MECE 4510: Evolutionary Computation and Design

Project – Phase B

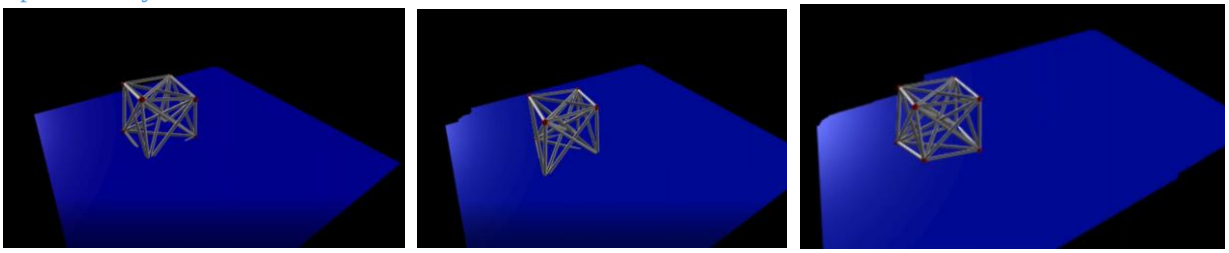
Jerry Zhang | UNI: jz2966 | Grace Hours Remaining: 62 +4 = 66
Zhengyang Du | UNI: zd2219 | Grace Hours Remaining: 57 +4 = 61

Instructor: Professor Hod Lipson
Date Submitted: 11/30/2018

RESULTS

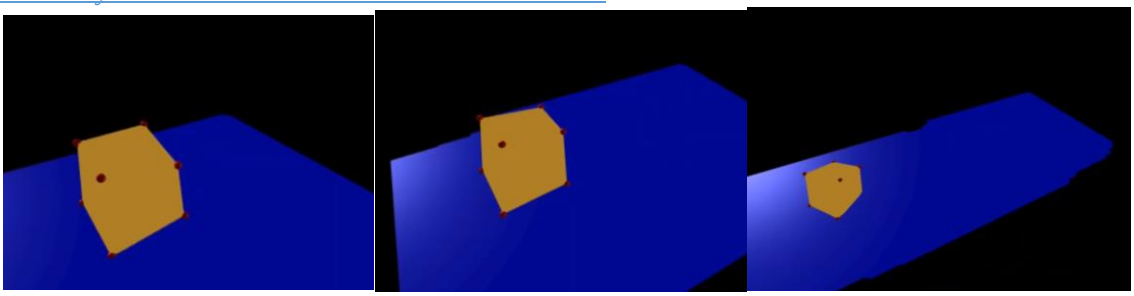
Fastest Robot:

https://www.youtube.com/watch?v=ffnTuuv-64s&t=3s



Fastest Robot w/ shaded faces:

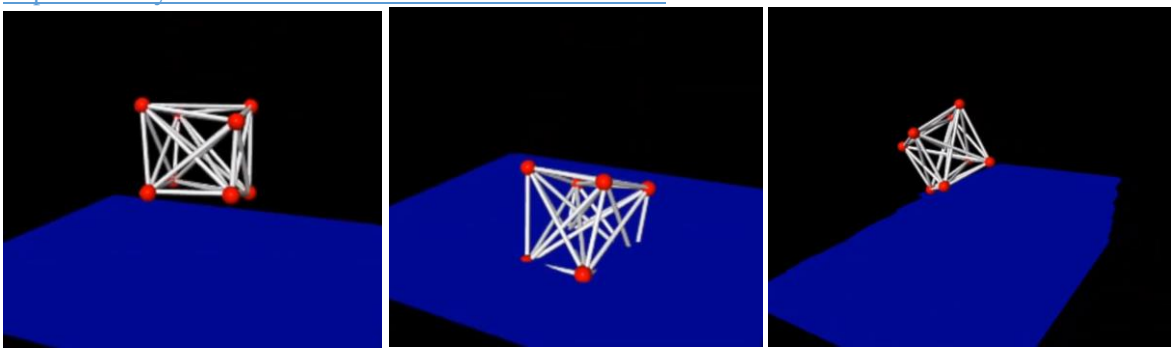
https://www.youtube.com/watch?v=ffnTuuv-64s&t=26s



Speed:1.4 m/s
Fitness: 2.83 m (when simulated for 2 seconds)

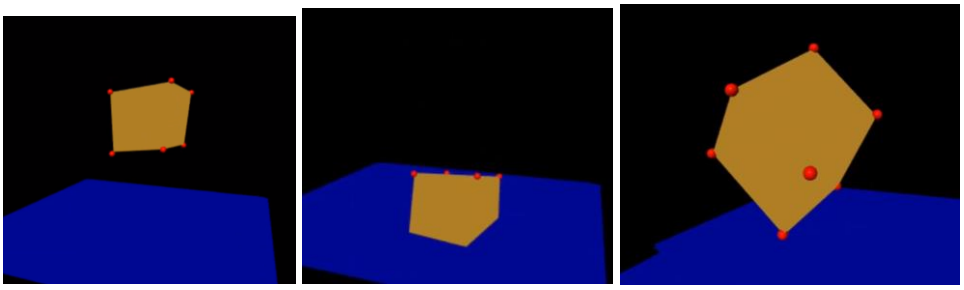
Fastest Robot Bouncing:

https://www.youtube.com/watch?v=ffnTuuv-64s&t=48s



Fastest Robot Bouncing w/ Shaded Faces:

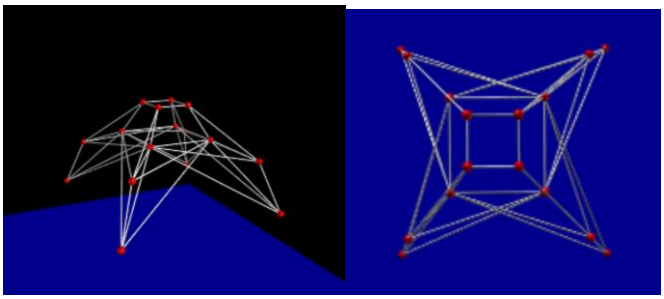
https://www.youtube.com/watch?v=ffnTuuv-64s&t=71s



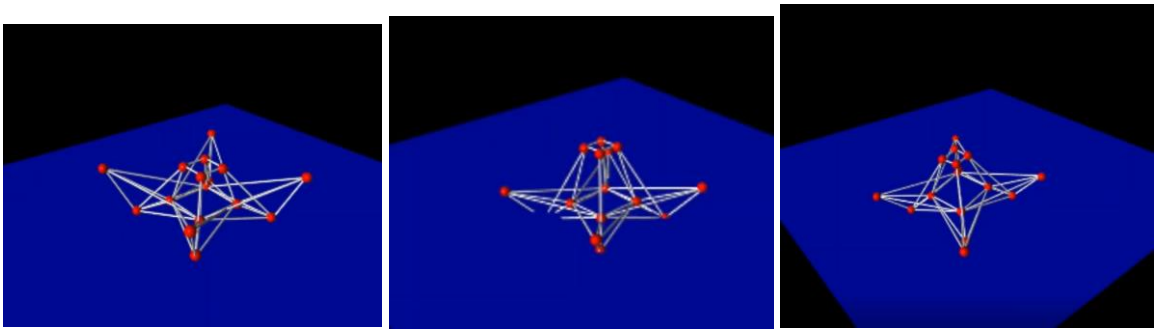
Speed: 2.35 m/s
Fitness: 4.70 m (when simulated for 2 seconds)

Innovative Robot

Design:



Evolution:
https://www.youtube.com/watch?v=ffnTuuv-64s&t=90s



Speed: 1.06 m/s
Fitness: 2.11 m (when simulated for 2 seconds)

METHODS

Actuation Methodology

A cube robot with 28 springs and 8 mass nodes was evolved. Each lengths between each mass, all 28, were evolved to actuate based on an initially random and then evolved sinusoidal value. In order to evolve the cube, a locomotion pattern was evolved between each mass. Rest length L_0 of each spring is defined as:

$$L_0 = a + b * \sin(wt + c)$$

Where a is the original rest length, and b is a random value between $[-0.2, 0.2]$ and c is a random value between $[-\pi, \pi]$.

Based on python's limitations, the spring evaluations per second is as follows:

Number of Spring Evaluations per second: 16632 springs/second.

Performance (Maximum Diameter per Cycle):
The velocity of our cube moving was 2.8m / 2s which would give us 1.4 m/s. The diameter for our cube was 1.73 meters. Therefore, by dividing 1.4m/s by 1.73m, 80.92%/s was obtained which demonstrated that our robot walk for a distance per second that was almost 81 percent of the diameter.

Evolutionary Process and Analysis

Below are the simulation and robot parameters used for the cube robot
Simulation Parameters:

Total Evolution Time	dT	Gravity	Spring Constant Floor
2 sec	.001 sec	$9.81\ m/s^2$	N/m

Robot Parameters:

# Masses	# Springs	Total Mass	Spring Constant
8	28	0.8 kg	N/m

For the evolutionary process, we wanted to evolve a robot that would move as far and fast as possible. Thus, fitness was defined as the magnitude of the initial origin and the final distance its center of mass has moved. A total population of 10 cubes were evolved due to the computational limitation of vpython (a simulation library). The mutation rate was set to 20%, and when achieved, would randomly switch the locomotion parameters b, c of two springs in the children of the population. Cross over was achieved by first selecting a set of 8 locomotion parameters of b, c, and k randomly from the 28 total possible locomotion ordered pairs. Note that 8 individual locomotion parameters were not randomly selected from the 28; rather, the 8 were locomotion parameters from 1 or two mass nodes. This crossover strategy and value was selected to increase linkage and preserve dominant genes.

However, there were still some limitations about the crossover and mutation mechanism. Since there were lack of new individual being added in to the population, the crossover and mutation occurred only between the initially generated parameters. The strategy still managed to grow and give better generations, but the diversity within the population would became limited once more generations were evaluated. Increasing the population size would improve this a little bit but a better way to improve it might be introducing new random parameters when mutation occurred so that new genes order can be explored that could potentially increase the performance of the generations and better fitness.

