Question 2

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D\dot{q} + g(q) = u$$

$e = q - q_r \quad \dot{e} = \dot{q} \quad \ddot{e} = \ddot{q}$ $\quad q_r$ : constant reference trajectory

$$V_0 = \frac{1}{2} e^T e + \frac{1}{2} \dot{e}^T \dot{e}$$

$$\dot{V}_0 = \dot{e}^T e + \ddot{e}^T \dot{e} \quad \because \ddot{e}^T \dot{e} = \dot{e}^T \ddot{e} \text{ (scalar)}$$

$$= \dot{e}^T e + \dot{e}^T \ddot{e}$$

$$= \dot{e}^T \left[ e + \ddot{e} \right]$$

$e_1 = e \quad e_2 = \dot{e}$

$$\dot{e}_1 = e_2$$

$$\dot{e}_2 = + \left[ M(q)^{-1} \left[ -C\dot{q} - D\dot{q} - g + u \right] \right]$$

$$\dot{V}_0^T = \dot{q}^T \left[ \underbrace{q - q_r + M^{-1} \left[ u - C\dot{q} - D\dot{q} - g \right]}_{-\dot{q}} \right]$$

if

$$K_0 = u_{des} = M \left[ -\dot{q}_r = (q - q_r) \right] + C\dot{q} + D\dot{q} + g$$

$$\dot{V}_0 = -\dot{q}^T \dot{q} < 0$$

now for the system with the integrator:

$$\ddot{q} = H(q)^{-1} \left[ -C\dot{q} - D\dot{q} - g + \xi_3 \right]$$

$$\dot{\xi}_3 = \tau$$

Let $V_1 = V_0 + \frac{1}{2} \left[ \xi_3 - K_0 \right]^T \left[ \xi_3 - K_0 \right]$

$$\dot{V}_1 = \dot{V}_0 + \left[ \xi_3 - K_0 \right]^T \left[ \dot{\xi}_3 - \dot{K} \right]$$

$$\dot{V}_1 = \dot{q}^T \left[ (q - q_r) + H^{-1} \left[ \xi_3 - C\dot{q} - D\dot{q} - g \right] \right] + \dot{q}^T \dot{q} - \dot{q}^T \dot{q}$$

$$+ \left[ \xi_3 - H \left[ -\dot{q} - (q - q_r) \right] + C\dot{q} + D\dot{q} + g \right] \left[ \tau - \dot{K} \right]$$

$$\dot{V}_1 = -\dot{q}^T\dot{q} + \dot{q}^T\left[(q-q_r) + H^{-1}(\varepsilon_s - C\dot{q} - D\dot{q} - g) + \dot{q}\right]$$

$$+ \left[\varepsilon_s - H(-\dot{q} - (q-q_r)) - (C\dot{q} + D\dot{q} + g)\right]^T [\tau - k]$$

~~$$\dot{V}_1 = -\dot{q}^T\dot{q} + \dot{q}^T\left[(q-q_r) + H^{-1}[H\ddot{q}] + \dot{q}\right]$$~~

~~$$+ \left[H[\dot{q} + (q-q_r)] + H\ddot{q}\right]$$~~

$$\dot{V} = -\dot{q}^T\dot{q} + \dot{q}^T\left[(q-q_r) + \dot{q} + H^{-1}(\varepsilon_s - C\dot{q} - D\dot{q} - g)\right]$$

$$+ \left[H[(q-q_r) + \dot{q} + H^{-1}(\varepsilon_s - C\dot{q} - D\dot{q} - g)]\right]^T [\tau - k]$$

$$\dot{V} = -\dot{q}^T\dot{q} + \left[(q-q_r) + \dot{q} + H^{-1}(\varepsilon_s - C\dot{q} - D\dot{q} - g)\right]^T \left[\dot{q} + H^T\tau - H^T k\right]$$

$$\dot{q} + H^T\tau - H^T k = -\left[(q-q_r) + \dot{q} + H^{-1}(\varepsilon_s - C\dot{q} - D\dot{q} - g)\right]$$

$$\tau = H^{-1}\left[H^T k - \dot{q} - \left[(q-q_r) + \dot{q} + H^{-1}(\varepsilon_s - C\dot{q} - D\dot{q} - g)\right]\right]$$

$$\boxed{\tau = k + (H^T)^{-1}\left[-(q-q_r) - 2\dot{q} - H^{-1}(\varepsilon_s - C\dot{q} - D\dot{q} - g)\right]}$$

Input from backstepping

$\therefore$

$\dot{V} < 0$ $\therefore$

$$\dot{V} = -\dot{q}^T\dot{q} - \left[(q-q_r) + \dot{q} + H^{-1}(\varepsilon_s - C\dot{q} - D\dot{q} - g)\right]^T$$
$$\left[(q-q_r) + \dot{q} + H^{-1}(\varepsilon_s - C\dot{q} - D\dot{q} - g)\right]$$

