_plus].evalf()
    # Integrate is really picky about same type in, same type out. So this is␣
 ↪not possible
    # return [*states[0:3], theta1_plus_num, theta2_plus_num, theta3_plus_num]
    states[3:6] = [theta1_plus_num, theta2_plus_num, theta3_plus_num]
    return states

state_test = [0,0,0 , -1,-1,-1]
state_plus = impact_update_trip_pend(state_test)

md_print(f"### Problem 7 solution \n\n **Output theta1 theta2 ... theta2_dot␣
 ↪theta3_dot on plus side is:** \n\n {state_plus}")
```

1.7.1 Problem 7 solution

Output theta1 theta2 ... theta2_dot theta3_dot on plus side is:

[0, 0, 0, -1.00000000000000, -1.00000000000000, 11.0000000000000]

1.8 Problem 8

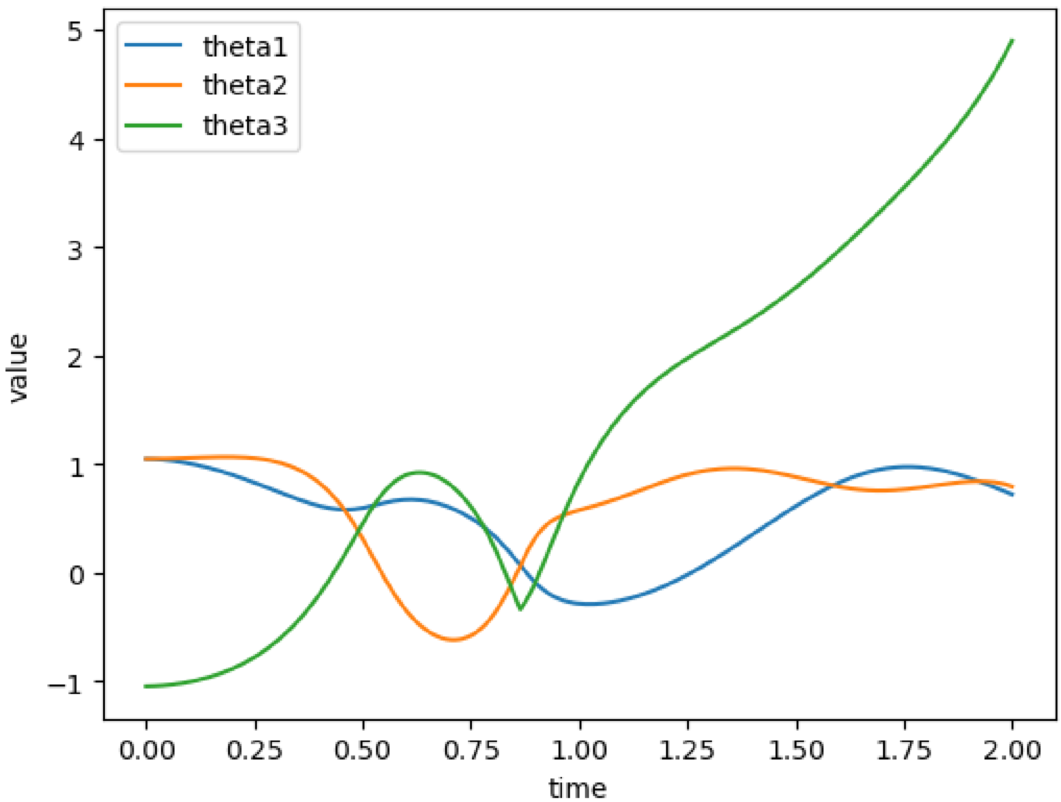
[13]:
```
#p8



# phi_lambda = sym.lambdify([*q] , phi)
phi_lambda = sym.lambdify([theta1 , theta2 , theta3] , phi)



def p8_get_constrain_phi_value(s)->float:
    return phi_lambda(*s[0:3])
def p8_impact_condition(s:list[float,float] , initial_sign)->bool:
    """Check if impact has happened
    """
    # if phi has changed sign.
    # print(f"phi {p8_get_constrain_phi_value(s)}")
    return p8_get_constrain_phi_value(s) * initial_sign <0
```