Innovative Robot Evolutionary Process and Analysis

Below are the parameters for the innovative robot:

Alternative Robot Parameters:

# Masses	# Springs	Total Mass	Spring Constant
16	120	0.8 kg	N/m

For the alternative robot, we wanted to create a spider with two joints per leg. To simulate the robot, we created two-layers of masses that initially lie on similar respective planes, as well as springs that connect them with the respective masses to control the degrees of freedom in which they can move. Unfortunately, due to the way our code is written, we are unable to compartmentalize individual locomotion in individual springs. Everything in our code operates as a reaction to a mass, that subsequently affects every other mass. This method was optimal for our cube robot, but less optimal for our alternative design. We'd optimally only want to actuate 32 new springs, but we are actually actuating 120 springs with our design. Theoretically, the evolutionary algorithm should still be able to effectively evolve our robot to move as far as possible, but it would do so in a much less efficient way than if we were to be able to control locomotion in individual springs. A ground up rework of our code is necessary if we were to continue with this spider design.

PERFORMANCE PLOTS

Learning Curve:

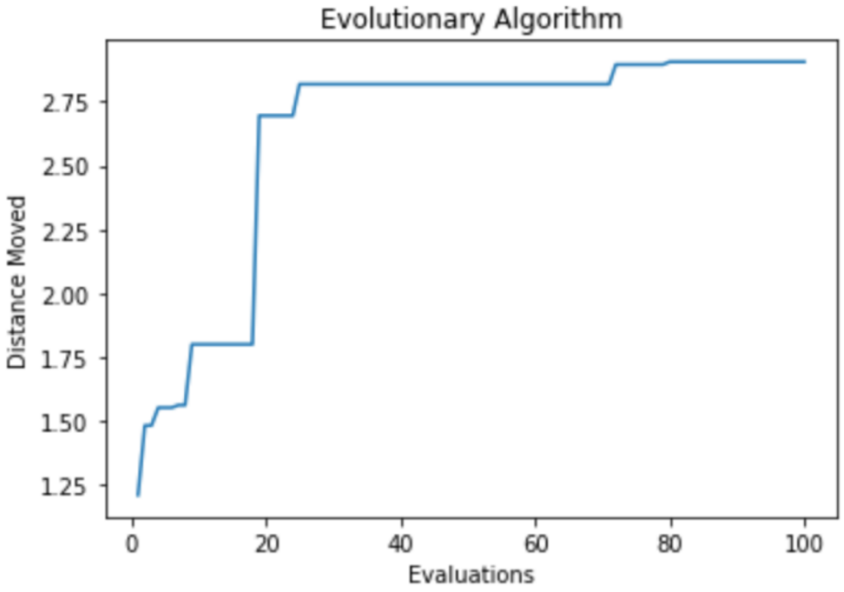


Figure 4: Learning curve for walking robot with 100 generations (2000 evaluations)

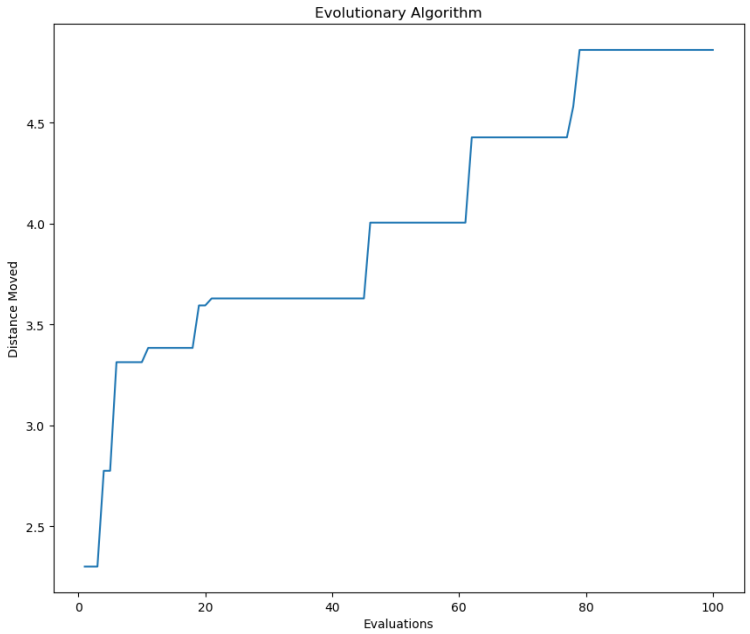


Figure 5: Learning curve for walking robot with bouncing with 100 generations (2000 evlatuions)

Dot Chart:

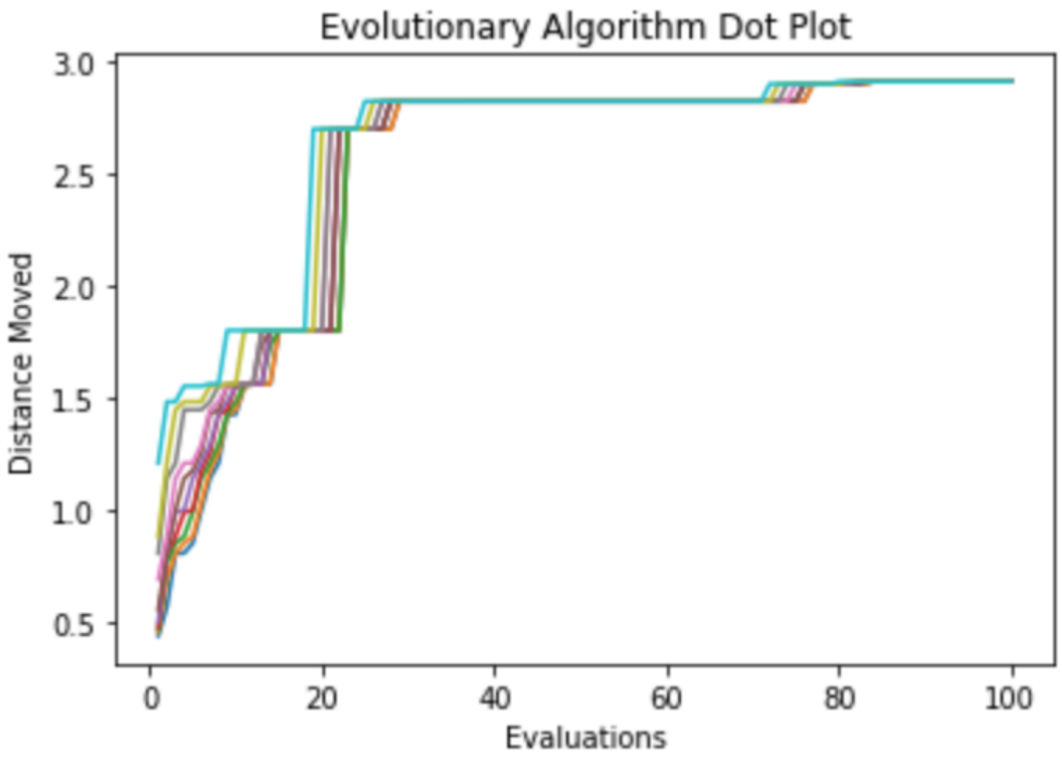


Figure 5: Dot chart for walking cubes

APPENDIX

Evolutionary Algorithm for Walking Cube Evolution

```
import vpython as vp
import itertools
import random
import numpy as np
from math import *
import matplotlib.pyplot as plt

# so first we create 10 parents randomly
# then what we gonna do is to crossover the 10 parents each pair of parents would generate two
children
# then we pick the best 10 out of the 20 which would become the next generation
scene = vp.canvas()
# vp.display(width=100, height=100)
floor = vp.box(pos=vp.vector(0, 0, 0), length=100, height=0.001, width=100, color=vp.color.blue)

v = 0
dt = 0.001
mass = 0.1
g = 9.81


# k1 = 1000


def getCOM(v):
    COM = (v[0].pos + v[1].pos + v[2].pos + v[3].pos + v[4].pos + v[5].pos + v[6].pos + v[7].pos)
/ 16
    return COM


def mutation(r):
    ran = random.randint(0, 9)
    if ran > mutation_rate:
        x = random.randint(0, 27)
        y = random.randint(0, 27)
        #         z = random.randint(0,27)
        r[x], r[y] = r[y], r[x]
        #         r[z][0] = np.random.uniform(-0.2,0.2)
        #         r[z][1] = np.random.uniform(-np.pi,np.pi)
        return r
    else:
        return r


def Crossover(m, n):
    M = []
    N = []
    for i in range(len(m)):
        M.append(m[i])
        N.append(n[i])
    rand = random.randint(0, 20)
    index = list(range(rand, rand + 8))
    exchangem = []
    exchangen = []
    for i in range(len(index)):
        exchangem.append(M[index[i]])
        exchangen.append(N[index[i]])
    for j in range(len(index)):
        M[index[j]] = exchangen[j]
        N[index[j]] = exchangem[j]
    return (M, N)


dis1 = 0
good_dis = []
best_dis = []
mutation_rate = 7
```

```
## Generate 10 parents
pa1 = []
for i in range(10):
    pa = []
    for i in range(28):
        p = []
        b = np.random.uniform(-0.2, 0.2)
        c = np.random.uniform(-np.pi, np.pi)
        k1 = np.random.uniform(1000, 6000)
        p.append(b)
        p.append(c)
        p.append(k1)
        pa.append(p)
    pa1.append(pa)
# print('PA1',len(pa1))

dots = []
for a in range(100):
    for i in np.arange(0, 10, 2):
        child1, child2 = Crossover(pa1[i], pa1[i + 1])
        pa1.append(child1)
        pa1.append(child2)

    for i in range(10, 20):
        pa1[i] = mutation(pa1[i])

    #     print('PA1after',pa1)

    total_dis = []
    for h in range(20):
        ballname = ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8']
        ballvectors = [vp.vector(0, 0, 0), vp.vector(0, 1, 0), vp.vector(0, 0, 1), vp.vector(1,
0, 0),
                       vp.vector(1, 1, 0), vp.vector(0, 1, 1),
                       vp.vector(1, 0, 1), vp.vector(1, 1, 1)]

        # for i in range(8):
        #     ballvectors[i] = ballvectors[i].rotate(angle=3.14/4, axis =vp.vector(1,1,1))
        OriginalCOM = (ballvectors[0] + ballvectors[1] + ballvectors[2] + ballvectors[3] +
ballvectors[4] + ballvectors[
            5] + ballvectors[6] + ballvectors[7]) / 16

        springvecs = []
        for i in range(len(ballname)):
            ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.1, color=vp.color.red,
velocity=vp.vector(0, 0, 0))
        for z in itertools.combinations(ballvectors, 2):
            springvecs.append(z)

        spring = ['s1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12',
's13', 's14', 's15', 's16',
                  's17', 's18', 's19', 's20', 's21', 's22', 's23', 's24', 's25', 's26', 's27',
's28']

        for i in range(28):
            position = springvecs[i][1] - springvecs[i][0]
            spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
radius=.03,
                                    color=vp.color.white)

        g_vector = vp.vector(0, 9.81, 0)
        for i in range(len(ballname)):
            ballname[i].velocity = vp.vector(0, 0, 0)

        F_c = vp.vector(0, 1000, 0)

        L0 = np.zeros((8, 8))
        for i in range(8):
            for j in range(8):
                if i == j:
                    L0[j][i] = 0
```

```
            else:
                position = ballname[j].pos - ballname[i].pos
                L0[j][i] = vp.mag(position)

    t = 0.001
    # t1 = 0
    # Time = []
    # E_H = []
    # E_S = []
    # E_K = []
    c = 1
    w = 10 * np.pi
    while True:
        #     vp.rate(200)
        L0rate = np.zeros((8, 8))
        L0rate[0][1] = L0[0][1] + pa1[h][0][0] * sin(w * t + pa1[h][0][1])
        L0rate[1][0] = L0[1][0] + pa1[h][0][0] * sin(w * t + pa1[h][0][1])
        L0rate[0][2] = L0[0][2] + pa1[h][1][0] * sin(w * t + pa1[h][1][1])
        L0rate[2][0] = L0[2][0] + pa1[h][1][0] * sin(w * t + pa1[h][1][1])
        L0rate[0][3] = L0[0][3] + pa1[h][2][0] * sin(w * t + pa1[h][2][1])
        L0rate[3][0] = L0[3][0] + pa1[h][2][0] * sin(w * t + pa1[h][2][1])
        L0rate[0][4] = L0[0][4] + pa1[h][3][0] * sin(w * t + pa1[h][3][1])
        L0rate[4][0] = L0[4][0] + pa1[h][3][0] * sin(w * t + pa1[h][3][1])
        L0rate[0][5] = L0[0][5] + pa1[h][4][0] * sin(w * t + pa1[h][4][1])
        L0rate[5][0] = L0[5][0] + pa1[h][4][0] * sin(w * t + pa1[h][4][1])
        L0rate[0][6] = L0[0][6] + pa1[h][5][0] * sin(w * t + pa1[h][5][1])
        L0rate[6][0] = L0[6][0] + pa1[h][5][0] * sin(w * t + pa1[h][5][1])
        L0rate[0][7] = L0[0][7] + pa1[h][6][0] * sin(w * t + pa1[h][6][1])
        L0rate[7][0] = L0[7][0] + pa1[h][6][0] * sin(w * t + pa1[h][6][1])
        L0rate[1][2] = L0[1][2] + pa1[h][7][0] * sin(w * t + pa1[h][7][1])
        L0rate[2][1] = L0[2][1] + pa1[h][7][0] * sin(w * t + pa1[h][7][1])
        L0rate[1][3] = L0[1][3] + pa1[h][8][0] * sin(w * t + pa1[h][8][1])
        L0rate[3][1] = L0[3][1] + pa1[h][8][0] * sin(w * t + pa1[h][8][1])
        L0rate[1][4] = L0[1][4] + pa1[h][9][0] * sin(w * t + pa1[h][9][1])
        L0rate[4][1] = L0[4][1] + pa1[h][9][0] * sin(w * t + pa1[h][9][1])
        L0rate[1][5] = L0[1][5] + pa1[h][10][0] * sin(w * t + pa1[h][10][1])
        L0rate[5][1] = L0[5][1] + pa1[h][10][0] * sin(w * t + pa1[h][10][1])
        L0rate[1][6] = L0[1][6] + pa1[h][11][0] * sin(w * t + pa1[h][11][1])
        L0rate[6][1] = L0[6][1] + pa1[h][11][0] * sin(w * t + pa1[h][11][1])
        L0rate[1][7] = L0[1][7] + pa1[h][12][0] * sin(w * t + pa1[h][12][1])
        L0rate[7][1] = L0[7][1] + pa1[h][12][0] * sin(w * t + pa1[h][12][1])
        L0rate[2][3] = L0[2][3] + pa1[h][13][0] * sin(w * t + pa1[h][13][1])
        L0rate[3][2] = L0[3][2] + pa1[h][13][0] * sin(w * t + pa1[h][13][1])
        L0rate[2][4] = L0[2][4] + pa1[h][14][0] * sin(w * t + pa1[h][14][1])
        L0rate[4][2] = L0[4][2] + pa1[h][14][0] * sin(w * t + pa1[h][14][1])
        L0rate[2][5] = L0[2][5] + pa1[h][15][0] * sin(w * t + pa1[h][15][1])
        L0rate[5][2] = L0[5][2] + pa1[h][15][0] * sin(w * t + pa1[h][15][1])
        L0rate[2][6] = L0[2][6] + pa1[h][16][0] * sin(w * t + pa1[h][16][1])
        L0rate[6][2] = L0[6][2] + pa1[h][16][0] * sin(w * t + pa1[h][16][1])
        L0rate[2][7] = L0[2][7] + pa1[h][17][0] * sin(w * t + pa1[h][17][1])