Since $g$ can be neglected

$$K = M[-\dot{q} - (q - q_r)] + C\dot{q} + D\dot{q} + g \text{ neglected}$$

$$\dot{K} = \dot{M}[-\dot{q} - (q - q_r)] + C\ddot{q} + D\ddot{q}$$

$$+ M[-\ddot{q} - \dot{q}] + \dot{C}\dot{q}$$

Question 1: Robotic Manioulator using passivity based control

```python
import numpy as np
from math import sin, cos, tanh
from numpy.linalg import inv
import matplotlib.pyplot as plt
# import scipy.integrate as integrate
import matplotlib.animation as animation


class RoboticManipulator():
   def __init__(self,
               q1_ref_0 = 0   ,
               q2_ref_0 = 0   ,
               q1_0     = 0 ,
               q2_0     = 0 ,
               q1dot_0  = 0 ,
               q2dot_0  = 0 ,
               q1dot2_0 = 0 ,
               q2dot2_0 = 0 ):

      ### CONSTANTS
      self.p1 = 3.31     #kg.m^2
      self.p2 = 0.116
      self.p3 = 0.16

      self.L1 = 0.8      #m
      self.L2 = 0.5
      self.O1 = -0.666   #m
      self.O2 = 0.333

      self.dt = 0.05     #s

      ### TUNING CONSTANTS
      self.k = 1         # control gain
      self.Kp = np.array([[1,0],
                          [0,1]])
      self.D =  np.array([[1,0],
                          [0,1]])

      ### VARIABLES
      self.q1     = q1_0
      self.q2     = q2_0
      self.q1dot  = q1dot_0
      self.q2dot  = q2dot_0
      self.q1dot2 = q1dot2_0
      self.q2dot2 = q2dot2_0

      self.x1 = self.L1*cos(self.q1) + self.L2*cos(self.q1 + self.q2) + self.O1
      self.x2 = self.L1*sin(self.q1) + self.L2*sin(self.q1 + self.q2) + self.O2

      self.x21 = self.L1*cos(self.q1) + self.O1
      self.x22 = self.L1*sin(self.q1) + self.O2

      ### REFERNCE INPUT
      self.q1_ref = q1_ref_0
      self.q2_ref = q2_ref_0
```

```
    self.x1_ref = self.L1*cos(self.q1_ref) + self.L2*cos(self.q1_ref + self.q2_ref) + self.O1
    self.x2_ref = self.L1*sin(self.q1_ref) + self.L2*sin(self.q1_ref + self.q2_ref) + self.

    self.x21_ref = self.L1*cos(self.q1_ref) + self.O1
    self.x22_ref = self.L1*sin(self.q2_ref) + self.O2

    self.update_state_matrices()



def update_state_matrices(self):
    self.M = np.array([ [ self.p1 + 2*self.p3*cos(self.q2) ,  self.p2 + self.p3*cos(self.q2) ] ,
                        [ self.p2 + self.p3*cos(self.q2)   ,  self.p2                        ] ] )

    self.C = np.array([ [ -self.q2dot , self.q1dot + self.q2dot ] ,
                        [ self.q1dot  , 0                        ] ] ) *self.p3*sin(self.q2)

    self.g = np.array([ [ 0  ] ,
                        [ 0  ] ] )

    self.Omega = np.array([ [ self.x1 ] ,
                            [ self.x2 ] ] )

    self.omega2 = np.array([ [ self.x21 ] ,
                             [ self.x22 ] ] )

    self.q_ref = np.array([ [ self.q1_ref  ] ,
                            [ self.q2_ref  ] ] )

    self.x_ref = np.array([[self.x1_ref],
                           [self.x2_ref]])

    self.q = np.array([ [ self.q1  ] ,
                        [ self.q2  ] ] )

    self.qdot = np.array([ [ self.q1dot  ] ,
                           [ self.q2dot  ] ] )

    self.qdot2 = np.array([ [ self.q1dot2  ] ,
                            [ self.q2dot2  ] ] )

    self.E = np.copy(self.q - self.q_ref)

    self.Edot = np.copy(self.qdot)

    self.Edot2 = np.copy(self.qdot2)

    self.v_input = -self.k* np.array([ [ tanh(self.Edot[0][0])  ] ,
                                       [ tanh(self.Edot[1][0])  ] ] )

    self.U = self.g -np.dot(self.Kp,self.E) + self.v_input


def advance_and_update_states(self):
```

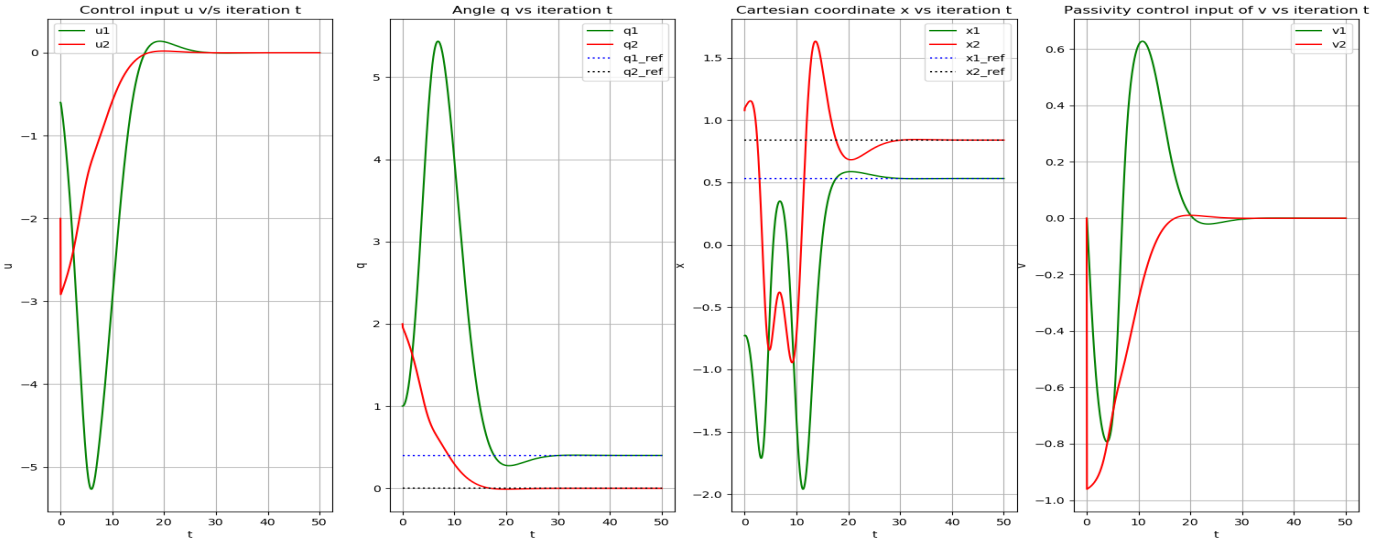
```
        self.qdot2 = np.dot(inv(self.M),-np.dot(self.Kp, self.E) -np.dot(self.D, self.Edot)
-np.dot(self.C, self.Edot) + self.v_input)
        self.qdot  = self.qdot + self.qdot2*self.dt
        self.q     = self.q + self.qdot*self.dt

        self.q1,    self.q2    = self.q[0][0]    , self.q[1][0]
        self.q1dot, self.q2dot = self.qdot[0][0] , self.q[1][0]
        self.q1dot2,self.q2    = self.qdot2[0][0], self.q[1][0]

        self.x1 = self.L1*cos(self.q1) + self.L2*cos(self.q1 + self.q2) + self.O1
        self.x2 = self.L1*sin(self.q1) + self.L2*sin(self.q1 + self.q2) + self.O2

        self.x21 = self.L1*cos(self.q1) + self.O1
        self.x22 = self.L1*sin(self.q1) + self.O2

        self.update_state_matrices()


    def simulate(self, t_f):
        self.t = np.linspace(0,t_f,int(t_f/self.dt))
        self.u_array      = []
        self.u_1_array    = []
        self.u_2_array    = []

        self.v_array      = []
        self.v_1_array    = []
        self.v_2_array    = []

        self.q_array      = []
        self.q_r_array    = []
        self.q_1_array    = []
        self.q_1_r_array  = []
        self.q_2_array    = []
        self.q_2_r_array  = []

        self.x_array      = []
        self.x_r_array    = []
        self.x_1_array    = []
        self.x_2_array    = []
        self.x_1_r_array  = []
        self.x_2_r_array  = []

        self.x02_1_array  = []
        self.x02_2_array  = []
        self.x02_1_r_array= []
        self.x02_2_r_array= []


        for _ in range(len(self.t)):
            self.u_array.append(np.linalg.norm(self.U))
            self.u_1_array.append(self.U[0])
            self.u_2_array.append(self.U[1])

            self.v_array.append(np.linalg.norm(self.v_input))
            self.v_1_array.append(self.v_input[0])
            self.v_2_array.append(self.v_input[1])
```

