# Introduction

I seperately made each part a single python script. Each script is independent.

# Part 1

In [ ]:
```
import numpy as np
import modern_robotics as mr


# Parameters from https://hades.mech.northwestern.edu/images/d/d9/UR5-parame
M01 = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0.089159], [0, 0, 0, 1]]
M12 = [[0, 0, 1, 0.28], [0, 1, 0, 0.13585], [-1, 0, 0, 0], [0, 0, 0, 1]]
M23 = [[1, 0, 0, 0], [0, 1, 0, -0.1197], [0, 0, 1, 0.395], [0, 0, 0, 1]]
M34 = [[0, 0, 1, 0], [0, 1, 0, 0], [-1, 0, 0, 0.14225], [0, 0, 0, 1]]
M45 = [[1, 0, 0, 0], [0, 1, 0, 0.093], [0, 0, 1, 0], [0, 0, 0, 1]]
M56 = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0.09465], [0, 0, 0, 1]]
M67 = [[1, 0, 0, 0], [0, 0, 1, 0.0823], [0, -1, 0, 0], [0, 0, 0, 1]]
G1 = np.diag([0.010267495893, 0.010267495893,  0.00666, 3.7, 3.7, 3.7])
G2 = np.diag([0.22689067591, 0.22689067591, 0.0151074, 8.393, 8.393, 8.393])
G3 = np.diag([0.049443313556, 0.049443313556, 0.004095, 2.275, 2.275, 2.275]
G4 = np.diag([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219])
G5 = np.diag([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219])
G6 = np.diag([0.0171364731454, 0.0171364731454, 0.033822, 0.1879, 0.1879, 0.
Glist = [G1, G2, G3, G4, G5, G6]
Mlist = [M01, M12, M23, M34, M45, M56, M67]
Slist = [[0,         0,         0,         0,        0,        0],
        [0,         1,         1,         1,        0,        1],
        [1,         0,         0,         0,       -1,        0],
        [0, -0.089159, -0.089159, -0.089159, -0.10915, 0.005491],
        [0,         0,         0,         0,  0.81725,        0],
        [0,         0,     0.425,   0.81725,        0,  0.81725]]


def Puppet(thetalist,
           dthetalist,
           g,
           Mlist,
           Slist,
           Glist,
           t,
           dt,
           damping,
           stiffness,
           restLength):
    """
    Simulate the system with the given parameters.

    Args:
        thetalist: A list of initial joint angles.
        dthetalist: A list of initial joint velocities.
```

```
        g: The gravity vector.
        Mlist: A list of link transposes.
        Slist: A list of joint screw axes.
        Glist: A list of link inertia matrices.
        t: The total time of the simulation.
        dt: The timestep of the simulation.
        damping: The damping coefficient.
        stiffness: The spring stiffness.
        restLength: The spring rest length.

    Returns:
        thetamat: A matrix of joint angles at each timestep.
        dthetamat: A matrix of joint velocities at each timestep.
    """
    assert len(thetalist) == len(dthetalist)
    n = len(thetalist)
    N = int(t/dt)

    thetamat = np.zeros((N + 1, n))
    thetamat[0] = thetalist

    dthetamat = np.zeros((N + 1, n))
    dthetamat[0] = dthetalist

    for idx in range(N):
    #    print(str(round(idx/N*100, 2))+' %')
        damping_t = -damping * dthetalist
        thdd = mr.ForwardDynamics(
            thetalist,
            dthetalist,
            damping_t,
            g,
            np.zeros(n),
            Mlist,
            Glist,
            Slist)

        th, dth = mr.EulerStep(
            thetalist,
            dthetalist,
            thdd,
            dt)

        thetalist = th
        dthetalist = dth

        thetamat[idx+1] = thetalist
        dthetamat[idx+1] = dthetalist

    return thetamat, dthetamat


def main():
    thetalist = np.array([0, 0, 0, 0, 0, 0])
    dhetalist = np.array([0, 0, 0, 0, 0, 0])
    g = np.array([0, 0, -9.81])
```

```
    thetamat, dthetamat = Puppet(
        thetalist,
        dhetalist,
        g,
        Mlist,
        Slist,
        Glist,
        5,
        0.001,
        0.0,
        None,
        None)

    np.savetxt('part1a.csv', thetamat, delimiter=',')

    thetamat, dthetamat = Puppet(
        thetalist,
        dhetalist,
        g,
        Mlist,
        Slist,
        Glist,
        5,
        0.05,
        0.0,
        None,
        None)

    np.savetxt('part1b.csv', thetamat, delimiter=',')


if __name__ == '__main__':
    main()
```

To calculate the total energy, we can use Hamiltonian as a metric. It's the sum of Kinetic Energy and Potential Energy in this system.

