HW2

October 10, 2023

1 ME314 Homework 2

Leo 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
# Helper function provided

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

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

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

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

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

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

```python
def animate_double_pend(theta_array,L1=1,L2=1,T=10):
    """
    Function to generate web-based animation of double-pendulum system

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

    Returns: None
    """

    ################################
    # Imports required for animation.
    from plotly.offline import init_notebook_mode, iplot
    from IPython.display import display, HTML
    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])
    N = len(theta_array[0]) # Need this for specifying length of simulation

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

    ###########################
    # Defining data dictionary.
    # Trajectories are here.
    data=[dict(x=xx1, y=yy1,
               mode='lines', name='Arm',
               line=dict(width=2, color='blue')
              ),
          dict(x=xx1, y=yy1,
               mode='lines', name='Mass 1',
               line=dict(width=2, color='purple')
              ),
          dict(x=xx2, y=yy2,
               mode='lines', name='Mass 2',
               line=dict(width=2, color='green')
              ),
          dict(x=xx1, y=yy1,
               mode='markers', name='Pendulum 1 Traj',
               marker=dict(color="purple", size=2)
              ),
          dict(x=xx2, y=yy2,
               mode='markers', name='Pendulum 2 Traj',
               marker=dict(color="green", size=2)
              ),
        ]

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

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

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

def animate_bead(xy_array,T=10):
    """
    Function to generate web-based animation of constrained bead system

    Parameters:
    ================================================
    xy_array:
        trajectory of x and y, should be a NumPy array with
        shape of (2,N)
    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 trajectories.
    N = len(xy_array[0]) # Need this for specifying length of simulation

    ####################################
    # Using these to specify axis limits.
    xm=np.min(xy_array[0])-0.5
    xM=np.max(xy_array[0])+0.5
    ym=np.min(xy_array[1])-0.5
    yM=np.max(xy_array[1])+0.5

    ###########################
    # Defining data dictionary.
    # Trajectories are here.
    data=[dict(x=xy_array[0], y=xy_array[1],
               mode='markers', name='bead',
               marker=dict(color="blue", size=10)
              ),
          dict(x=xy_array[0], y=xy_array[1],
               mode='lines', name='trajectory',
               line=dict(width=2, color='red')
              ),
        ]

    ################################
    # 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='Constrained Bead 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 bead at each time
    # step of simulation.
    frames=[dict(data=[go.Scatter(
                        x=[xy_array[0][k]],
                        y=[xy_array[1][k]],
                        mode="markers",
                        marker=dict(color="blue", size=10))
                  ]) 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

Find a sympy of Lagrangian

```
[11]: # p1

t= sym.symbols('t')

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



theta1_dot = theta1.diff(t)
theta2_dot = theta2.diff(t)
theta1_ddot = theta1_dot.diff(t)
theta2_ddot = theta2_dot.diff(t)

R1 , R2, m1 , m2  = sym.symbols('R_1,R_2,m_1,m_2')
g = 9.8 # Define positive G

# Origin is top of the pendulum, the corner of theta1.
# Define x is down, which is also the dir of g.
# Z is defined of the screen.
# wight right hand rule, y is right.

# The XY for m1 and m2 are position wrt origin. not diff between them.
# For m1
m1_x = R1*sym.cos(theta1)
m1_y = R1*sym.sin(theta1)

# For m2
m2_x = R1*sym.cos(theta1) + R2*sym.cos(theta1+theta2)
m2_y = R1*sym.sin(theta1) + R2*sym.sin(theta1+theta2)
# m1_xdot
# m1_ydot
# m1_xdot
# m1_ydot



# Now do the energy stuff
m1_kinematic = 0.5* m1 * (m1_x.diff(t) **2 + m1_y.diff(t) **2)
m2_kinematic = 0.5* m2 * (m2_x.diff(t) **2 + m2_y.diff(t) **2)
m1_potential = -m1*g*m1_x
m2_potential = -m2*g*m2_x
total_kinematic = m1_kinematic + m2_kinematic
total_potential = m1_potential + m2_potential

lagrangian = sym.simplify(total_kinematic - total_potential)
```

```
md_print(f"""
### Problem1 solution
Lagrangian of the system is :