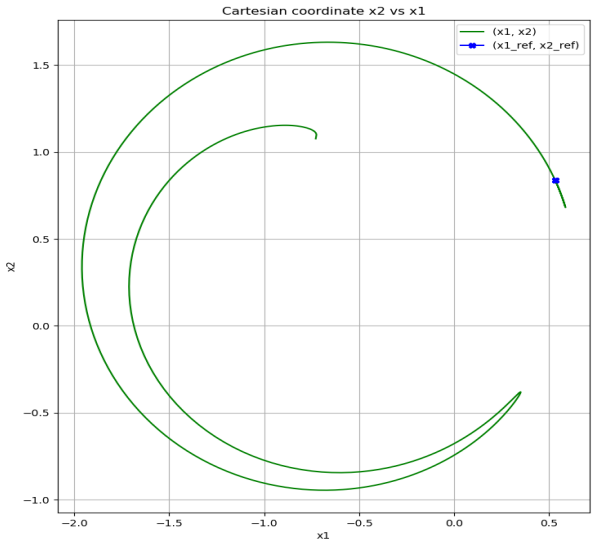
```
        self.q_array.append(np.linalg.norm(self.q))
        self.q_r_array.append(np.linalg.norm(self.q_ref))
        self.q_1_array.append(self.q[0])
        self.q_2_array.append(self.q[1])
        self.q_1_r_array.append(self.q1_ref)
        self.q_2_r_array.append(self.q2_ref)

        self.x_r_array.append(self.x_ref)
        self.x_1_r_array.append(self.x1_ref)
        self.x_2_r_array.append(self.x2_ref)
        self.x_array.append(self.Omega)
        self.x_1_array.append(self.x1)
        self.x_2_array.append(self.x2)

        self.x02_1_array.append(self.x21)
        self.x02_2_array.append(self.x22)
        self.x02_1_r_array.append(self.x21_ref)
        self.x02_2_r_array.append(self.x22_ref)

        self.advance_and_update_states()


def plot(self):
    fig, ax = plt.subplots(1, 4)

    # ax[0].plot(self.t, self.u_array, '-c')
    ax[0].plot(self.t, self.u_1_array, '-g')
    ax[0].plot(self.t, self.u_2_array, '-r')

    ax[0].legend(["u1", "u2"])
    ax[0].set_ylabel("u")
    ax[0].set_xlabel("t")
    ax[0].grid()
    ax[0].set_title("Control input u v/s iteration t")

    # ax[1].plot(self.t, self.q_array, '-b')
    ax[1].plot(self.t, self.q_1_array, '-g')
    ax[1].plot(self.t, self.q_2_array, '-r')
    ax[1].plot(self.t, self.q_1_r_array, ':b')
    ax[1].plot(self.t, self.q_2_r_array, ':k')

    ax[1].legend(["q1", "q2", "q1_ref", "q2_ref"])
    ax[1].set_ylabel("q")
    ax[1].set_xlabel("t")
    ax[1].grid()
    ax[1].set_title("Angle q vs iteration t")

    # ax[2].plot(self.t, self.x_array, '-b')
    ax[2].plot(self.t, self.x_1_array, '-g')
    ax[2].plot(self.t, self.x_2_array, '-r')
    ax[2].plot(self.t, self.x_1_r_array, ':b')
    ax[2].plot(self.t, self.x_2_r_array, ':k')

    ax[2].legend(["x1", "x2", "x1_ref", "x2_ref"])
    ax[2].set_ylabel("x")
```