        L0rate[7][2] = L0[7][2] + pa1[h][17][0] * sin(w * t + pa1[h][17][1])
        L0rate[3][4] = L0[3][4] + pa1[h][18][0] * sin(w * t + pa1[h][18][1])
        L0rate[4][3] = L0[4][3] + pa1[h][18][0] * sin(w * t + pa1[h][18][1])
        L0rate[3][5] = L0[3][5] + pa1[h][19][0] * sin(w * t + pa1[h][19][1])
        L0rate[5][3] = L0[5][3] + pa1[h][19][0] * sin(w * t + pa1[h][19][1])
        L0rate[3][6] = L0[3][6] + pa1[h][20][0] * sin(w * t + pa1[h][20][1])
        L0rate[6][3] = L0[6][3] + pa1[h][20][0] * sin(w * t + pa1[h][20][1])
        L0rate[3][7] = L0[3][7] + pa1[h][21][0] * sin(w * t + pa1[h][21][1])
        L0rate[7][3] = L0[7][3] + pa1[h][21][0] * sin(w * t + pa1[h][21][1])
        L0rate[4][5] = L0[4][5] + pa1[h][22][0] * sin(w * t + pa1[h][22][1])
        L0rate[5][4] = L0[5][4] + pa1[h][22][0] * sin(w * t + pa1[h][22][1])
        L0rate[4][6] = L0[4][6] + pa1[h][23][0] * sin(w * t + pa1[h][23][1])
        L0rate[6][4] = L0[6][4] + pa1[h][23][0] * sin(w * t + pa1[h][23][1])
        L0rate[4][7] = L0[4][7] + pa1[h][24][0] * sin(w * t + pa1[h][24][1])
        L0rate[7][4] = L0[7][4] + pa1[h][24][0] * sin(w * t + pa1[h][24][1])
        L0rate[5][6] = L0[5][6] + pa1[h][25][0] * sin(w * t + pa1[h][25][1])
        L0rate[6][5] = L0[6][5] + pa1[h][25][0] * sin(w * t + pa1[h][25][1])
        L0rate[5][7] = L0[5][7] + pa1[h][26][0] * sin(w * t + pa1[h][26][1])
        L0rate[7][5] = L0[7][5] + pa1[h][26][0] * sin(w * t + pa1[h][26][1])
        L0rate[6][7] = L0[6][7] + pa1[h][27][0] * sin(w * t + pa1[h][27][1])
        L0rate[7][6] = L0[7][6] + pa1[h][27][0] * sin(w * t + pa1[h][27][1])
```

```
        ks = np.zeros((8, 8))
        ks[0][1] = pa1[h][0][2]
        ks[1][0] = pa1[h][0][2]
        ks[0][2] = pa1[h][1][2]
        ks[2][0] = pa1[h][1][2]
        ks[0][3] = pa1[h][2][2]
        ks[3][0] = pa1[h][2][2]
        ks[0][4] = pa1[h][3][2]
        ks[4][0] = pa1[h][3][2]
        ks[0][5] = pa1[h][4][2]
        ks[5][0] = pa1[h][4][2]
        ks[0][6] = pa1[h][5][2]
        ks[6][0] = pa1[h][5][2]
        ks[0][7] = pa1[h][6][2]
        ks[7][0] = pa1[h][6][2]
        ks[1][2] = pa1[h][7][2]
        ks[2][1] = pa1[h][7][2]
        ks[1][3] = pa1[h][8][2]
        ks[3][1] = pa1[h][8][2]
        ks[1][4] = pa1[h][9][2]
        ks[4][1] = pa1[h][9][2]
        ks[1][5] = pa1[h][10][2]
        ks[5][1] = pa1[h][10][2]
        ks[1][6] = pa1[h][11][2]
        ks[6][1] = pa1[h][11][2]
        ks[1][7] = pa1[h][12][2]
        ks[7][1] = pa1[h][12][2]
        ks[2][3] = pa1[h][13][2]
        ks[3][2] = pa1[h][13][2]
        ks[2][4] = pa1[h][14][2]
        ks[4][2] = pa1[h][14][2]
        ks[2][5] = pa1[h][15][2]
        ks[5][2] = pa1[h][15][2]
        ks[2][6] = pa1[h][16][2]
        ks[6][2] = pa1[h][16][2]
        ks[2][7] = pa1[h][17][2]
        ks[7][2] = pa1[h][17][2]
        ks[3][4] = pa1[h][18][2]
        ks[4][3] = pa1[h][18][2]
        ks[3][5] = pa1[h][19][2]
        ks[5][3] = pa1[h][19][2]
        ks[3][6] = pa1[h][20][2]
        ks[6][3] = pa1[h][20][2]
        ks[3][7] = pa1[h][21][2]
        ks[7][3] = pa1[h][21][2]
        ks[4][5] = pa1[h][22][2]
        ks[5][4] = pa1[h][22][2]
        ks[4][6] = pa1[h][23][2]
        ks[6][4] = pa1[h][23][2]
        ks[4][7] = pa1[h][24][2]
        ks[7][4] = pa1[h][24][2]
        ks[5][6] = pa1[h][25][2]
        ks[6][5] = pa1[h][25][2]
        ks[5][7] = pa1[h][26][2]
        ks[7][5] = pa1[h][26][2]
        ks[6][7] = pa1[h][27][2]
        ks[7][6] = pa1[h][27][2]
        #     print(L0rate)
        t += 0.001
        for i in range(8):
            ballvectors[i] = ballname[i].pos
        springvecs = []
        for z in itertools.combinations(ballvectors, 2):
            springvecs.append(z)
        for i in range(28):
            position = springvecs[i][1] - springvecs[i][0]
            spring[i].pos = springvecs[i][0]
            spring[i].axis = position
            spring[i].length = vp.mag(position)
```

```
            dampening = 0.99

            F_mat = np.zeros((8, 8))
            F_vec = []
            F_v = []
            a = np.array(np.zeros((8, 8)))

            for i in range(8):
                for k in range(8):
                    if k == i:
                        L = 0
                        F_mat[i][k] = 0
                        F_vec.append(vp.vector(0, 0, 0))
                    else:
                        L = vp.mag(ballname[k].pos - ballname[i].pos) - L0rate[k][i]
                        #                 E_s.append(1/2*k_sp*L**2)
                        F_mat[i][k] = L * ks[k][i]
                        pf0 = ballname[k].pos - ballname[i].pos
                        # a[i,k] = vp.norm(pf0)*L*k_sp
                        F_vec.append(vp.norm(pf0) * L * ks[k][i])
                        #     E_S.append(sum(E_s)/2)
            a = np.array(F_vec).reshape(8, 8)
            F = a.sum(axis=0)

            for i in range(8):
                F[i] = F[i] + g_vector * mass
                if ballname[i].pos.y < floor.pos.y:
                    F_N = ((floor.pos.y - ballname[i].pos.y) ** 2) * 800
                    F[i].y = F[i].y - F_N
                    mu = 1
                    F_st = mu * F_N
                    F_horiz = (F[i].x ** 2 + F[i].z ** 2) ** 0.5
                    v_xz = (ballname[i].velocity.x ** 2 + ballname[i].velocity.z ** 2) ** 0.5
                    vx = ballname[i].velocity.x / v_xz
                    vz = ballname[i].velocity.z / v_xz
                    if F_st < F_horiz:
                        F[i].x += F_horiz * vx - F_N * vx
                        F[i].z += F_horiz * vz - F_N * vz
                    else:
                        F[i].x = F_horiz * vx
                        F[i].z = F_horiz * vz
                        ballname[i].velocity.x = 0
                        ballname[i].velocity.z = 0

            for i in range(8):
                ballname[i].velocity -= (F[i] / mass * dt) * dampening
                ballname[i].pos += ballname[i].velocity * dt

            c += 1
            if c == 2000:
                break

        # Calculating COM
        COM = getCOM(ballname)
        dvec = COM - OriginalCOM
        dis = sqrt(dvec.x ** 2 + dvec.z ** 2)
        #         print(dis)
        total_dis.append(dis)
    dis_index = np.argsort(total_dis)
    sorted_dis = []
    sorted_pa1 = []
    for i in range(20):
        sorted_dis.append(total_dis[dis_index[i]])
        sorted_pa1.append(pa1[dis_index[i]])
    good_dis = sorted_dis[-10:]
    dots.append(good_dis)
    print('GOODDIS', good_dis[-1])
    best_dis.append(good_dis[-1])
    pa1 = sorted_pa1[-10:]
    print('PA1END', len(pa1))
```

```
#         if dis > dis1:
#             dis1 = dis
#         good_dis.append(dis1)

print(best_dis)
print(pa1[-1])
evals = list(range(1, 101))

plt.plot(evals, best_dis)
plt.xlabel('Evaluations')
plt.ylabel('Distance Moved')
plt.title('Evolutionary Algorithm')
plt.show()

plt.plot(evals, dots)
plt.xlabel('Evaluations')
plt.ylabel('Distance Moved')
plt.title('Evolutionary Algorithm Dot Plot')
plt.show()
```