{lax_eq( sym.expand(lagrangian))}
""")
```

1.1.1 Problem1 solution

Lagrangian of the system is :

$$0.5R_1^2 m_1 \left(\frac{d}{dt}\theta_1(t)\right)^2 + 0.5R_1^2 m_2 \left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0R_1 R_2 m_2 \cos(\theta_2(t)) \left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0R_1 R_2 m_2 \cos(\theta_2(t)) \frac{d}{dt}\theta_1(t) \frac{d}{dt}\theta_2(t) + 9.8R_1 m_1 \cos(\theta_1(t)) + 9.8R_1 m_2 \cos(\theta_1(t)) + 0.5R_2^2 m_2 \left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0R_2^2 m_2 \frac{d}{dt}\theta_1(t) \frac{d}{dt}\theta_2(t) + 0.5R_2^2 m_2 \left(\frac{d}{dt}\theta_2(t)\right)^2 + 9.8R_2 m_2 \cos(\theta_1(t) + \theta_2(t))$$

1.2 Problem 2

Euler-Lagrange, solve for theta ddot s

[12]:
```
q = sym.Matrix([theta1,theta2])
q_dot = sym.Matrix([theta1_dot,theta2_dot])
q_ddot = sym.Matrix([theta1_ddot,theta2_ddot])

dL_dq = sym.Matrix([lagrangian]).jacobian(q).T
dL_dqdot = sym.Matrix([lagrangian]).jacobian(q_dot).T

euler_lagrange =sym.simplify(dL_dqdot.diff(t) - dL_dq)
print("eular_lagrange is :")
display(euler_lagrange)

el_equation = sym.Eq(euler_lagrange, sym.Matrix([0,0]))
solved_el = sym.solve(euler_lagrange, [theta1_ddot, theta2_ddot], dict=True)[0]

md_print("### Problem2 solution sets")

solved_eq_dict = {}
md_print("solution:")
for var in q_ddot:
    eq_local = sym.simplify(sym.Eq(var , solved_el[var]))
    display(sym.expand(eq_local))
    solved_eq_dict[var] = solved_el[var]
```