```python
def p8_simulate_impact(f, x0, tspan, dt, integrate):
    """
    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))

    phi_0 = p8_get_constrain_phi_value(x0)
    for i in range(N):
        if p8_impact_condition( x , phi_0):
            print("Impact happened")
            x = impact_update_trip_pend(x)
        xtraj[:,i]=integrate(f,x,dt)
        x = np.copy(xtraj[:,i])
    return tvec , xtraj

from  numpy import pi

s0 = [pi / 3, pi / 3, -pi / 3, 0, 0, 0]
t_span = [0,2]
dt = 0.01
p8_tvec,p8_traj = p8_simulate_impact(p5_system_equation , s0,t_span , dt␣
 ↪,integrate)

plt.figure(1)
plt.plot(p8_tvec , p8_traj[0] , label = "theta1")
```

```
plt.plot(p8_tvec , p8_traj[1] , label = "theta2")
plt.plot(p8_tvec , p8_traj[2] , label = "theta3")
plt.xlabel("time")
plt.ylabel("value")
plt.legend()
plt.plot()

animate_triple_pend(p8_traj,T=2)
```

```
Impact happened
```

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

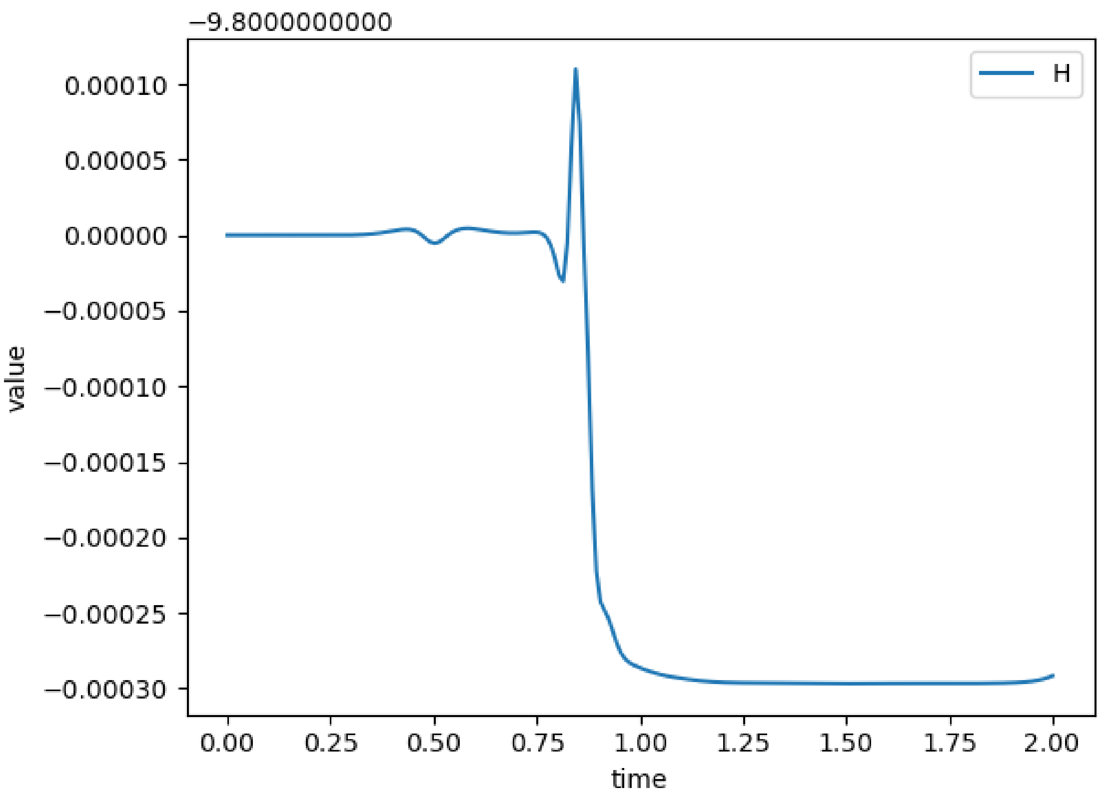


1.9 Problem 9

[14]:
```
H_list = []

for i in range(len(p8_tvec)):

    H_val = H.subs({
        theta1: p8_traj[0,i],
```

```
        theta2: p8_traj[1,i],
        theta3: p8_traj[2,i],
        theta1.diff(t): p8_traj[3,i],
        theta2.diff(t): p8_traj[4,i],
        theta3.diff(t): p8_traj[5,i],
    }).evalf()
    H_list.append(H_val)

plt.figure(1)
plt.plot(p8_tvec , H_list , label = "H")
plt.xlabel("time")
plt.ylabel("value")
plt.legend()
plt.plot()
```

[14]: []



1.10 Collaboration list

- Srikanth Schelbert
- Graham Clifford
- Ananya Agarwal

The notebook is generated locally. Thus no google collab is available.