Evolutionary Algorithm for Walking Cube SIMULATED

```
import vpython as vp
import itertools
import numpy as np
from math import *
import matplotlib.pyplot as plt

scene = vp.canvas()
# vp.display(width=100, height=100)
floor = vp.box(pos=vp.vector(0, 0, 0), length=5, height=0.001, width=5, color=vp.color.blue)

ballname = ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8']
ballvectors = [vp.vector(0, 0, 0), vp.vector(0, 1, 0), vp.vector(0, 0, 1), vp.vector(1, 0, 0),
vp.vector(1, 1, 0),
               vp.vector(0, 1, 1),
               vp.vector(1, 0, 1), vp.vector(1, 1, 1)]
# for i in range(8):
#     ballvectors[i] = ballvectors[i].rotate(angle=3.14/4, axis =vp.vector(1,1,1))
OriginalCOM = (ballvectors[0] + ballvectors[1] + ballvectors[2] + ballvectors[3] + ballvectors[4]
+ ballvectors[5] +
               ballvectors[6] + ballvectors[7]) / 8

triangles = []
for z in itertools.combinations(ballvectors, 3):
    triangles.append(z)

T = list(range(56))
for i in range(len(triangles)):
    T[i] = vp.triangle(v0= vp.vertex(pos=triangles[i][0]), v1=vp.vertex(pos=triangles[i][1] ),
v2=vp.vertex(pos=triangles[i][2]), texture = "https://i.imgur.com/eQueRtf.jpg")



springvecs = []
for i in range(len(ballname)):
    ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.05, color=vp.color.red,
velocity=vp.vector(0, 0, 0))
for z in itertools.combinations(ballvectors, 2):
    springvecs.append(z)

spring = ['s1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13',
's14', 's15', 's16', 's17',
          's18', 's19', 's20', 's21', 's22', 's23', 's24', 's25', 's26', 's27', 's28']

# for i in range(28):
#     position = springvecs[i][1] - springvecs[i][0]
#     spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
```

```
radius=.03,
#                             color=vp.color.white)

v = 0
dt = 0.001
mass = 0.1
g = 9.81
# k_sp = []
# k1 = 1000

g_vector = vp.vector(0, 9.81, 0)
for i in range(len(ballname)):
    ballname[i].velocity = vp.vector(0, 0, 0)

F_c = vp.vector(0, 1000, 0)


def getCOM(v):
    COM = (v[0].pos + v[1].pos + v[2].pos + v[3].pos + v[4].pos + v[5].pos + v[6].pos + v[7].pos)
/ 8
    return COM


# pa = []
# for i in range(28):
#     p = []
#     b = np.random.uniform(-1,1)
#     c = np.random.uniform(-np.pi,np.pi)
#     p.append(b)
#     p.append(c)
#     pa.append(p)
# print(pa)
pa = [[-0.0999684941118836, -2.436063476356824, 2296.7581897299874],
      [-0.07967391969199222, -0.08634880798548794, 2361.5826862217555],
      [0.09067974114260546, -0.21159318157779383, 4755.324973113422],
      [0.18380205176538172, -2.0565408755726033, 4409.3898116994515],
      [-0.09146751801551414, -1.1488912489420897, 1828.5235061201834],
      [0.12405268447947199, -2.813723875627788, 4433.034537634187],
      [0.11126570634985072, 2.9781789480327108, 1536.2099136022057],
      [0.17140895971224107, 3.0375520686516975, 4274.469299579441],
      [0.03410203251472729, 0.3029379290102514, 1816.8271542650991],
      [-0.042351202453747266, 1.3271962303422749, 4865.486843841038],
      [-0.12401891611004096, -1.455914399642346, 2241.1865101427006],
      [0.16860887015307674, 2.3914072461699805, 2474.754155099656],
      [-0.11773307676195617, -0.3926900646112008, 1067.4545888208283],
      [0.1889606677054101, 1.5196174154857083, 5192.421195053015],
      [0.035816105198113485, -2.1041344746790127, 4666.689845214234],
      [-0.010549926384152558, -1.3311801212100072, 1236.6327877338986],
      [-0.10281383568907182, -0.6452025562267107, 1809.5808026975737],
      [-0.13964559177630298, 1.0742302809635627, 4488.763201981452],
      [0.13159014364690885, -1.6264225015119274, 1043.3984921278495],
      [-0.15788353962765111, 2.445306398612243, 4962.665844610705],
      [-0.10355923252096791, -2.0136708324532977, 3816.315590216662],
      [0.10326981592129303, -0.9756094020324113, 2285.446390155172],
      [-0.051793601473909684, -2.0312235905388243, 2908.6840001433184],
      [-0.0443805533814709, -1.7160858876498892, 4580.86053237835],
      [0.14274328807395192, 0.5766512368889116, 4898.871085132674],
      [-0.1280887144442297, 0.32902508167789923, 5017.558774143425],
      [-0.15039361389519904, -2.2832635644031134, 4323.8339489367545],
      [0.10632846454515804, 1.6343248793719205, 3613.8574144730146]]
L0 = np.zeros((8, 8))
for i in range(8):
    for j in range(8):
        if i == j:
            L0[j][i] = 0
        else:
            position = ballname[j].pos - ballname[i].pos
            L0[j][i] = vp.mag(position)

# print(L0)
# now we have L0 matrix with 56 elements
```

```
L0rate = np.zeros((8, 8))

t = 0.001
# t1 = 0
# Time = []
# E_H = []
# E_S = []
# E_K = []
c = 1
w = 10 * np.pi
eta = 1
while True:
    vp.rate(200)
    floor = vp.box(pos=vp.vector(ballvectors[5].x, 0, ballvectors[1].z), length=5, height=0.001,
width=5, color=vp.color.blue)
    scene.forward = vp.vector(-1,-1,1.5)
    #scene.center.x = ballvectors[1].x
    #scene.center.z = -1*ballvectors[1].z
    scene.center.y = ballvectors[1].y
    scene.center.z = ballvectors[1].z

    L0rate[0][1] = L0[0][1] + pa[0][0] * sin(w * t + pa[0][1]) * eta
    L0rate[1][0] = L0[1][0] + pa[0][0] * sin(w * t + pa[0][1]) * eta
    L0rate[0][2] = L0[0][2] + pa[1][0] * sin(w * t + pa[1][1]) * eta
    L0rate[2][0] = L0[2][0] + pa[1][0] * sin(w * t + pa[1][1]) * eta
    L0rate[0][3] = L0[0][3] + pa[2][0] * sin(w * t + pa[2][1]) * eta
    L0rate[3][0] = L0[3][0] + pa[2][0] * sin(w * t + pa[2][1]) * eta
    L0rate[0][4] = L0[0][4] + pa[3][0] * sin(w * t + pa[3][1]) * eta
    L0rate[4][0] = L0[4][0] + pa[3][0] * sin(w * t + pa[3][1]) * eta
    L0rate[0][5] = L0[0][5] + pa[4][0] * sin(w * t + pa[4][1]) * eta
    L0rate[5][0] = L0[5][0] + pa[4][0] * sin(w * t + pa[4][1]) * eta
    L0rate[0][6] = L0[0][6] + pa[5][0] * sin(w * t + pa[5][1]) * eta
    L0rate[6][0] = L0[6][0] + pa[5][0] * sin(w * t + pa[5][1]) * eta
    L0rate[0][7] = L0[0][7] + pa[6][0] * sin(w * t + pa[6][1]) * eta
    L0rate[7][0] = L0[7][0] + pa[6][0] * sin(w * t + pa[6][1]) * eta
    L0rate[1][2] = L0[1][2] + pa[7][0] * sin(w * t + pa[7][1]) * eta
    L0rate[2][1] = L0[2][1] + pa[7][0] * sin(w * t + pa[7][1]) * eta
    L0rate[1][3] = L0[1][3] + pa[8][0] * sin(w * t + pa[8][1]) * eta
    L0rate[3][1] = L0[3][1] + pa[8][0] * sin(w * t + pa[8][1]) * eta
    L0rate[1][4] = L0[1][4] + pa[9][0] * sin(w * t + pa[9][1]) * eta
    L0rate[4][1] = L0[4][1] + pa[9][0] * sin(w * t + pa[9][1]) * eta
    L0rate[1][5] = L0[1][5] + pa[10][0] * sin(w * t + pa[10][1]) * eta
    L0rate[5][1] = L0[5][1] + pa[10][0] * sin(w * t + pa[10][1]) * eta
    L0rate[1][6] = L0[1][6] + pa[11][0] * sin(w * t + pa[11][1]) * eta
    L0rate[6][1] = L0[6][1] + pa[11][0] * sin(w * t + pa[11][1]) * eta
    L0rate[1][7] = L0[1][7] + pa[12][0] * sin(w * t + pa[12][1]) * eta
    L0rate[7][1] = L0[7][1] + pa[12][0] * sin(w * t + pa[12][1]) * eta
    L0rate[2][3] = L0[2][3] + pa[13][0] * sin(w * t + pa[13][1]) * eta
    L0rate[3][2] = L0[3][2] + pa[13][0] * sin(w * t + pa[13][1]) * eta
    L0rate[2][4] = L0[2][4] + pa[14][0] * sin(w * t + pa[14][1]) * eta
    L0rate[4][2] = L0[4][2] + pa[14][0] * sin(w * t + pa[14][1]) * eta
    L0rate[2][5] = L0[2][5] + pa[15][0] * sin(w * t + pa[15][1]) * eta