```
eular_lagrange is :
```

$$\left[\begin{matrix} 1.0R_1^2 m_1 \frac{d^2}{dt^2}\theta_1(t) + 9.8R_1 m_1 \sin(\theta_1(t)) + 9.8m_2 (R_1 \sin(\theta_1(t)) + R_2 \sin(\theta_1(t) + \theta_2(t))) + m_2 \left(R_1^2 \frac{d^2}{dt^2}\theta_1(t) - 2R_1 R \\ R_2 m_2 \left(1.0R_1 \sin(\theta_2(t)) \left(\frac{d}{dt}\theta_1(t)\right)^2 + 1.0R_1 \cos(\end{matrix}\right.$$

1.2.1 Problem2 solution sets

solution:

$$\frac{d^2}{dt^2}\theta_1(t) = \frac{0.5R_1m_2\sin(2\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1m_1+R_1m_2\sin^2(\theta_2(t))} + \frac{1.0R_2m_2\sin(\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1m_1+R_1m_2\sin^2(\theta_2(t))} + \frac{2.0R_2m_2\sin(\theta_2(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t)}{R_1m_1+R_1m_2\sin^2(\theta_2(t))} + \frac{1.0R_2m_2\sin(\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2}{R_1m_1+R_1m_2\sin^2(\theta_2(t))} - \frac{9.8m_1\sin(\theta_1(t))}{R_1m_1+R_1m_2\sin^2(\theta_2(t))} + \frac{4.9m_2\sin(\theta_1(t)+2\theta_2(t))}{R_1m_1+R_1m_2\sin^2(\theta_2(t))} - \frac{4.9m_2\sin(\theta_1(t))}{R_1m_1+R_1m_2\sin^2(\theta_2(t))}$$

$$\frac{d^2}{dt^2}\theta_2(t) = -\frac{1.0R_1^2m_1\sin(\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{1.0R_1^2m_2\sin(\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{1.0R_1R_2m_2\sin(2\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{1.0R_1R_2m_2\sin(2\theta_2(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t)}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{0.5R_1R_2m_2\sin(2\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} + \frac{4.9R_1m_1\sin(\theta_1(t)-\theta_2(t))}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{4.9R_1m_1\sin(\theta_1(t)+\theta_2(t))}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} + \frac{4.9R_1m_2\sin(\theta_1(t)-\theta_2(t))}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{4.9R_1m_2\sin(\theta_1(t)+\theta_2(t))}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{1.0R_2^2m_2\sin(\theta_2(t))\left(\frac{d}{dt}\theta_1(t)\right)^2}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{2.0R_2^2m_2\sin(\theta_2(t))\frac{d}{dt}\theta_1(t)\frac{d}{dt}\theta_2(t)}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{1.0R_2^2m_2\sin(\theta_2(t))\left(\frac{d}{dt}\theta_2(t)\right)^2}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} + \frac{9.8R_2m_1\sin(\theta_1(t))}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} - \frac{4.9R_2m_2\sin(\theta_1(t)+2\theta_2(t))}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))} + \frac{4.9R_2m_2\sin(\theta_1(t))}{R_1R_2m_1+R_1R_2m_2\sin^2(\theta_2(t))}$$

1.3 Problem 3

Solve for it, and simulate the system

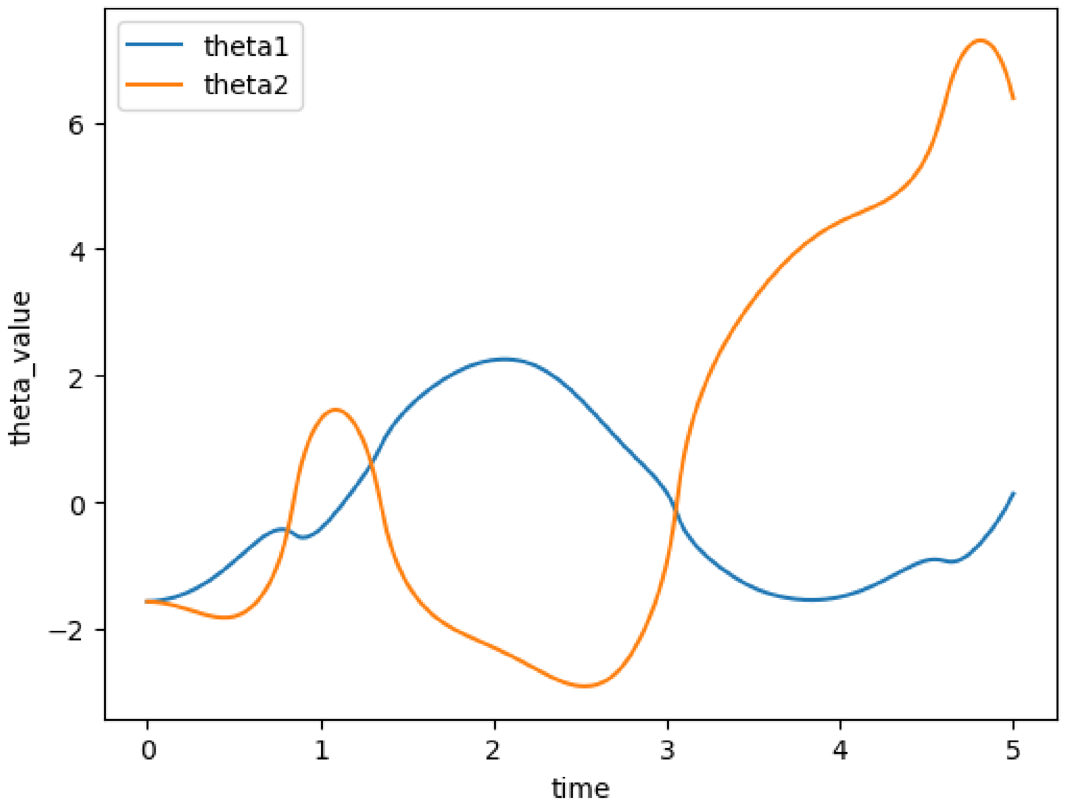
[5]:
```python
# P3