# Part 2

In [ ]:

```
import numpy as np
import modern_robotics as mr


# Parameters from https://hades.mech.northwestern.edu/images/d/d9/UR5-parame
M01 = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0.089159], [0, 0, 0, 1]]
M12 = [[0, 0, 1, 0.28], [0, 1, 0, 0.13585], [-1, 0, 0, 0], [0, 0, 0, 1]]
M23 = [[1, 0, 0, 0], [0, 1, 0, -0.1197], [0, 0, 1, 0.395], [0, 0, 0, 1]]
M34 = [[0, 0, 1, 0], [0, 1, 0, 0], [-1, 0, 0, 0.14225], [0, 0, 0, 1]]
M45 = [[1, 0, 0, 0], [0, 1, 0, 0.093], [0, 0, 1, 0], [0, 0, 0, 1]]
M56 = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0.09465], [0, 0, 0, 1]]
M67 = [[1, 0, 0, 0], [0, 0, 1, 0.0823], [0, -1, 0, 0], [0, 0, 0, 1]]
G1 = np.diag([0.010267495893, 0.010267495893,  0.00666, 3.7, 3.7, 3.7])
G2 = np.diag([0.22689067591, 0.22689067591, 0.0151074, 8.393, 8.393, 8.393])
```

```
G3 = np.diag([0.049443313556, 0.049443313556, 0.004095, 2.275, 2.275, 2.275]
G4 = np.diag([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219])
G5 = np.diag([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219])
G6 = np.diag([0.0171364731454, 0.0171364731454, 0.033822, 0.1879, 0.1879, 0.
Glist = [G1, G2, G3, G4, G5, G6]
Mlist = [M01, M12, M23, M34, M45, M56, M67]
Slist = [[0,         0,         0,         0,        0,        0],
        [0,         1,         1,         1,        0,        1],
        [1,         0,         0,         0,       -1,        0],
        [0, -0.089159, -0.089159, -0.089159, -0.10915, 0.005491],
        [0,         0,         0,         0,  0.81725,        0],
        [0,         0,     0.425,   0.81725,        0,  0.81725]]


def Puppet(thetalist,
           dthetalist,
           g,
           Mlist,
           Slist,
           Glist,
           t,
           dt,
           damping,
           stiffness,
           restLength):
    """
    Simulate the system with the given parameters.

    Args:
        thetalist: A list of initial joint angles.
        dthetalist: A list of initial joint velocities.
        g: The gravity vector.
        Mlist: A list of link transposes.
        Slist: A list of joint screw axes.
        Glist: A list of link inertia matrices.
        t: The total time of the simulation.
        dt: The timestep of the simulation.
        damping: The damping coefficient.
        stiffness: The spring stiffness.
        restLength: The spring rest length.

    Returns:
        thetamat: A matrix of joint angles at each timestep.
        dthetamat: A matrix of joint velocities at each timestep.
    """
    assert len(thetalist) == len(dthetalist)
    n = len(thetalist)
    N = int(t/dt)

    thetamat = np.zeros((N + 1, n))
    thetamat[0] = thetalist

    dthetamat = np.zeros((N + 1, n))
    dthetamat[0] = dthetalist

    for idx in range(N):
```

```
    #    print(str(round(idx/N*100, 2))+' %')
        damping_t = -damping * dthetalist
        thdd = mr.ForwardDynamics(
            thetalist,
            dthetalist,
            damping_t,
            g,
            np.zeros(n),
            Mlist,
            Glist,
            Slist)

        th, dth = mr.EulerStep(
            thetalist,
            dthetalist,
            thdd,
            dt)

        thetalist = th
        dthetalist = dth

        thetamat[idx+1] = thetalist
        dthetamat[idx+1] = dthetalist

    return thetamat, dthetamat


def main():
    thetalist = np.array([0, 0, 0, 0, 0, 0])
    dhetalist = np.array([0, 0, 0, 0, 0, 0])
    g = np.array([0, 0, -9.81])

    thetamat, dthetamat = Puppet(
        thetalist,
        dhetalist,
        g,
        Mlist,
        Slist,
        Glist,
        5,
        0.01,
        3.0,
        None,
        None)

    np.savetxt('part2a.csv', thetamat, delimiter=',')

    thetamat, dthetamat = Puppet(
        thetalist,
        dhetalist,
        g,
        Mlist,
        Slist,
        Glist,
        5,
        0.01,
```

```
        -0.02,
        None,
        None)

    np.savetxt('part2b.csv', thetamat, delimiter=',')


if __name__ == '__main__':
    main()
```