    L0rate[5][2] = L0[5][2] + pa[15][0] * sin(w * t + pa[15][1]) * eta
    L0rate[2][6] = L0[2][6] + pa[16][0] * sin(w * t + pa[16][1]) * eta
    L0rate[6][2] = L0[6][2] + pa[16][0] * sin(w * t + pa[16][1]) * eta
    L0rate[2][7] = L0[2][7] + pa[17][0] * sin(w * t + pa[17][1]) * eta
    L0rate[7][2] = L0[7][2] + pa[17][0] * sin(w * t + pa[17][1]) * eta
    L0rate[3][4] = L0[3][4] + pa[18][0] * sin(w * t + pa[18][1]) * eta
    L0rate[4][3] = L0[4][3] + pa[18][0] * sin(w * t + pa[18][1]) * eta
    L0rate[3][5] = L0[3][5] + pa[19][0] * sin(w * t + pa[19][1]) * eta
    L0rate[5][3] = L0[5][3] + pa[19][0] * sin(w * t + pa[19][1]) * eta
    L0rate[3][6] = L0[3][6] + pa[20][0] * sin(w * t + pa[20][1]) * eta
    L0rate[6][3] = L0[6][3] + pa[20][0] * sin(w * t + pa[20][1]) * eta
    L0rate[3][7] = L0[3][7] + pa[21][0] * sin(w * t + pa[21][1]) * eta
    L0rate[7][3] = L0[7][3] + pa[21][0] * sin(w * t + pa[21][1]) * eta
    L0rate[4][5] = L0[4][5] + pa[22][0] * sin(w * t + pa[22][1]) * eta
    L0rate[5][4] = L0[5][4] + pa[22][0] * sin(w * t + pa[22][1]) * eta
    L0rate[4][6] = L0[4][6] + pa[23][0] * sin(w * t + pa[23][1]) * eta
    L0rate[6][4] = L0[6][4] + pa[23][0] * sin(w * t + pa[23][1]) * eta
    L0rate[4][7] = L0[4][7] + pa[24][0] * sin(w * t + pa[24][1]) * eta
    L0rate[7][4] = L0[7][4] + pa[24][0] * sin(w * t + pa[24][1]) * eta
```

```
    L0rate[5][6] = L0[5][6] + pa[25][0] * sin(w * t + pa[25][1]) * eta
    L0rate[6][5] = L0[6][5] + pa[25][0] * sin(w * t + pa[25][1]) * eta
    L0rate[5][7] = L0[5][7] + pa[26][0] * sin(w * t + pa[26][1]) * eta
    L0rate[7][5] = L0[7][5] + pa[26][0] * sin(w * t + pa[26][1]) * eta
    L0rate[6][7] = L0[6][7] + pa[27][0] * sin(w * t + pa[27][1]) * eta
    L0rate[7][6] = L0[7][6] + pa[27][0] * sin(w * t + pa[27][1]) * eta

    ks = np.zeros((8, 8))
    ks[0][1] = pa[0][2]
    ks[1][0] = pa[0][2]
    ks[0][2] = pa[1][2]
    ks[2][0] = pa[1][2]
    ks[0][3] = pa[2][2]
    ks[3][0] = pa[2][2]
    ks[0][4] = pa[3][2]
    ks[4][0] = pa[3][2]
    ks[0][5] = pa[4][2]
    ks[5][0] = pa[4][2]
    ks[0][6] = pa[5][2]
    ks[6][0] = pa[5][2]
    ks[0][7] = pa[6][2]
    ks[7][0] = pa[6][2]
    ks[1][2] = pa[7][2]
    ks[2][1] = pa[7][2]
    ks[1][3] = pa[8][2]
    ks[3][1] = pa[8][2]
    ks[1][4] = pa[9][2]
    ks[4][1] = pa[9][2]
    ks[1][5] = pa[10][2]
    ks[5][1] = pa[10][2]
    ks[1][6] = pa[11][2]
    ks[6][1] = pa[11][2]
    ks[1][7] = pa[12][2]
    ks[7][1] = pa[12][2]
    ks[2][3] = pa[13][2]
    ks[3][2] = pa[13][2]
    ks[2][4] = pa[14][2]
    ks[4][2] = pa[14][2]
    ks[2][5] = pa[15][2]
    ks[5][2] = pa[15][2]
    ks[2][6] = pa[16][2]
    ks[6][2] = pa[16][2]
    ks[2][7] = pa[17][2]
    ks[7][2] = pa[17][2]
    ks[3][4] = pa[18][2]
    ks[4][3] = pa[18][2]
    ks[3][5] = pa[19][2]
    ks[5][3] = pa[19][2]
    ks[3][6] = pa[20][2]
    ks[6][3] = pa[20][2]
    ks[3][7] = pa[21][2]
    ks[7][3] = pa[21][2]
    ks[4][5] = pa[22][2]
    ks[5][4] = pa[22][2]
    ks[4][6] = pa[23][2]
    ks[6][4] = pa[23][2]
    ks[4][7] = pa[24][2]
    ks[7][4] = pa[24][2]
    ks[5][6] = pa[25][2]
    ks[6][5] = pa[25][2]
    ks[5][7] = pa[26][2]
    ks[7][5] = pa[26][2]
    ks[6][7] = pa[27][2]
    ks[7][6] = pa[27][2]
    #     print(L0rate)
    t += 0.001
    for i in range(8):
        ballvectors[i] = ballname[i].pos
    springvecs = []
    for z in itertools.combinations(ballvectors, 2):
        springvecs.append(z)
```

```
    # for i in range(28):
    #     position = springvecs[i][1] - springvecs[i][0]
    #     spring[i].pos = springvecs[i][0]
    #     spring[i].axis = position
    #     spring[i].length = vp.mag(position)


    triangles = []
    for z in itertools.combinations(ballvectors, 3):
        triangles.append(z)
    for i in range(len(triangles)):
        T[i].v0.pos = triangles[i][0]
        T[i].v1.pos = triangles[i][1]
        T[i].v2.pos = triangles[i][2]


    dampening = 0.99

    F_mat = np.zeros((8, 8))
    F_vec = []
    F_v = []
    a = np.array(np.zeros((8, 8)))
    #     E_s = []
    for i in range(8):
        for k in range(8):
            if k == i:
                L = 0
                F_mat[i][k] = 0
                F_vec.append(vp.vector(0, 0, 0))
            else:
                L = vp.mag(ballname[k].pos - ballname[i].pos) - L0rate[k][i]
                #                 E_s.append(1/2*k_sp*L**2)
                F_mat[i][k] = L * ks[k][i]
                pf0 = ballname[k].pos - ballname[i].pos
                # a[i,k] = vp.norm(pf0)*L*k_sp
                F_vec.append(vp.norm(pf0) * L * ks[k][i])
            #     E_S.append(sum(E_s)/2)
    a = np.array(F_vec).reshape(8, 8)
    F = a.sum(axis=0)

    for i in range(8):
        F[i] = F[i] + g_vector * mass
        if ballname[i].pos.y < floor.pos.y:
            F_N = ((floor.pos.y - ballname[i].pos.y) ** 2) * 800
            F[i].y = F[i].y - F_N
            mu = 1
            F_st = mu * F_N
            F_horiz = (F[i].x ** 2 + F[i].z** 2) ** 0.5
            v_xz = (ballname[i].velocity.x ** 2 + ballname[i].velocity.z ** 2) ** 0.5
            vx = ballname[i].velocity.x / v_xz
            vz = ballname[i].velocity.z / v_xz
            if F_st < F_horiz:
                F[i].x += F_horiz * vx - F_N * vx
                F[i].z += F_horiz * vz - F_N * vz
            else:
                F[i].x = F_horiz * vx
                F[i].z = F_horiz * vz
                ballname[i].velocity.x = 0
                ballname[i].velocity.z = 0

    for i in range(8):
        ballname[i].velocity -= (F[i] / mass * dt) * dampening
        ballname[i].pos += ballname[i].velocity * dt

    #     for i in range(8):
    #         F[i] = F[i] + mass*g_vector
    #         if ballname[i].pos.y <= floor.pos.y:
    #             F[i].y = F[i].y - dampening*((floor.pos.y-ballname[i].pos.y)**2)*10000
    # #     E_k = []
    # #     E_h = []
    #     for i in range(8):
```

```
    #         ballname[i].velocity -= F[i]/mass*dt
    # #           E_k.append(1/2*mass*(vp.mag(ballname[i].velocity))**2)
    #         ballname[i].pos += ballname[i].velocity*dt
    # #           E_h.append(mass*9.81*(ballname[i].pos.y))
    # #     E_K.append(sum(E_k))
    # #     E_H.append(sum(E_h))
    c += 1
    if c == 6000:
        break

# Calculating COM
COM = getCOM(ballname)
dvec = COM - OriginalCOM
dis = sqrt(dvec.x ** 2 + dvec.z ** 2)
print(dis)
#     t1 += 0.004
#     Time.append(t1)
#     if t > 60:
#         break

# E_total = []
# E_potential = []
# for i in range(len(E_H)):
#     E_total.append(E_K[i]+E_H[i]+E_S[i])
# for i in range(len(E_H)):
#     E_potential.append(E_H[i]+E_S[i])

# plt.plot(Time, E_total)
# plt.xlabel('Time(s)')
# plt.ylabel('Total Energy(J)')
# plt.title('Breathing Total Energy')
# plt.show()

# plt.plot(Time, E_potential)
# plt.xlabel('Time(s)')
# plt.ylabel('Potential Energy(J)')
# plt.title('Breathing Cube Potential Energy')
# plt.show()

# plt.plot(Time, E_K)
# plt.xlabel('Time(s)')
# plt.ylabel('Kinetic Energy(J)')
# plt.title('Breathing Cube Kinetic Energy')
# plt.show()

#         if ballname[i].pos.y < floor.pos.y:
#             ballname[i].velocity = dampening*-1*(ballname[i].velocity)+ F_c*(floor.pos.y -
ballname[i].pos.y)*(floor.pos.y - ballname[i].pos.y)
```