subs_list = [(R1, 2), (R2, 1), (m1, 1), (m2, 2)]

initial_state = [-np.pi / 2, -np.pi / 2, 0, 0]
t_range = [0,5]
time_step = 0.01
lambda_dict = {}

print("Value of q_ddot with initial condition plugged in:")
for var in q_ddot:
    substituted = solved_eq_dict[var].subs(subs_list)
    substituted = sym.simplify(substituted)
    lambda_func = sym.lambdify([theta1, theta2, theta1_dot, theta2_dot],␣
 ↪substituted )
    lambda_dict[var] = lambda_func
    value = lambda_func(-np.pi / 2, -np.pi / 2, 0, 0)
    md_print(f"{lax_eq(var)}: {value}")

# now simulate

def system_equation(state):
    '''
    Takes theta1, theta2 , theta1dot, theta2dot as input
    output theta1dot, theta2dot , theta1ddot, theta2ddot
    '''

    t1_pos = state[0]
    t2_pos = state[1]
    t1_v = state[2]
    t2_v = state[3]
    t1_a = lambda_dict[theta1_ddot](t1_pos,t2_pos , t1_v,t2_v)
    t2_a = lambda_dict[theta2_ddot](t1_pos,t2_pos , t1_v,t2_v)

    return np.array([t1_v,t2_v,t1_a,t2_a])


time_vec , q_traj = simulate(system_equation , initial_state , t_range ,␣
↪time_step , integrate)

plt.figure(1)
plt.plot(time_vec , q_traj[0] , label = "theta1")
plt.plot(time_vec , q_traj[1] , label = "theta2")
plt.xlabel("time")
plt.ylabel("theta_value")
plt.legend()
plt.plot()
```

```
Value of q_ddot with initial condition plugged in:
```

$\frac{d^2}{dt^2}\theta_1(t)$: 4.9

$\frac{d^2}{dt^2}\theta_2(t)$: -4.8999999999999995

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



1.4 Problem 4

animation

[6]:
```
##################################################
# Example of animation

# provide a trajectory of double-pendulum
# (note that this array below is not an actual simulation,
# but lets you see this animation code work)

# second, animate!
animate_double_pend(q_traj,L1=2,L2=1,T=5)
```

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

2 Problem 5

Lagrangian on constrain

[7]:
```python
x = sym.Function(r'x')(t)
x_dot = x.diff(t)
x_ddot = x_dot.diff(t)

m = sym.symbols('m')

y = sym.Function(r'y')(t)
y_dot = y.diff(t)
y_ddot = y_dot.diff(t)

lambda_scaler = sym.symbols('\lambda')
p5_potential = m*9.8*y
p5_kinetic = 0.5*m*(x_dot**2 + y_dot**2)
p5_lagrangian = p5_kinetic - p5_potential

# constrain
constrain_phi = sym.cos(x) - y
constrain_eq = sym.Eq( constrain_phi , 0)

md_print(f"""
### Problem5 solution
**Lagrangian**
{lax_eq(p5_lagrangian)}

**constrain**
{lax_eq(constrain_eq)}
""")
```

2.0.1 Problem5 solution

Lagrangian $0.5m\left(\left(\frac{d}{dt}x(t)\right)^2 + \left(\frac{d}{dt}y(t)\right)^2\right) - 9.8my(t)$

constrain $-y(t) + \cos\left(x(t)\right) = 0$

2.1 Problem 6

Solve for x ddot y ddot and lambda

[8]:
```python
p6_q = sym.Matrix([x,y])
p6_q_dot = sym.Matrix([x_dot,y_dot])
p6_q_ddot = sym.Matrix([x_ddot,y_ddot])