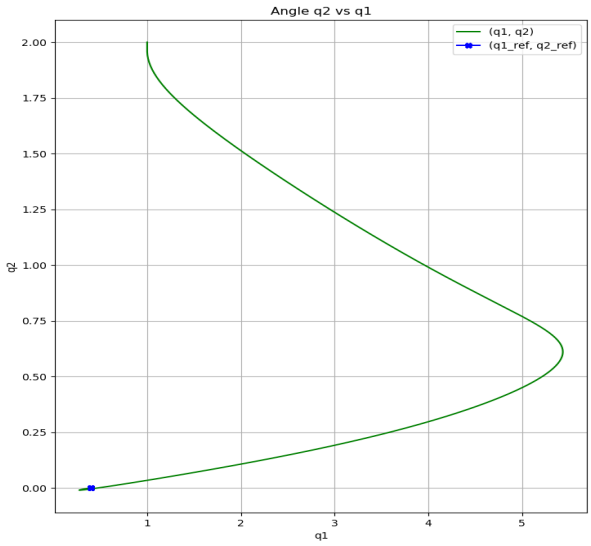
```
        ax[2].set_xlabel("t")
        ax[2].grid()
        ax[2].set_title("Cartesian coordinate x vs iteration t")

        # ax[3].plot(self.t, self.v_array, '-b')
        ax[3].plot(self.t, self.v_1_array, '-g')
        ax[3].plot(self.t, self.v_2_array, '-r')

        ax[3].legend(["v1", "v2"])
        ax[3].set_ylabel("v")
        ax[3].set_xlabel("t")
        ax[3].grid()
        ax[3].set_title("Passivity control input of v vs iteration t")

        # plt.show()

        fig, ax = plt.subplots(1, 2)

        ax[0].plot(self.x_1_array, self.x_2_array, '-g')
        ax[0].plot(self.x_1_r_array, self.x_2_r_array, 'X-b')

        ax[0].legend(["(x1, x2)", "(x1_ref, x2_ref)"])
        ax[0].set_ylabel("x2")
        ax[0].set_xlabel("x1")
        ax[0].grid()
        ax[0].set_title("Cartesian coordinate x2 vs x1")


        ax[1].plot(self.q_1_array, self.q_2_array, '-g')
        ax[1].plot(self.q_1_r_array, self.q_2_r_array, 'X-b')
        ax[1].legend(["(q1, q2)", "(q1_ref, q2_ref)"])
        ax[1].set_ylabel("q2")
        ax[1].set_xlabel("q1")
        ax[1].grid()
        ax[1].set_title("Angle q2 vs q1")

        # plt.show()

    def animate_this(self):
        fig = plt.figure()
        ax = fig.add_subplot(111, autoscale_on=False, xlim=(-2, 2), ylim=(-2, 2))
        ax.set_aspect('equal')
        ax.grid()

        line, = ax.plot([], [], 'o-', lw=2)
        time_template = 'time = %.1fs'
        time_text = ax.text(0.05, 0.9, '', transform=ax.transAxes)
        def init():
            line.set_data([], [])
            time_text.set_text('')
            return line, time_text
        def animate(i):
            thisx = [self.O1, self.x02_1_array[i], self.x_1_array[i]]
            thisy = [self.O2, self.x02_2_array[i], self.x_2_array[i]]

            line.set_data(thisx, thisy)
```

```
            time_text.set_text(time_template % (i*self.dt))
            return line, time_text

        ani = animation.FuncAnimation(fig, animate, np.arange(1, len(self.x_1_array)),
                                      interval=25, blit=True, init_func=init)

        plt.show()

if __name__ == '__main__':

    q_ref = np.array([[3],
                      [0 ]])
    q_0   = np.array([[3],
                      [1]])

    simulation_time = 50

    tlm1 = RoboticManipulator(q_ref[0][0],
                              q_ref[1][0],
                              q_0[0][0]  ,
                              q_0[1][0] )
    tlm1.simulate(simulation_time)

    tlm1.plot()
    tlm1.animate_this()
```

PLOTS:





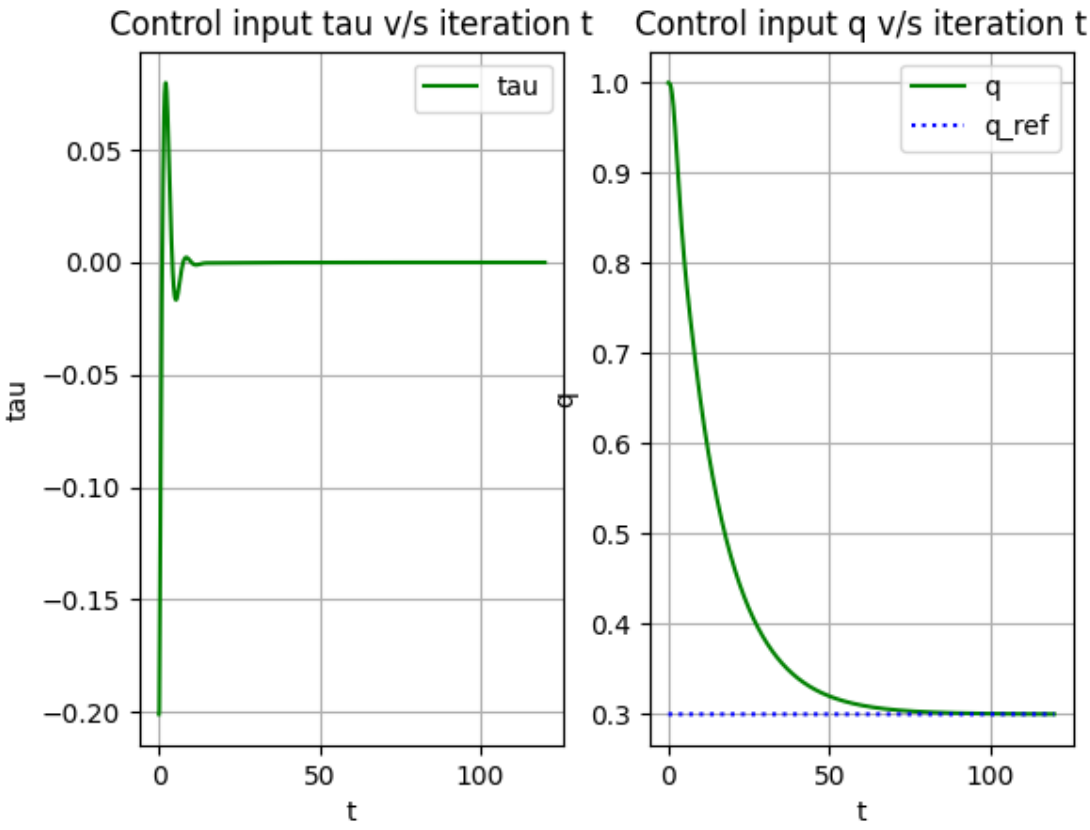

Question 2: Robotic Manipulator using Backstepping

```python
import numpy as np
import matplotlib.pyplot as plt

class manipulator:
    def __init__(self, p_1, p_3, q_0, q_r):

        # system states
        self.q = q_0
        self.q_r = q_r
        self.e = self.q - self.q_r
        self.q_dot = 0.
        self.q_dot_dot = 0.
        self.xi = 0.
        self.xi_dot = 0

        # constant values
        self.p_1 = p_1
        self.p_3 = p_3
        self.dt = 0.1

        self.update_mc()

    def update_mc(self):
        self.m = self.p_1 + (2 * self.p_3 * np.cos(self.q))
        self.c = -self.p_3 * np.sin(self.q) * self.q_dot
        self.m_dot = -2 * self.p_3 * np.sin(self.q) * self.q_dot
```

```
        self.c_dot = -(self.p_3 * np.cos(self.q) * (self.q_dot**2)) - (self.p_3 *
np.sin(self.q) * self.q_dot_dot)

    def advance_state(self, tau):
        dt = self.dt

        # dynamics
        self.q_dot_dot = (self.xi - (self.c * self.q_dot)) / self.m
        self.xi_dot = tau

        # update states
        self.q_dot += self.q_dot_dot * dt
        self.q += self.q_dot * dt
        self.xi += self.xi_dot * dt
        self.update_mc()

    def compute_control_law(self): # taking the control lyapunov function to be
V(q,q_dot) = e^2/2 + q_dot^2/2
        # k_0 = Cq_dot + mq_r - mq - mq_dot
        k_0_dot = (self.c_dot * self.q_dot + self.c * self.q_dot_dot) + self.m_dot *
self.q_r - (self.m_dot*self.q + self.m*self.q_dot) - (self.m_dot*self.q_dot +
self.m*self.q_dot_dot)
        self.tau = k_0_dot - (2*self.q_dot/self.m) - (self.xi/(self.m**2)) +
(self.c*self.q_dot/(self.m**2)) + ((self.q_r - self.q)/self.m)



    def simulate(self, t_f) :
        self.t = np.linspace(0,t_f,int(t_f/self.dt))
        self.compute_control_law()
        self.tau_array = []
        self.q_array = []
        self.q_r_array = []
        for _ in range (len(self.t)):
            self.tau_array.append(self.tau)
            self.q_array.append(self.q)
            self.q_r_array.append(self.q_r)
            self.advance_state(self.tau)
            self.compute_control_law()

    def plot(self):

        fig, ax = plt.subplots(1, 2)

        ax[0].plot(self.t, self.tau_array, '-g')
```

```
        # ax[0].plot(self.t, self.tau_array, '-r')

        ax[0].legend(["tau"])
        ax[0].set_ylabel("tau")
        ax[0].set_xlabel("t")
        ax[0].grid()
        ax[0].set_title("Control input tau v/s iteration t")

        ax[1].plot(self.t, self.q_array, '-g')
        ax[1].plot(self.t, self.q_r_array, ':b')
        ax[1].legend(["q", "q_ref"])
        ax[1].set_ylabel("q")
        ax[1].set_xlabel("t")
        ax[1].grid()
        ax[1].set_title("Control input q v/s iteration t")

        plt.show()

if __name__ == '__main__' :

    p_1 = 3.31
    p_3 = 0.16

    q_r = 0.3
    q_0 = 1.

    m = manipulator(p_1,p_3,q_0,q_r)
    m.simulate(120)
    m.plot()
```

PLOTS:




Question 3: Rigid Body controller using passivity

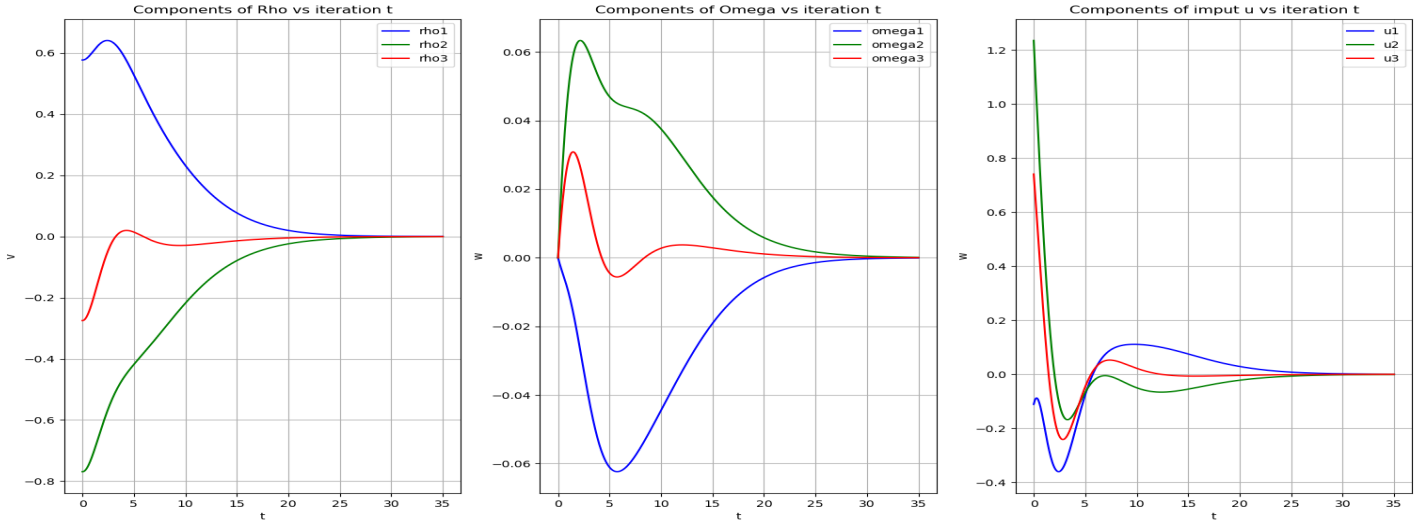
```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt, tanh, tan

class rigid_body:
   def __init__(self, J, rho_0, omega_0):
       self.J = J
       self.rho = rho_0
       print(self.rho)
       self.rho_cross = self.get_curl(self.rho)
       self.omega = omega_0
       self.I = np.identity(3)
       self.dt = 0.05
       self.k_1 = 1 # storage function gain
       self.k_2 = 12 # control law gain
       self.compute_dW_drho()

   def get_curl(self,vec):
       curl = np.array([[0      , -vec[2], vec[1] ],
                        [vec[2] , 0      , -vec[0]],
```

```
                        [-vec[1], vec[0] , 0      ]])
        return curl

    # now compute the jacobian of the storage function to compute the control law
    def compute_dW_drho(self):
        self.dW_drho = (2 * self.k_1 / (1 + np.dot(self.rho.T, self.rho))) * self.rho

    def advance_state(self, u) :
        dt = self.dt

        # dynamics
        self.rho_dot = np.dot((self.I + self.rho_cross + np.dot(self.rho,
self.rho.T)), self.omega)
        self.omega_dot = np.dot(np.linalg.inv(self.J),(-(np.cross(self.omega ,
np.dot(self.J,self.omega))) + u))
        self.y = self.omega
        self.compute_dW_drho()

        # update states
        self.omega += self.omega_dot * dt
        self.rho += self.rho_dot * dt
        self.rho_cross = self.get_curl(self.rho)

    def compute_control_law(self):
        self.nu = -self.k_2 * np.tanh(self.omega)
        self.u = self.nu - np.dot(self.dW_drho,(self.I + self.rho_cross +
np.dot(self.rho, self.rho.T))).T

    def simulate(self, t_f):
        self.t = np.linspace(0,t_f,int(t_f/self.dt))
        self.compute_control_law()
        self.u_1_array     = []
        self.u_2_array     = []
        self.u_3_array     = []
        self.rho_1_array   = []
        self.rho_2_array   = []
        self.rho_3_array   = []
        self.omega_1_array = []
        self.omega_2_array = []
        self.omega_3_array = []

        for _ in range (len(self.t)):
            self.rho_1_array.append(self.rho[0])
            self.rho_2_array.append(self.rho[1])
            self.rho_3_array.append(self.rho[2])
```

```
            self.omega_1_array.append(self.omega[0])
            self.omega_2_array.append(self.omega[1])
            self.omega_3_array.append(self.omega[2])
            self.u_1_array.append(self.u[0])
            self.u_2_array.append(self.u[1])
            self.u_3_array.append(self.u[2])
            self.advance_state(self.u)
            self.compute_control_law()

    def plot(self):

        fig, ax = plt.subplots(1, 3)

        ax[0].plot(self.t, self.rho_1_array, '-b')
        ax[0].plot(self.t, self.rho_2_array, '-g')
        ax[0].plot(self.t, self.rho_3_array, '-r')

        ax[0].legend(["rho1", "rho2", "rho3"])
        ax[0].set_ylabel("v")
        ax[0].set_xlabel("t")
        ax[0].grid()
        ax[0].set_title("Components of Rho vs iteration t")

        ax[1].plot(self.t, self.omega_1_array, '-b')
        ax[1].plot(self.t, self.omega_2_array, '-g')
        ax[1].plot(self.t, self.omega_3_array, '-r')

        ax[1].legend(["omega1", "omega2", "omega3"])
        ax[1].set_ylabel("w")
        ax[1].set_xlabel("t")
        ax[1].grid()
        ax[1].set_title("Components of Omega vs iteration t")

        ax[2].plot(self.t, self.u_1_array, '-b')
        ax[2].plot(self.t, self.u_2_array, '-g')
        ax[2].plot(self.t, self.u_3_array, '-r')

        ax[2].legend(["u1", "u2", "u3"])
        ax[2].set_ylabel("w")
        ax[2].set_xlabel("t")
        ax[2].grid()
        ax[2].set_title("Components of input u vs iteration t")

        plt.show()
```

```python
if __name__ == '__main__':

    alpha_1 = 0
    theta_initial = -35
    J       = np.array([[20   , 1.2  , 0.9  ],
                        [1.2  , 17   , 1.4  ],
                        [0.9  , 1.4  , 15   ]])

    orient  = np.array([1/sqrt(3), 1/sqrt(3), 1/sqrt(3)]).T
    k       = np.array([1, 0, 0]).T
    rho_0   = orient*np.cos(theta_initial) + np.cross(k,
orient)*np.sin(theta_initial) + k*(np.dot(k, orient))*(1-np.cos(theta_initial))

    omega_0 = np.array([0.0, 0.0 , 0.0]).T

    rot_dyn = rigid_body(J, rho_0, omega_0)
    rot_dyn.simulate(35)
    rot_dyn.plot()
```