Innovative Robot Evolution

```
##INNOVATIVE SPIDER

import vpython as vp
import itertools
import numpy as np
from math import *
import random
import matplotlib.pyplot as plt

scene = vp.canvas()  # title = 'Box', width = 600, height = 400, center = vp.vector(0,0,0))

floor = vp.box(pos=vp.vector(0, -1.3, 0), length=10, height=0.01, width=10, color=vp.color.blue)


def getCOM(v):
    COM = (v[0].pos + v[1].pos + v[2].pos + v[3].pos + v[4].pos + v[5].pos + v[6].pos + v[7].pos
+ v[8].pos + v[9].pos +
           v[10].pos + v[11].pos + v[12].pos + v[13].pos + v[14].pos + v[15].pos) / 16
    return COM
```

```
def mutation(r):
    ran = random.randint(0, 9)
    if ran > mutation_rate:
        x = random.randint(0, 25)
        y = random.randint(0, 25)
        r[x], r[y] = r[y], r[x]
        return r
    else:
        return r


def Crossover(m, n):
    M = []
    N = []
    for i in range(len(m)):
        M.append(m[i])
        N.append(n[i])
    rand = random.randint(0, 20)
    index = list(range(rand, rand + 6))
    exchangem = []
    exchangen = []
    for i in range(len(index)):
        exchangem.append(M[index[i]])
        exchangen.append(N[index[i]])
    for j in range(len(index)):
        M[index[j]] = exchangen[j]
        N[index[j]] = exchangem[j]
    return (M, N)


dis1 = 0
good_dis = []
best_dis = []
mutation_rate = 7

v = 0
dt = 0.001
mass = 0.1
g = 9.81
k_sp = 2000
g_vector = vp.vector(0, 9.81, 0)

pa1 = []
for i in range(10):
    pa = []
    for j in range(26):
        p = []
        b = np.random.uniform(-0.4, 0.4)
        c = np.random.uniform(-np.pi, np.pi)
        p.append(b)
        p.append(c)
        pa.append(p)

    pa1.append(pa)

dots = []

for a in range(2):
    print(a, "gen")
    for i in np.arange(0, 10, 2):
        child1, child2 = Crossover(pa1[i], pa1[i + 1])

        pa1.append(child1)
        pa1.append(child2)

    for i in range(10, 20):
        pa1[i] = mutation(pa1[i])

    total_dis = []
```

```
    take = []
    for h in range(20):
        print(h)
        ballname = ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8']
        ballvectors = [vp.vector(0, 0, 0), vp.vector(0, 0, 1), vp.vector(1, 0, 0), vp.vector(1,
0, 1),
                       vp.vector(0.25, 0.5, 0.25), vp.vector(0.25, 0.5, 0.75), vp.vector(0.75,
0.5, 0.25),
                       vp.vector(0.75, 0.5, 0.75)]

        for i in range(len(ballname)):
            ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.1, color=vp.color.red,
f_k=vp.vector(0, 0, 0))

        springvecs = [[ballvectors[0], ballvectors[4]], [ballvectors[0], ballvectors[1]],
                      [ballvectors[0], ballvectors[2]],
                      [ballvectors[1], ballvectors[5]], [ballvectors[1], ballvectors[3]],
                      [ballvectors[2], ballvectors[6]],
                      [ballvectors[3], ballvectors[7]], [ballvectors[3], ballvectors[2]],
                      [ballvectors[4], ballvectors[5]],
                      [ballvectors[4], ballvectors[6]], [ballvectors[7], ballvectors[5]],
                      [ballvectors[7], ballvectors[6]]]

        spring = ['s1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11',
                  's12']  # , 's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20','s21',
's22', 's23', 's24', 's25', 's26', 's27', 's28']

        for i in range(len(springvecs)):
            position = springvecs[i][1] - springvecs[i][0]
            spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
radius=.02,
                                    color=vp.color.white)

        ballname2 = ['b9', 'b10', 'b11', 'b12']
        ballvectors2 = [vp.vector(-0.5, -0.25, -0.5), vp.vector(-0.5, -0.25, 1.5), vp.vector(1.5,
-0.25, 1.5),
                        vp.vector(1.5, -0.25, -0.5)]
        for i in range(len(ballname2)):
            ballname2[i] = vp.sphere(pos=ballvectors2[i], radius=0.1, color=vp.color.red,
f_k=vp.vector(0, 0, 0))
        springvecs2 = []
        for i in range(4):
            corner1 = ballvectors2[i]
            if i == 0:
                springvecs2.append(
                    [[corner1, ballvectors[i]], [corner1, ballvectors[i + 1]], [corner1,
ballvectors[i + 2]],
                     [corner1, ballvectors[i + 4]]])
            elif i == 1:
                springvecs2.append(
                    [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i + 2]],
                     [corner1, ballvectors[i + 4]]])
            elif i == 2:
                springvecs2.append(
                    [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i + 1]],
                     [corner1, ballvectors[i + 5]]])
            elif i == 3:
                springvecs2.append(
                    [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i - 3]],
                     [corner1, ballvectors[i + 3]]])

        spring2 = ['s_1', 's_2', 's_3', 's_4', 's_5', 's_6', 's_7', 's_8', 's_9', 's_10', 's_11',
's_12', 's_13',
                   's_14', 's_15', 's_16']

        for i in range(4):
            for j in range(4):
                position2 = springvecs2[i][j][1] - springvecs2[i][j][0]
```

```
                spring2[j + i * 4] = vp.cylinder(pos=springvecs2[i][j][0], axis=position2,
length=vp.mag(position2),
                                                 radius=.02, color=vp.color.white)

        ballname3 = ['b13', 'b14', 'b15', 'b16']
        ballvectors3 = [vp.vector(-0.75, -1, -0.75), vp.vector(-0.75, -1, 1.75), vp.vector(1.75,
-1, 1.75),
                        vp.vector(1.75, -1, -0.75)]
        for i in range(len(ballname3)):
            ballname3[i] = vp.sphere(pos=ballvectors3[i], radius=0.1, color=vp.color.red,
f_k=vp.vector(0, 0, 0))

        springvecs3 = []
        for i in range(4):
            corner1 = ballvectors3[i]
            if i == 0:
                springvecs3.append(
                    [[corner1, ballvectors2[i]], [corner1, ballvectors[i + 1]], [corner1,
ballvectors[i]],
                     [corner1, ballvectors[i + 2]]])
            elif i == 1:
                springvecs3.append(
                    [[corner1, ballvectors2[i]], [corner1, ballvectors[i]], [corner1,
ballvectors[i + 2]],
                     [corner1, ballvectors[i - 1]]])
            elif i == 2:
                springvecs3.append(
                    [[corner1, ballvectors2[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i + 1]],
                     [corner1, ballvectors[i]]])
            elif i == 3:
                springvecs3.append(
                    [[corner1, ballvectors2[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i - 3]],
                     [corner1, ballvectors[i]]])

        spring3 = ['s_1', 's_2', 's_3', 's_4', 's_5', 's_6', 's_7', 's_8', 's_9', 's_10', 's_11',
's_12', 's_13',
                   's_14', 's_15', 's_16']
        for i in range(4):
            for j in range(4):
                position3 = springvecs3[i][j][1] - springvecs3[i][j][0]
                spring3[j + i * 4] = vp.cylinder(pos=springvecs3[i][j][0], axis=position3,
length=vp.mag(position3),
                                                 radius=.02, color=vp.color.white)

        ballnameC = ballname + ballname2 + ballname3

        OriginalCOM = (ballnameC[0].pos + ballnameC[1].pos + ballnameC[2].pos + ballnameC[3].pos
+ ballnameC[4].pos +
                       ballnameC[5].pos + ballnameC[6].pos + ballnameC[7].pos + ballnameC[8].pos
+ ballnameC[9].pos +
                       ballnameC[10].pos + ballnameC[11].pos + ballnameC[12].pos +
ballnameC[13].pos + ballnameC[
                           14].pos + ballnameC[15].pos) / 16

        for i in range(len(ballnameC)):
            ballnameC[i].velocity = vp.vector(0, 0, 0)

        # for i in range(len(ballname2)):
        #   ballname2[i].velocity = vp.vector(0, 0, 0)

        F_c = vp.vector(0, 1000, 0)

        L0 = np.zeros((16, 16))
        for i in range(16):
            for j in range(16):
                if i == j:
                    L0[j][i] = 0
                else:
                    position = ballnameC[j].pos - ballnameC[i].pos
```

```
                    L0[j][i] = vp.mag(position)

        Repeated = []
        total_indices = []
        for x in range(16):
            for y in range(16):
                value = L0[y][x]
                if value not in Repeated:
                    Repeated.append(value)
                    indices = []
                    for n in range(16):
                        for m in range(16):
                            index = []
                            if L0[m][n] == value:
                                index.append(m)
                                index.append(n)
                                indices.append(index)
                    total_indices.append(indices)
        #         print(len(Repeated))

        del total_indices[0]

        ballvectorsC = ballvectors + ballvectors2 + ballvectors3

        F = np.zeros((16, 3))
        ic, fnum = 0, 0
        c = 1
        while 1:
            #vp.rate(50)

            L0rate = np.zeros((16, 16))
            w = 10 * np.pi
            t = 0.001
            #print(total_indices,"ASdfasdfasdf")
           # asdfff = 0
            for i in range(len(total_indices)):
                for z in total_indices[i]:
                    L0rate[z[0]][z[1]] = L0[z[0]][z[1]] + pa1[h][i][0] * sin(w * t +
pa1[h][i][1])
            #         asdfff += 1
           # print(asdfff, len(total_indices), len(total_indices[10]),"ASdfasdfasdfsf")

            for i in range(8):
                ballvectors[i] = ballnameC[i].pos
               # ballnameC[i].pos = ballname[i].pos
#
            springvecs = [[ballvectors[0], ballvectors[4]], [ballvectors[0], ballvectors[1]],
                          [ballvectors[0], ballvectors[2]],
                          [ballvectors[1], ballvectors[5]], [ballvectors[1], ballvectors[3]],
                          [ballvectors[2], ballvectors[6]],
                          [ballvectors[3], ballvectors[7]], [ballvectors[3], ballvectors[2]],
                          [ballvectors[4], ballvectors[5]],
                          [ballvectors[4], ballvectors[6]], [ballvectors[7], ballvectors[5]],
                          [ballvectors[7], ballvectors[6]]]




            for i in range(12):
                position = springvecs[i][1] - springvecs[i][0]  # - L0[i]
                spring[i].pos = springvecs[i][0]
                spring[i].axis = position
                spring[i].length = vp.mag(position)

            for i in range(4):
                ballvectors2[i] = ballnameC[i+8].pos
              #  ballnameC[i+4].pos = ballname2[i].pos


            springvecs2 = []
            for i in range(4):
```

```
                corner1 = ballvectors2[i]
                if i == 0:
                    springvecs2.append(
                        [[corner1, ballvectors[i]], [corner1, ballvectors[i + 1]], [corner1,
ballvectors[i + 2]],
                         [corner1, ballvectors[i + 4]]])
                elif i == 1:
                    springvecs2.append(
                        [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i + 2]],
                         [corner1, ballvectors[i + 4]]])
                elif i == 2:
                    springvecs2.append(
                        [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i + 1]],
                         [corner1, ballvectors[i + 5]]])
                elif i == 3:
                    springvecs2.append(
                        [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i - 3]],
                         [corner1, ballvectors[i + 3]]])

            for i in range(4):
                for j in range(4):
                    position2 = springvecs2[i][j][1] - springvecs2[i][j][0]
                    spring2[j + i * 4].pos = springvecs2[i][j][0]
                    spring2[j + i * 4].axis = position2
                    spring2[j + i * 4].length = vp.mag(position2)

            for i in range(4):
                ballvectors3[i] = ballnameC[i+12].pos
#                ballnameC[i+8].pos = ballname3[i].pos

            springvecs3 = []
            for i in range(4):
                corner1 = ballvectors3[i]
                if i == 0:
                    springvecs3.append(
                        [[corner1, ballvectors2[i]], [corner1, ballvectors[i + 1]], [corner1,
ballvectors[i]],
                         [corner1, ballvectors[i + 2]]])
                elif i == 1:
                    springvecs3.append(
                        [[corner1, ballvectors2[i]], [corner1, ballvectors[i]], [corner1,
ballvectors[i + 2]],
                         [corner1, ballvectors[i - 1]]])
                elif i == 2:
                    springvecs3.append(
                        [[corner1, ballvectors2[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i + 1]],
                         [corner1, ballvectors[i]]])
                elif i == 3:
                    springvecs3.append(
                        [[corner1, ballvectors2[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i - 3]],
                         [corner1, ballvectors[i]]])

            for i in range(4):
                for j in range(4):
                    position3 = springvecs3[i][j][1] - springvecs3[i][j][0]
                    spring3[j + i * 4].pos = springvecs3[i][j][0]
                    spring3[j + i * 4].axis = position3
                    spring3[j + i * 4].length = vp.mag(position3)

            dampening = 0.9
            #ballvectorsC = ballvectors + ballvectors2 + ballvectors3
           # ballnameC = ballname + ballname2 + ballname3

            F_mat = np.zeros((16, 16))
            F_vec = []
            F_v = []
```

```
            a = np.array(np.zeros((16, 16)))
            for i in range(16):
                for k in range(16):
                    if k == i:
                        L = 0
                        F_mat[i][k] = 0
                        F_vec.append(vp.vector(0, 0, 0))
                    else:
                        L = vp.mag(ballnameC[k].pos - ballnameC[i].pos) - L0rate[k][i]
                        F_mat[i][k] = L * k_sp
                        pf0 = ballnameC[k].pos - ballnameC[i].pos
                        F_vec.append(vp.norm(pf0) * L * k_sp)

            a = np.array(F_vec).reshape(16, 16)
            F = a.sum(axis=0)
            # print(pa1, "asdfasdfasdfasdf")
            for i in range(16):

                F[i] = F[i] + g_vector * mass
                if ballnameC[i].pos.y < floor.pos.y:
                    F_N = ((floor.pos.y - ballnameC[i].pos.y) ** 2) * 500
                    F[i].y = 0.99 * (F[i].y - F_N)
                    mu = 1
                    F_st = mu * F_N
                    F_horiz = (F[i].x ** 2 + F[i].z ** 2) ** 0.5
                    v_xz = (ballnameC[i].velocity.x ** 2 + ballnameC[i].velocity.z ** 2) ** 0.5
                    vx = ballnameC[i].velocity.x / v_xz
                    vz = ballnameC[i].velocity.z / v_xz
                    if F_st < F_horiz:
                        F[i].x += F_horiz * vx - F_N * vx
                        F[i].z += F_horiz * vz - F_N * vz
                    else:
                        F[i].x = F_horiz * vx
                        F[i].z = F_horiz * vz
                        ballnameC[i].velocity.x = 0
                        ballnameC[i].velocity.z = 0

            for i in range(16):
                ballnameC[i].velocity -= F[i] / mass * dt
                ballnameC[i].pos += ballnameC[i].velocity * dt
            t += 0.001
            c += 1
            if c == 2000:
                break
        COM = getCOM(ballnameC)
        dvec = COM - OriginalCOM
        dis = sqrt(dvec.x ** 2 + dvec.z ** 2)
        total_dis.append(dis)
    dis_index = np.argsort(total_dis)
    sorted_dis = []
    sorted_pa1 = []
    for i in range(20):
        sorted_dis.append(total_dis[dis_index[i]])
        sorted_pa1.append(pa1[dis_index[i]])
    good_dis = sorted_dis[-10:]
    dots.append(good_dis)
    print(dots)
    print('GOODDIS', good_dis[-1])
    # for i in good_dis:
    #     if i <= 50:
    #         take.append(i)
    best_dis.append(good_dis[-1])
    pa1 = sorted_pa1[-10:]
    print('PA1END', len(pa1))

print(best_dis)
print(pa1[-1])
evals = list(range(1, 3))

plt.plot(evals, best_dis)
plt.xlabel('Evaluations')
```

```
plt.ylabel('Distance Moved')
plt.title('Evolutionary Algorithm')
plt.show()

plt.plot(evals, dots)
plt.xlabel('Evaluations')
plt.ylabel('Distance Moved')
plt.title('Evolutionary Algorithm Dot Plot')
plt.show()
```