p6_dL_dq = sym.Matrix([p5_lagrangian]).jacobian(p6_q).T
p6_dL_dqdot = sym.Matrix([p5_lagrangian]).jacobian(p6_q_dot).T

p6_eu_la = p6_dL_dqdot.diff(t) - p6_dL_dq

phi_dx = constrain_phi.diff(x)
```

```
phi_dy = constrain_phi.diff(y)


lhs = sym.Matrix([p6_eu_la[0] - lambda_scaler * phi_dx, p6_eu_la[1] -␣
 ↪lambda_scaler * phi_dy, constrain_phi.diff(t).diff(t)])
rhs =sym.Matrix([0,0,0])

constrained_system_eq = sym.Eq(lhs, rhs)

display(constrained_system_eq)
p6_vars = [x_ddot,y_ddot,lambda_scaler]
p6_solution = sym.solve(constrained_system_eq , p6_vars , dict = True)


md_print("### Problem 6 Solution sets")
for sol in p6_solution:
    md_print("Solution:")
    for q in p6_vars:
        display(sym.Eq(q,sol[q]))
```

$$\begin{bmatrix} \lambda\sin(x(t)) + 1.0m\frac{d^2}{dt^2}x(t) \\ \lambda + 1.0m\frac{d^2}{dt^2}y(t) + 9.8m \\ -\sin(x(t))\frac{d^2}{dt^2}x(t) - \cos(x(t))\left(\frac{d}{dt}x(t)\right)^2 - \frac{d^2}{dt^2}y(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

2.1.1 Problem 6 Solution sets

Solution:

$$\frac{d^2}{dt^2}x(t) = -\frac{5.0\sin(x(t))\cos(x(t))\left(\frac{d}{dt}x(t)\right)^2}{5.0\sin^2(x(t)) + 5.0} + \frac{49.0\sin(x(t))}{5.0\sin^2(x(t)) + 5.0}$$

$$\frac{d^2}{dt^2}y(t) = -\frac{49.0\sin^2(x(t))}{5.0\sin^2(x(t)) + 5.0} - \frac{5.0\cos(x(t))\left(\frac{d}{dt}x(t)\right)^2}{5.0\sin^2(x(t)) + 5.0}$$

$$\lambda = \frac{5.0m\cos(x(t))\left(\frac{d}{dt}x(t)\right)^2}{5.0\sin^2(x(t)) + 5.0} - \frac{49.0m}{5.0\sin^2(x(t)) + 5.0}$$

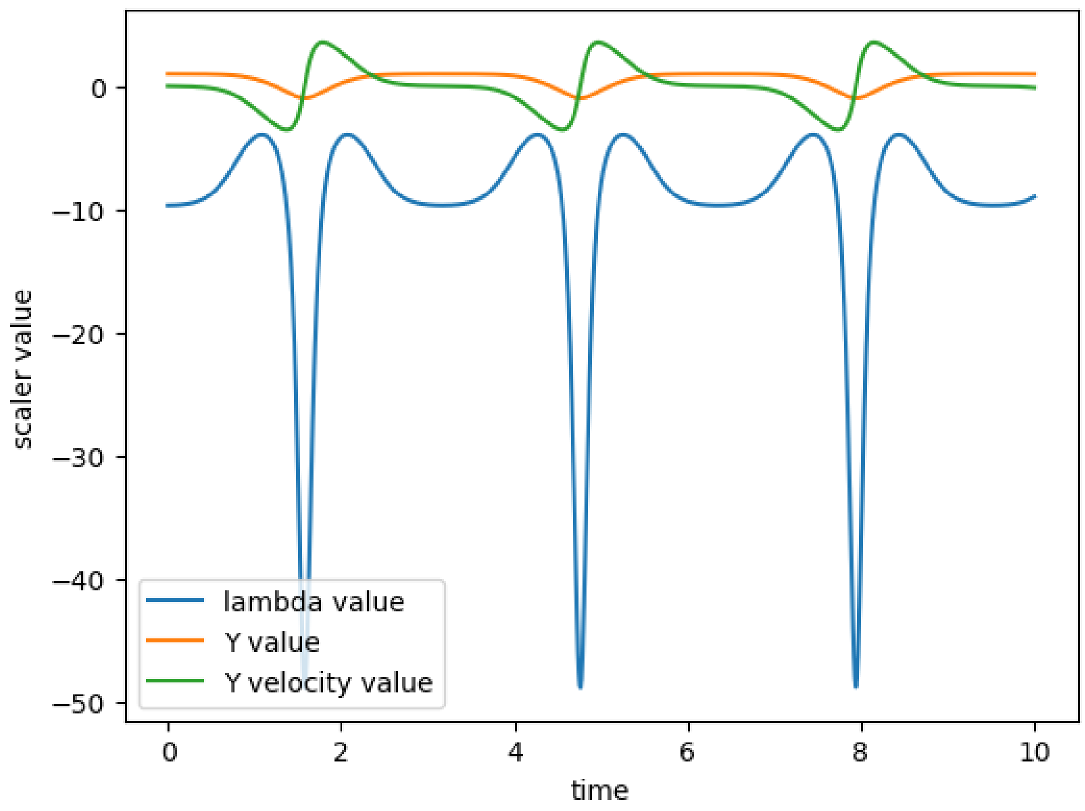
2.2 problem 7

[9]:
```
p7_subs_list = [(m,1) ]