When I choose the damping parameter to be large, the csv file would generate nan values, thus it fails to visualize. This is because large damping can lead to rapid change in the system state, thus bringing instability to the euler integration process. However, after I changed the dt from 0.01 to 0.001 (when damping = 10.0), this problem is addressed.

# Part 3

In [ ]:

```
import numpy as np
import modern_robotics as mr


# Parameters from https://hades.mech.northwestern.edu/images/d/d9/UR5-parame
M01 = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0.089159], [0, 0, 0, 1]]
M12 = [[0, 0, 1, 0.28], [0, 1, 0, 0.13585], [-1, 0, 0, 0], [0, 0, 0, 1]]
M23 = [[1, 0, 0, 0], [0, 1, 0, -0.1197], [0, 0, 1, 0.395], [0, 0, 0, 1]]
M34 = [[0, 0, 1, 0], [0, 1, 0, 0], [-1, 0, 0, 0.14225], [0, 0, 0, 1]]
M45 = [[1, 0, 0, 0], [0, 1, 0, 0.093], [0, 0, 1, 0], [0, 0, 0, 1]]
M56 = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0.09465], [0, 0, 0, 1]]
M67 = [[1, 0, 0, 0], [0, 0, 1, 0.0823], [0, -1, 0, 0], [0, 0, 0, 1]]
M07 = np.array(M01) @ np.array(M12) @ np.array(M23) @ np.array(M34) @\
        np.array(M45) @ np.array(M56) @ np.array(M67)
G1 = np.diag([0.010267495893, 0.010267495893,  0.00666, 3.7, 3.7, 3.7])
G2 = np.diag([0.22689067591, 0.22689067591, 0.0151074, 8.393, 8.393, 8.393])
G3 = np.diag([0.049443313556, 0.049443313556, 0.004095, 2.275, 2.275, 2.275]
G4 = np.diag([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219])
G5 = np.diag([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219])
G6 = np.diag([0.0171364731454, 0.0171364731454, 0.033822, 0.1879, 0.1879, 0.
Glist = [G1, G2, G3, G4, G5, G6]
Mlist = [M01, M12, M23, M34, M45, M56, M67]
Slist = [[0,         0,         0,         0,        0,        0],
        [0,         1,         1,         1,        0,        1],
        [1,         0,         0,         0,       -1,        0],
        [0, -0.089159, -0.089159, -0.089159, -0.10915, 0.005491],
        [0,         0,         0,         0,  0.81725,        0],
        [0,         0,     0.425,   0.81725,        0,  0.81725]]


def referencePos():
    """
    Returns the reference position of the spring.

    Returns:
        springPos: The reference position of the spring.
```

```python
    """
    springPos = np.array([0, 1, 1])
    return springPos


def springForce(Slist, thetalist, restLength, stiffness):
    """
    Returns the wrench in the ee frame.

    Args:
        Slist: A list of joint screw axes.
        thetalist: A list of joint angles.
        restLength: The spring rest length.
        stiffness: The spring stiffness.

    Returns:
        wrench: The wrench in the ee frame.
    """
    T_be = mr.FKinSpace(M07, Slist, thetalist)
    T_eb = mr.TransInv(T_be)
    p_be = T_be[:3, 3]
    dL = np.linalg.norm(referencePos() - p_be) - restLength
    F_scalar = - stiffness * dL
    F_vector = (referencePos() - p_be)/np.linalg.norm(referencePos() - p_be)
    F = F_scalar * F_vector
    F_e = T_eb @ np.array([F[0], F[1], F[2], 1]).T
    wrench = np.array([0, 0, 0, F_e[0], F_e[1], F_e[2]])
    return wrench


def Puppet(thetalist,
           dthetalist,
           g,
           Mlist,
           Slist,
           Glist,
           t,
           dt,
           damping,
           stiffness,
           restLength):
    """
    Simulate the system with the given parameters.

    Args:
        thetalist: A list of initial joint angles.
        dthetalist: A list of initial joint velocities.
        g: The gravity vector.
        Mlist: A list of link transposes.
        Slist: A list of joint screw axes.
        Glist: A list of link inertia matrices.
        t: The total time of the simulation.
        dt: The timestep of the simulation.
        damping: The damping coefficient.
        stiffness: The spring stiffness.
        restLength: The spring rest length.
```

```
    Returns:
        thetamat: A matrix of joint angles at each timestep.
        dthetamat: A matrix of joint velocities at each timestep.
    """
    assert len(thetalist) == len(dthetalist)
    n = len(thetalist)
    N = int(t/dt)

    thetamat = np.zeros((N + 1, n))
    thetamat[0] = thetalist

    dthetamat = np.zeros((N + 1, n))
    dthetamat[0] = dthetalist

    for idx in range(N):
        # print(str(round(idx/N*100, 2))+' %')
        damping_t = -damping * dthetalist

        wrench = springForce(Slist, thetalist, restLength, stiffness)
        thdd = mr.ForwardDynamics(
            thetalist,
            dthetalist,
            damping_t,
            g,
            wrench,
            Mlist,
            Glist,
            Slist)

        th, dth = mr.EulerStep(
            thetalist,
            dthetalist,
            thdd,
            dt)

        thetalist = th
        dthetalist = dth

        thetamat[idx+1] = thetalist
        dthetamat[idx+1] = dthetalist

    return thetamat, dthetamat


def main():
    thetalist = np.array([0, 0, 0, 0, 0, 0])
    dhetalist = np.array([0, 0, 0, 0, 0, 0])
    g = np.array([0, 0, 0])

    thetamat, dthetamat = Puppet(
        thetalist,
        dhetalist,
        g,
        Mlist,
        Slist,
```

```
        Glist,
        10,
        0.01,
        0.0,
        5.0,
        0.0)

    np.savetxt('part3a.csv', thetamat, delimiter=',')

    thetamat, dthetamat = Puppet(
        thetalist,
        dhetalist,
        g,
        Mlist,
        Slist,
        Glist,
        10,
        0.01,
        2.0,
        20.0,
        0.0)

    np.savetxt('part3b.csv', thetamat, delimiter=',')


if __name__ == '__main__':
    main()
```

