```
from sympy import *
import math
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

th, al, a, d, th0, th1, th2, th3, th4, th5 = symbols('theta alpha a d theta0 theta1
theta2 theta3 theta4 theta5', real=True)
#please give all the inputs in degrees in place of the zero for all the thetas
#please uncomment all below lines to get the simplified version of all matrices
#(teta not equal to zero)
# th0 = 0
# th1 = 0
# th2 = 0
# th3 = 0
# th4 = 0
# th5 = 0
# # converts the theta value into radians
# th0 = (sp.rad(th0))
# th1 = (sp.rad(th1))
# th2 = (sp.rad(th2))
# th3 = (sp.rad(th3))
# th4 = (sp.rad(th4))
# th5 = (sp.rad(th5))
a =[]
b=[]


"""To compute the DH parameters matrix"""
def Transformation_matrix(a, al, d, th):
      return Matrix([
      [sp.cos(th), -sp.sin(th) * sp.cos(al), sp.sin(th) * sp.sin(al), a *
      sp.cos(th)],
      [sp.sin(th), sp.cos(th) * sp.cos(al), -sp.cos(th) * sp.sin(al), a *
      sp.sin(th)],
      [0, sp.sin(al), sp.cos(al), d],
      [0, 0, 0, 1]
      ])

m = 12.92
l=1.0
lc= 0.5
I= (m*l*l)/12.0


g = 9.8


T1 = (Transformation_matrix(0, sp.pi/2, 1.2, N(rad(th0))))
T2 = (Transformation_matrix(1, 0, 0, N(rad(th1+sp.pi/4))))
T3 = (Transformation_matrix(1, 0, 0, N(rad(th2-sp.pi/4))))
T02 = (T1 * T2)
T03 = (T1 * T2 * T3)

z1 = T1[2,3]
z2 = T02[2,3]
z3 = T03[2,3]
```

```
q2=0.0
q3=0.0
q2_dot=0.0
q3_dot=0.0



PE = g*((m*z1/2)+(m*(z1+((z2-z1)/2)))+(m*(z2+((z3-z2)/2))))

# print(PE)
G =  Matrix([[diff(PE, th0)],
      [diff(PE, th1)],
      [diff(PE, th2)],
      ])
M_q = Matrix([[0, 0, 0],
              [0, m*lc*lc + m*(l*l+l*l+(2*l*l*sp.cos(q2)))+(2.0*I), m*((lc*lc)
+l*l*sp.cos(q2))+I],
              [0, m*((lc*lc)+(l*lc*sp.cos(q2)))+I, (m*lc*lc)+I]])

C_q = Matrix([[0, 0, 0],
            [0, -(m*l*lc*q3_dot*sp.sin(q3)), -(m*l*lc*(q2_dot+q3_dot)*sp.sin(q3))],
            [0, (m*l*lc*q2_dot*sp.sin(q3)), 0]])

# pprint(G)



F = Matrix([[-10],
            [0],
            [0],
            [0],
            [0],
            [0]])

torque1 = []
torque2 = []
torque3 = []
# print("gravity matrix in terms of joint angle is given by ")
# print("\n")
# pprint(G)

dy = 0
wx = 0
wy = 0
wz = 0
t = 0

def j_matrix (th0,th1,th2):
    T1 = (Transformation_matrix(0, sp.pi/2, 1.2, (th0)))
    T2 = (Transformation_matrix(1, 0, 0, (th1+sp.pi/4)))
    T3 = (Transformation_matrix(1, 0, 0, (th2-sp.pi/4)))

    # t1= Transformation_matrix(0,N(rad(90)),1.2, (a1))
    # t2= Transformation_matrix(1,0, 0,(a2+N(sp.pi/4)) )
    # t3 = Transformation_matrix(1, 0, 0,(a3-N(sp.pi/4)))

    T06 = ((T1 * T2 * T3))
    T02 =(  T1 * T2)
    T03 = (T1 * T2 * T3)
    # print("T03..........................................")
    # pprint(N(T03))
    O06 = np.array([[T06[0, 3]], [T06[1, 3]], [T06[2, 3]]])
    O01 = np.array([[T1[0, 3]], [T1[1, 3]], [T1[2, 3]]])
    O02 = np.array([[T02[0, 3]], [T02[1, 3]], [T02[2, 3]]])

    R0 = sp.Matrix([[1, 0, 0, 0],
                  [0, 1, 0, 0],
                  [0, 0, 1, 0],
                  [0, 0, 0, 1]])

    R01 = np.array(T1[:3, :3])
    R02 = np.array(T02[:3, :3])
    R00 = np.array(R0[:3,:3])

    d1 = O06 - O01
    d2 = O06 - O02

    first_col_1 = np.cross((R00[:,-1]).flatten(),O06.flatten())
    second_col_1 = np.cross((R01[:, -1]).flatten(), d1.flatten())
    third_col_1 = np.cross((R02[:,-1]).flatten(), d2.flatten())


    first_col_2 = R00[:,-1]
    second_col_2 = R01[:,-1]
    third_col_2 = R02[:,-1]

    j1 = np.concatenate((first_col_1, first_col_2), axis=None)
    j2 = np.concatenate((second_col_1, second_col_2), axis=None)
    j3 = np.concatenate((third_col_1, third_col_2), axis=None)

    J= np.column_stack((j1, j2, j3))
    J_matrix = (sp.Matrix(J))
    return J_matrix


# q = np.matrix([[0], [rad(45)], [rad(-45)]])
q = np.matrix([[0], [0], [0]])
r = 0.1
Ti = 0
# pprint(T03)
dt = 0.1
q_dot = 0
Time = 20
j = N(j_matrix(0,0,0))
# print("jocobian matrix at intial instant is")
# print("\n")
# pprint (j)
# print("\n")


j_actual = j_matrix (th0,th1,th2)

# print("Actual jocobian matrix is")
print("\n")
# pprint (j_actual)
# print("\n")

jac_float = np.matrix(j).astype(np.float64)
```

```
J_inv = np.linalg.pinv(jac_float)


# Lists to store trajectory points
x_values = []
y_values = []
z_values = []

q1_ = []
q2_ = []
q3_ = []



print("calculatinhg.... please wait it takes time..............")