Plots:



Q1 $M(q)\ddot{q} + C(q,\dot{q})\dot{q} + D\dot{q} + g(q) = u$

$q \in \mathbb{R}^n$ : generalized coordinates

$\dot{q} \in$ generalized velocities

$M(q) > 0 \in \mathbb{R}^{n \times n}$ : positive definite inertia matrix

$C(q,\dot{q}) \in \mathbb{R}^{n \times n}$ : centrifugal + coriolis force

$D \in \mathbb{R}^{n \times n}$ : viscous damping

$g(q) \in \mathbb{R}^n$ : gravity

It is known $\dot{M} - 2C$ is skew symmetric

$D > 0$ : is a positive definite

Objective : track constant reference $q_r$

So that

$e = q - q_r$ and dynamics of $e$

$$M(q)\ddot{e} + C(q,\dot{q})\dot{e} + D\dot{e} + g(q) = u$$

we want $e = 0$ as GAS

consider control

$u = g(q) - K_p e + v$ for feedback passivation

$K_p = K_p^T > 0$

$K_p \in \mathbb{R}^{n \times n}$

Assignment - 4

classmate
Date
Page

ANS: dynamics of the above feedback :

$e_1 = e$ $\qquad \dot{e}_1 = \dot{e} = e_2$

$e_2 = \dot{e}$ $\qquad \dot{e}_2 = \ddot{e}$

$$M(q)\dot{e}_2 + C(q,\dot{q})e_2 + De_2 + g(q) = u = g(q) - K_p e_1 + v$$

$$\therefore u = g(q) - K_p e_1 + v$$

Dynamics — state space equation

$\dot{e}_1 = e_2$

$$M(q)\dot{e}_2 = -[K_p e_1 + De_2 + C(q,\dot{q})e_2] + v$$

consider the storage function:

$$V(x) = \frac{1}{2}\dot{e}^T M(q)\dot{e} + \frac{1}{2}e^T K_p e$$

$$\dot{V}(x) = \frac{1}{2}\dot{e}^T M(q)\ddot{e} \times 2 + \frac{1}{2}\dot{e}^T \dot{M}(q)\dot{e} + \frac{1}{2}e^T K_p \dot{e} \times 2$$

$$= \dot{e}^T\left[M(q)\ddot{e} + \frac{1}{2}\dot{M}(q)\dot{e}\right] + e^T K_p \dot{e}$$

$$= \dot{e}^T\left[v - K_p e - D\dot{e} - C\dot{e} + \frac{\dot{M}}{2}\dot{e}\right] + e^T K_p \dot{e}$$

we know $(\dot{M} - 2C)$ is skew symmetric and $\dot{e}^T(\dot{M} - 2C)\dot{e}$ for skew-symmetric matrix is zero.

$$= \dot{e}^T[v - K_p e - D\dot{e}] + e^T K_p \dot{e}$$

$$= -\dot{e}^T D\dot{e} + \dot{e}^T v - \dot{e}^T K_p e + e^T K_p \dot{e}$$

$(\dot{e}^T K_p e)^T = (\dot{e}^T K_p e)$ : (scalar)

$e^T K_p \dot{e} = \dot{e}^T K_p e$

$$\dot{V}(x) = -\dot{e}^T D\dot{e} + \dot{e}^T v$$

$\because K_p = K_p^T > 0$ (given)

$$\dot{V}(x) = -\dot{e}^T D\dot{e} + \dot{e}^T v$$

$$\dot{V}(x) + \dot{e}^T D\dot{e} = \dot{e}^T v$$

$$\because e_1 = q - q_d \longrightarrow \dot{e} = \dot{q} - 0$$

output : $\dot{q}$

$$\dot{V}(x) + \dot{q}^T (D\dot{q}) = \dot{e}^T v$$

... we know $D > 0$ Positive definite

hence $\dot{q}^T(D\dot{q}) > 0 \quad \forall\, q \neq 0$

hence,

$$\dot{V}(x) + y^T S(y) = \dot{e}^T v \quad \ldots\rightarrow$$

Strictly output passive

$$\dot{V}(x) \leq \dot{e}^T v$$

$\dot{e}^T v$ : scalar

$$\therefore \dot{e}^T v = (\dot{e}^T v)^T = v^T \dot{e}$$

$$\dot{V}(x) \leq v^T \dot{e} \quad \ldots \text{passivity proved}$$

for $y = \dot{e} = e_2$

$$\dot{V}(x) \leq v^T y$$

for output chosen

checking for zero-state observability for the chosen output

$$y = \dot{e} = 0 \longrightarrow e_2 = 0 \quad [\because e_2 = \dot{e}]$$

$$\dot{e}_2 = 0 \text{ (derivative of 0 is 0)}$$

$$M(q)\dot{e}_2 = [-K_p e_1 - C e_2 - D e_2] + v \quad \text{when } v = 0$$

$$-K_p e_1 = 0 \longrightarrow e_1 = 0 \quad \because K_p > 0$$

hence $y = 0 \longrightarrow (e_1, e_2) = (0,0)$ when $v = 0$

hence → zero-state observable.