The total energy of the system is supposed to be conserved when we give damping = 0, thus the behavior (swinging around) of the system makes sense.

When I set the stiffness to be large, the system oscillates much harder because the force also becomes large.

For part3a, the stiffness = 5.0, damping = 0.0

For part3b, the stiffness = 20.0, damping = 2.0

# Part 4

In [ ]:

```
import numpy as np
import modern_robotics as mr


# Parameters from https://hades.mech.northwestern.edu/images/d/d9/UR5-parame
M01 = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0.089159], [0, 0, 0, 1]]
M12 = [[0, 0, 1, 0.28], [0, 1, 0, 0.13585], [-1, 0, 0, 0], [0, 0, 0, 1]]
M23 = [[1, 0, 0, 0], [0, 1, 0, -0.1197], [0, 0, 1, 0.395], [0, 0, 0, 1]]
M34 = [[0, 0, 1, 0], [0, 1, 0, 0], [-1, 0, 0, 0.14225], [0, 0, 0, 1]]
M45 = [[1, 0, 0, 0], [0, 1, 0, 0.093], [0, 0, 1, 0], [0, 0, 0, 1]]
M56 = [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0.09465], [0, 0, 0, 1]]
M67 = [[1, 0, 0, 0], [0, 0, 1, 0.0823], [0, -1, 0, 0], [0, 0, 0, 1]]
M07 = np.array(M01) @ np.array(M12) @ np.array(M23) @ np.array(M34) @\
```

```
        np.array(M45) @ np.array(M56) @ np.array(M67)
G1 = np.diag([0.010267495893, 0.010267495893,  0.00666, 3.7, 3.7, 3.7])
G2 = np.diag([0.22689067591, 0.22689067591, 0.0151074, 8.393, 8.393, 8.393])
G3 = np.diag([0.049443313556, 0.049443313556, 0.004095, 2.275, 2.275, 2.275]
G4 = np.diag([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219])
G5 = np.diag([0.111172755531, 0.111172755531, 0.21942, 1.219, 1.219, 1.219])
G6 = np.diag([0.0171364731454, 0.0171364731454, 0.033822, 0.1879, 0.1879, 0.
Glist = [G1, G2, G3, G4, G5, G6]
Mlist = [M01, M12, M23, M34, M45, M56, M67]
Slist = [[0,         0,         0,         0,        0,        0],
        [0,         1,         1,         1,        0,        1],
        [1,         0,         0,         0,       -1,        0],
        [0, -0.089159, -0.089159, -0.089159, -0.10915, 0.005491],
        [0,         0,         0,         0,  0.81725,        0],
        [0,         0,     0.425,   0.81725,        0,  0.81725]]


def referencePos(t):
    """
    Returns the reference position of the spring oscillating in the y direct

    Args:
        t: The time.

    Returns:
        springPos: The reference position of the spring.
    """
    x, z = 1, 1
    y = np.cos(2 * np.pi * (t / 5))
    springPos = np.array([x, y, z])
    return springPos

def springForce(Slist, thetalist, restLength, stiffness, t):
    """
    Returns the wrench in the ee frame.

    Args:
        Slist: A list of joint screw axes.
        thetalist: A list of joint angles.
        restLength: The spring rest length.
        stiffness: The spring stiffness.

    Returns:
        wrench: The wrench in the ee frame.