print("\n")
while (Ti <= Time):

    # x_dot = r * np.cos(av*Ti) * av
    x_dot = 0
    # z_dot = r * np.sin(av*Ti) * av
    z_dot = -(0.707/20)
    y_dot = 0
    # z_dot = 0
    # y_dot = 0
    # y_dot = -(1.707/20)
    # y_dot = 0

    E = np.matrix([[x_dot], [y_dot], [z_dot], [0], [0], [0]])
    a=q_dot
    q_dot = J_inv * E
    ddq = (q_dot-a)/dt
    q = q + q_dot * dt
    Q2_dot = q_dot[1]
    Q3_dot = q_dot[2]
    # print(q)


    [a1, a2, a3] = [q[i].item() for i in range(3)]
    jac = j_matrix(a1,a2,a3)
    jac_float = np.matrix(jac).astype(np.float64)
    J_inv = np.linalg.pinv(jac_float)

    to0= Transformation_matrix(0,rad(0),0,(0))

    t1= Transformation_matrix(0,N(rad(90)),1.2, (a1))
    t2= Transformation_matrix(1,0, 0,(a2+N(sp.pi/4)) )
    t3 = Transformation_matrix(1, 0, 0,(a3-N(sp.pi/4)))

    t = (to0*t1*t2*t3)
    # print(N(t[:, 3]))


    x_values.append((t[0,3]))
    # x_values.append(1.707)
    y_values.append((t[1,3]))
    # y_values.append(2)
    z_values.append((t[2,3]))
    G1 = G.subs({th0: a1, th1: a2, th2: a3})
    M_q1 = M_q.subs({q2: rad(a2), th3: rad(a3)})
    C_q1 = C_q.subs({q2: rad(a2), q3: rad(a3), q2_dot:Q2_dot, q3_dot:Q3_dot})
    # pprint(G1)
    J_transpose = jac.T


    tou = (M_q1*ddq)+(C_q1*q_dot)+G1 - (J_transpose*F)

    # tou = (M_q1*ddq)+(C_q1*q_dot)

    # tou = G1 - (J_transpose*F)

    # q1_.append(q[0])
    # q2_.append(q[1])
    # q3_.append(q[2])

    tou = tou
    if(Ti>=0.1):

        q1_.append(q[0])
        q2_.append(q[1])
        q3_.append(q[2])
        # print(q1_)
        # print(q2_)
        # print(q3_)

        torque1.append(round(tou[0],4))
        torque2.append(round(tou[1],5))
        torque3.append(round(tou[2],2))

    # torque1.append(tou[0])
    # torque2.append(tou[1])
    # torque3.append(tou[2])
    # print(N(tou[0]))
    Ti = Ti + dt

# 3D Plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(x_values, y_values, z_values, label='End Effector Trajectory')
ax.set_xlabel('X coordinates')
ax.set_ylabel('Y coordinates')
ax.set_zlabel('Z coordinates')
ax.set_title('Trajectory of the robot')
# plt.show() uncomment this line to see the trajectory of the robot


time_array = np.arange(0, Time-dt, dt)

# Plot torque_1
plt.subplot(2, 3, 1)
plt.plot(time_array, torque1)
plt.title('Torque 1 vs Time')
plt.xlabel('Time (s)')
plt.ylabel('Torque 1 (Nm)')


# Plot torque_2
```

```
plt.subplot(2, 3, 2)
plt.plot(time_array, torque2)
plt.title('Torque 2 vs Time')
plt.xlabel('Time (s)')
plt.ylabel('Torque 2 (Nm)')

# Plot torque_3
plt.subplot(2, 3, 3)
plt.plot(time_array, torque3)
plt.title('Torque 3 vs Time')
plt.xlabel('Time (s)')
plt.ylabel('Torque 3 (Nm)')

# Plot joint angle 1
plt.subplot(2, 3, 4)
plt.plot(time_array, np.squeeze(q1_))
plt.title('Joint Angle 1 vs Time')
plt.xlabel('Time (s)')
plt.ylabel('Joint Angle 1')

# # Plot joint angle 2
plt.subplot(2, 3, 5)
plt.plot(time_array, np.squeeze(q2_))
plt.title('Joint Angle 2 vs Time')
plt.xlabel('Time (s)')
plt.ylabel('Joint Angle 2')

# # Plot joint angle 3
plt.subplot(2, 3, 6)
plt.plot(time_array, np.squeeze(q3_))
plt.title('Joint Angle 3 vs Time')
plt.xlabel('Time (s)')
plt.ylabel('Joint Angle 3')


plt.tight_layout()
ax.legend()

# Show the plots
plt.show()
```