$$\dot{e}_1 = e_2$$

$$M(q)\dot{e}_2 = -[K_p e_1 + C(q,\dot{q})e_2 + D(q)e_2] + v$$

$$G = \begin{pmatrix} 0 \\ 1 \end{pmatrix}^T \qquad \frac{\partial V}{\partial e} = \begin{bmatrix} e_1^T K_p \\ e_2^T M + \frac{1}{2} e_2^T \dot{M} e_2 \end{bmatrix}^T$$

where

$$V = \frac{1}{2} e_2^T M e_2 + \frac{1}{2} e_1^T K_p e_1$$

$$\left[\left[\frac{\partial V(x)}{\partial e}\right] G(x)\right]^T = e_2^T M + \frac{1}{2} e_2^T \frac{\partial \dot{M}}{\partial e_2} e_2$$

$$v = -k \tanh(y) \quad \text{or} \quad -ky$$

To make actuator input bounded taking $v = -k \tanh(y)$ as $v$ input

Since $\dot{V}(x) = 0$ only at $\dot{q} = 0$

and $\dot{V}(x) < 0$ otherwise

$\therefore v = -k \tanh(y)$ and $y = 0$ only at $\dot{q} = 0$, $\dot{e} = 0$

and $\dot{V}(x) \leq v^T y$

hence locally asymptotically stable.

and

Since $x$ has no bounds → globally asymptotically stable.

(RUB)

Q3.

$u \in \mathbb{R}^3$ : thrust

$\dot{s} = [I + s^\times + ss^T]\omega$ — $s \in \mathbb{R}^3$ ← Rodrigues parameter

$J\dot{\omega} = -\omega \times J\omega + u$ — $\omega \in \mathbb{R}^3$

$J - J^T > 0$ ← inertia matrix

with $y = \omega$

with $\omega = 0 \longrightarrow \dot{s} = 0 \implies$ Stable

hence any $\omega(s) > 0$ (RUB)

will ensure $\dot{W}(s)\big|_{\omega=0} \leq 0$

considering $V(\omega) = \frac{1}{2}\omega^T J \omega > 0$

$\dot{V}(\omega) = \omega^T J\dot{\omega} + \frac{1}{2}\omega^T \dot{J}\omega$

$= \omega^T[-\omega \times J\omega + u]$

$= \omega^T[-\omega \times J\omega + u]$

$= \omega^T[J\omega \times \omega + u]$

$= \omega^T(J\omega \times \omega) + \omega^T u$

$= -\omega^T(\omega \times J\omega) + \omega^T u$

negative definite hence

$\dot{V} < \omega^T u = u^T\omega$

$\dot{V} < u^T\omega$

where $y = \omega$

hence Proved

$\boxed{\dot{V} < u^T y}$

Taking Storage function

$U(\omega, s) = V(\omega) + W(s)$

$\dot{U}(\omega, s) = \dot{V}(\omega) + \dot{W}(s)$

$= \underbrace{-\omega^T(\omega \times J\omega)}_{\text{-ve semi definite}} + \frac{\partial W}{\partial s}\dot{s} + \omega^T u$

$\dot{U}(\omega, s) \leq \frac{\partial W}{\partial s}[I + [s] + ss^T]\omega + \omega^T u$

$$\dot{u}(\omega,s) \leq \frac{\partial w}{\partial s}\left[I + [s] + ss^T\right]\omega + u^T\omega$$

$\because u^T\omega = \omega^T u$ (scalar)

for $u = v - \left[\frac{\partial w}{\partial s}\left[I + [s] + ss^T\right]\right]^T$

with $W(s) = k \ln(1 + s^T s)$

find $u$ ;

and proove

the system to be zero state observable in $(v, y)$

$$\frac{\partial w}{\partial s} = \frac{k \cdot 2(s^T)}{(1+s^T s)}$$

$$\frac{\partial w}{\partial s} = \frac{2k \cdot s^T}{1+s^T s}$$

$$\dot{u}(\omega,s) \leq \frac{\partial w}{\partial s}\left[I + [s] + ss^T\right]\omega + v^T\omega - \left[\frac{\partial w}{\partial s}\left[I + [s] + ss^T\right]\right]\omega$$

$$\dot{u}(\omega,s) \leq v^T\omega \longrightarrow \dot{u}(\omega,s) \leq v^T y$$

$\omega = y$

to prove zero state observability $\rightarrow$ $(v, y)$

for $y = 0 \longrightarrow \omega = 0$ ; the input $v = 0$

$\omega = 0 \longrightarrow \dot{\omega} = 0$

$\omega = 0 \longrightarrow \dot{s} = 0$

$$J\dot{\omega} = -\omega \times J\omega + v - \left[\frac{\partial W(s)}{\partial s}\left[I + [s] + ss^T\right]\right]^T$$

hence $\frac{\partial W}{\partial s}\left[I + [S] + SS^T\right] = 0$

$$\frac{2k}{(1+S^TS)} S^T\left[I + [S] + SS^T\right] = 0$$

$k \neq 0$

$1 + S^TS \neq 0$ ∵ argument of $\ln(a)$ $\quad a > 0$

$I + [S] + SS^T = 0$ OR $S^T = 0$

$$\begin{bmatrix} 1 & -S_z & S_y \\ S_z & 1 & -S_x \\ -S_y & S_x & 1 \end{bmatrix} + \begin{bmatrix} S_x^2 & S_xS_y & S_xS_z \\ S_xS_y & S_y^2 & S_yS_z \\ S_xS_z & S_yS_z & S_z^2 \end{bmatrix}$$

$S_x\ S_x\ S_y\ S_x\ S_y\ S_z$

$1 + S_x^2 \neq 0$

$1 + S_y^2 \neq 0$

$1 + S_z^2 \neq 0$

hence only $S^T = 0$ ⇒ the only solution

$S = 0$ ∵ $S^T = 0$

hence zero-state observability proved

∴ $u(w,y) \leq v^Tw$ ⟶ proves passivity

since $y = w$

choosing $\boxed{v = -k\tanh(y)}$

$$\boxed{u = -k\tanh(y) - \left[\frac{\partial W}{\partial s}\left[I + [S] + SS^T\right]\right]^T}$$