    """
    T_be = mr.FKinSpace(M07, Slist, thetalist)
    T_eb = mr.TransInv(T_be)
    p_be = T_be[:3, 3]
    dL = np.linalg.norm(referencePos(t) - p_be) - restLength
    F_scalar = - stiffness * dL
    F_vector = (referencePos(t) - p_be)/np.linalg.norm(referencePos(t) - p_b
    F = F_scalar * F_vector
    F_e = T_eb @ np.array([F[0], F[1], F[2], 1]).T
    wrench = np.array([0, 0, 0, F_e[0], F_e[1], F_e[2]])
    return wrench
```

```
def Puppet(thetalist,
           dthetalist,
           g,
           Mlist,
           Slist,
           Glist,
           t,
           dt,
           damping,
           stiffness,
           restLength):
    """
    Simulate the system with the given parameters.

    Args:
        thetalist: A list of initial joint angles.
        dthetalist: A list of initial joint velocities.
        g: The gravity vector.
        Mlist: A list of link transposes.
        Slist: A list of joint screw axes.
        Glist: A list of link inertia matrices.
        t: The total time of the simulation.
        dt: The timestep of the simulation.
        damping: The damping coefficient.
        stiffness: The spring stiffness.
        restLength: The spring rest length.

    Returns:
        thetamat: A matrix of joint angles at each timestep.
        dthetamat: A matrix of joint velocities at each timestep.
    """
    assert len(thetalist) == len(dthetalist)
    n = len(thetalist)
    N = int(t/dt)

    thetamat = np.zeros((N + 1, n))
    thetamat[0] = thetalist

    dthetamat = np.zeros((N + 1, n))
    dthetamat[0] = dthetalist

    current_t = 0

    for idx in range(N):
        # print(str(round(idx/N*100, 2))+' %')
        damping_t = -damping * dthetalist

        wrench = springForce(Slist, thetalist, restLength, stiffness, currer
        thdd = mr.ForwardDynamics(
            thetalist,
            dthetalist,
            damping_t,
            g,
            wrench,
            Mlist,
```

```
                Glist,
                Slist)

            th, dth = mr.EulerStep(
                thetalist,
                dthetalist,
                thdd,
                dt)

            thetalist = th
            dthetalist = dth

            thetamat[idx+1] = thetalist
            dthetamat[idx+1] = dthetalist

            current_t += dt

    return thetamat, dthetamat


def main():
    thetalist = np.array([0, 0, 0, 0, 0, 0])
    dhetalist = np.array([0, 0, 0, 0, 0, 0])
    g = np.array([0, 0, 0])

    thetamat, dthetamat = Puppet(
        thetalist,
        dhetalist,
        g,
        Mlist,
        Slist,
        Glist,
        10,
        0.01,
        2.0,
        100.0,
        0.0)

    np.savetxt('part4.csv', thetamat, delimiter=',')


if __name__ == '__main__':
    main()
```