initial_state = [0.1 , np.cos(0.1) , 0 , 0]
time_range = [0,10]
time_step = 0.01



p7_lambdafied = {}
for var in  p6_vars:
    substituted = p6_solution[0][var].subs(p7_subs_list)

    p7_lambdafied[var] = sym.lambdify( [x,y,x_dot,y_dot],  substituted)


def p7_system_eq(state):
    x_pos = state[0]
    y_pos = state[1]
    x_vel = state[2]
    y_vel = state[3]
    x_acc = p7_lambdafied[x_ddot](x_pos , y_pos , x_vel,y_vel)
    y_acc = p7_lambdafied[y_ddot](x_pos , y_pos , x_vel,y_vel)
    return np.array([x_vel,y_vel,x_acc,y_acc])


time_vec , q_traj = simulate(p7_system_eq , initial_state , time_range ,␣
 ↪time_step , integrate)

animate_bead(q_traj, T = 10)
```

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

2.3 Problem 8

[10]:
```
lambda_over_time = []

# for q1,q2,q3,q4 in zip(q_traj[0] ,q_traj[1] , q_traj[2] , q_traj[3] ):
#     lambda_over_time.append(p7_lambdafied[lambda_scaler](state[0], state[1],␣
 ↪state[2], state[3]))

lambda_values = p7_lambdafied[lambda_scaler](q_traj[0], q_traj[1], q_traj[2],␣
 ↪q_traj[3])



plt.plot(time_vec, lambda_values , label = "lambda value")
plt.plot(time_vec, q_traj[1] , label = 'Y value')
plt.plot(time_vec, q_traj[3] , label = 'Y velocity value')
plt.legend()
plt.xlabel("time")
plt.ylabel("scaler value")
plt.show()
```



2.3.1 Problem 8 conclusion

The value of lambda changes periodically over time as the bead move from one end to the other on the constrain.

The value of lambda is around 10 when Y value reaches 0 because the constrain is just resisting gravity, At two ends of he track, the bead is not really moving thus only need to resist gravity.

The value of lambda spike at minimal of Y since the bead is moving down at high velocity, reverting the direction of motion asked for a large acceleration and thus a large spike of force.

2.4 Collaboration list

- Srikanth Schelbert
- Graham Clifford
- Aditya Nair
- Jialu Yu
- Jihai Zhao
- Shail Dalal

This is from a local notebook, for better pdf generation.

The file is uploaded to google drive for access. But not runned in the google Colab.

https://drive.google.com/file/d/1QzgmIZfaPAmjDneIzu6ntIzgRdT9_0H0/view?usp=sharing