Innovative Robot SIMULATED

```
##INNOVATIVE SPIDER

import vpython as vp
import itertools
import numpy as np
from math import *
import random
import matplotlib.pyplot as plt

scene = vp.canvas()  # title = 'Box', width = 600, height = 400, center = vp.vector(0,0,0))

floor = vp.box(pos=vp.vector(0, -1.3, 0), length=10, height=0.01, width=10, color=vp.color.blue)


def getCOM(v):
    COM = (v[0].pos + v[1].pos + v[2].pos + v[3].pos + v[4].pos + v[5].pos + v[6].pos + v[7].pos
+ v[8].pos + v[9].pos +
           v[10].pos + v[11].pos + v[12].pos + v[13].pos + v[14].pos + v[15].pos) / 16
    return COM


# def mutation(r):
#     ran = random.randint(0, 9)
#     if ran > mutation_rate:
#         x = random.randint(0, 25)
#         y = random.randint(0, 25)
#         r[x], r[y] = r[y], r[x]
#         return r
#     else:
#         return r


# def Crossover(m, n):
#     M = []
#     N = []
#     for i in range(len(m)):
#         M.append(m[i])
#         N.append(n[i])
#     rand = random.randint(0, 20)
#     index = list(range(rand, rand + 6))
#     exchangem = []
#     exchangen = []
#     for i in range(len(index)):
#         exchangem.append(M[index[i]])
#         exchangen.append(N[index[i]])
#     for j in range(len(index)):
#         M[index[j]] = exchangen[j]
#         N[index[j]] = exchangem[j]
#     return (M, N)


# dis1 = 0
# good_dis = []
# best_dis = []
# mutation_rate = 7
```

```
v = 0
dt = 0.001
mass = 0.1
g = 9.81
k_sp = 2000
g_vector = vp.vector(0, 9.81, 0)

pa1 = [[-0.434696229094949, -1.2513194371194707], [0.08320033896360135, -0.12132769464311943], [-
0.23952514778932565, -0.6977196175028366], [0.48778400691319856, 2.174381800139548], [-
0.2086658768794406, -0.8779996844241418], [-0.376357657290179, 2.323553214736738], [-
0.46594984705057696, 0.8851432707864486], [0.34104851691186977, -2.4616366766545146],
[0.1277890758222554, -2.8152012870759657], [-0.1542795204052102, 1.0151808874895316],
[0.33554579321790756, -1.4811187764472873], [-0.24944431238178677, 1.5579879810530812],
[0.26853311646183675, 0.6564760856120833], [-0.4764843390912038, -1.6597697864720173], [-
0.36161061755303936, -1.0848908722658197], [0.03221651756881061, -2.79348292717759],
[0.19736385791164968, -1.6973394522742107], [0.000505719241802316, -1.459622399049625],
[0.2658082066664623, -1.0188028554505872], [0.30572015785923634, -0.13535255662189716], [-
0.4872600197470831, 0.4719150076073926], [0.3763770407890543, 0.6224060587818538],
[0.45363750141297743, 2.697225612527917 7], [-0.18382280642087878, 1.2425265400923005], [-
0.12602938437001165, -0.6795728984817262], [0.02406127199734276, -0.10955006838600667]]
# for i in range(10):
#     pa = []
#     for j in range(26):
#         p = []
#         b = np.random.uniform(-0.4, 0.4)
#         c = np.random.uniform(-np.pi, np.pi)
#         p.append(b)
#         p.append(c)
#         pa.append(p)

#     pa1.append(pa)

# dots = []

# for a in range(2):
#     print(a, "gen")
#     for i in np.arange(0, 10, 2):
#         child1, child2 = Crossover(pa1[i], pa1[i + 1])

#         pa1.append(child1)
#         pa1.append(child2)

#     for i in range(10, 20):
#         pa1[i] = mutation(pa1[i])

#     total_dis = []
#     take = []
#     for h in range(20):
#         print(h)
ballname = ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8']
ballvectors = [vp.vector(0, 0, 0), vp.vector(0, 0, 1), vp.vector(1, 0, 0), vp.vector(1, 0, 1),
               vp.vector(0.25, 0.5, 0.25), vp.vector(0.25, 0.5, 0.75), vp.vector(0.75, 0.5,
0.25),
               vp.vector(0.75, 0.5, 0.75)]

for i in range(len(ballname)):
    ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.1, color=vp.color.red, f_k=vp.vector(0,
0, 0))

springvecs = [[ballvectors[0], ballvectors[4]], [ballvectors[0], ballvectors[1]],
              [ballvectors[0], ballvectors[2]],
              [ballvectors[1], ballvectors[5]], [ballvectors[1], ballvectors[3]],
              [ballvectors[2], ballvectors[6]],
              [ballvectors[3], ballvectors[7]], [ballvectors[3], ballvectors[2]],
              [ballvectors[4], ballvectors[5]],
              [ballvectors[4], ballvectors[6]], [ballvectors[7], ballvectors[5]],
              [ballvectors[7], ballvectors[6]]]

spring = ['s1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11',
          's12']  # , 's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20','s21', 's22', 's23',
's24', 's25', 's26', 's27', 's28']
```

```
for i in range(len(springvecs)):
    position = springvecs[i][1] - springvecs[i][0]
    spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
radius=.02,
                            color=vp.color.white)

ballname2 = ['b9', 'b10', 'b11', 'b12']
ballvectors2 = [vp.vector(-0.5, -0.25, -0.5), vp.vector(-0.5, -0.25, 1.5), vp.vector(1.5, -0.25,
1.5),
                vp.vector(1.5, -0.25, -0.5)]
for i in range(len(ballname2)):
    ballname2[i] = vp.sphere(pos=ballvectors2[i], radius=0.1, color=vp.color.red,
f_k=vp.vector(0, 0, 0))
springvecs2 = []
for i in range(4):
    corner1 = ballvectors2[i]
    if i == 0:
        springvecs2.append(
            [[corner1, ballvectors[i]], [corner1, ballvectors[i + 1]], [corner1, ballvectors[i +
2]],
             [corner1, ballvectors[i + 4]]])
    elif i == 1:
        springvecs2.append(
            [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1, ballvectors[i +
2]],
             [corner1, ballvectors[i + 4]]])
    elif i == 2:
        springvecs2.append(
            [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1, ballvectors[i +
1]],
             [corner1, ballvectors[i + 5]]])
    elif i == 3:
        springvecs2.append(
            [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1, ballvectors[i -
3]],
             [corner1, ballvectors[i + 3]]])

spring2 = ['s_1', 's_2', 's_3', 's_4', 's_5', 's_6', 's_7', 's_8', 's_9', 's_10', 's_11', 's_12',
's_13',
           's_14', 's_15', 's_16']

for i in range(4):
    for j in range(4):
        position2 = springvecs2[i][j][1] - springvecs2[i][j][0]
        spring2[j + i * 4] = vp.cylinder(pos=springvecs2[i][j][0], axis=position2,
length=vp.mag(position2),
                                         radius=.02, color=vp.color.white)

ballname3 = ['b13', 'b14', 'b15', 'b16']
ballvectors3 = [vp.vector(-0.75, -1, -0.75), vp.vector(-0.75, -1, 1.75), vp.vector(1.75, -1,
1.75),
                vp.vector(1.75, -1, -0.75)]
for i in range(len(ballname3)):
    ballname3[i] = vp.sphere(pos=ballvectors3[i], radius=0.1, color=vp.color.red,
f_k=vp.vector(0, 0, 0))

springvecs3 = []
for i in range(4):
    corner1 = ballvectors3[i]
    if i == 0:
        springvecs3.append(
            [[corner1, ballvectors2[i]], [corner1, ballvectors[i + 1]], [corner1,
ballvectors[i]],
             [corner1, ballvectors[i + 2]]])
    elif i == 1:
        springvecs3.append(
            [[corner1, ballvectors2[i]], [corner1, ballvectors[i]], [corner1, ballvectors[i +
2]],
             [corner1, ballvectors[i - 1]]])
    elif i == 2:
```

```
        springvecs3.append(
            [[corner1, ballvectors2[i]], [corner1, ballvectors[i - 1]], [corner1, ballvectors[i +
1]],
             [corner1, ballvectors[i]]])
    elif i == 3:
        springvecs3.append(
            [[corner1, ballvectors2[i]], [corner1, ballvectors[i - 1]], [corner1, ballvectors[i -
3]],
             [corner1, ballvectors[i]]])

spring3 = ['s_1', 's_2', 's_3', 's_4', 's_5', 's_6', 's_7', 's_8', 's_9', 's_10', 's_11', 's_12',
's_13',
           's_14', 's_15', 's_16']
for i in range(4):
    for j in range(4):
        position3 = springvecs3[i][j][1] - springvecs3[i][j][0]
        spring3[j + i * 4] = vp.cylinder(pos=springvecs3[i][j][0], axis=position3,
length=vp.mag(position3),
                                         radius=.02, color=vp.color.white)

ballnameC = ballname + ballname2 + ballname3

OriginalCOM = (ballnameC[0].pos + ballnameC[1].pos + ballnameC[2].pos + ballnameC[3].pos +
ballnameC[4].pos +
               ballnameC[5].pos + ballnameC[6].pos + ballnameC[7].pos + ballnameC[8].pos +
ballnameC[9].pos +
               ballnameC[10].pos + ballnameC[11].pos + ballnameC[12].pos + ballnameC[13].pos +
ballnameC[
                   14].pos + ballnameC[15].pos) / 16

for i in range(len(ballnameC)):
    ballnameC[i].velocity = vp.vector(0, 0, 0)

# for i in range(len(ballname2)):
#   ballname2[i].velocity = vp.vector(0, 0, 0)

F_c = vp.vector(0, 1000, 0)

L0 = np.zeros((16, 16))
for i in range(16):
    for j in range(16):
        if i == j:
            L0[j][i] = 0
        else:
            position = ballnameC[j].pos - ballnameC[i].pos
            L0[j][i] = vp.mag(position)

Repeated = []
total_indices = []
for x in range(16):
    for y in range(16):
        value = L0[y][x]
        if value not in Repeated:
            Repeated.append(value)
            indices = []
            for n in range(16):
                for m in range(16):
                    index = []
                    if L0[m][n] == value:
                        index.append(m)
                        index.append(n)
                        indices.append(index)
            total_indices.append(indices)
#         print(len(Repeated))

del total_indices[0]

ballvectorsC = ballvectors + ballvectors2 + ballvectors3

F = np.zeros((16, 3))
ic, fnum = 0, 0
```

```
c = 1
while 1:
    vp.rate(150)

    L0rate = np.zeros((16, 16))
    w = 10 * np.pi
    t = 0.001
    #print(total_indices,"ASdfasdfasdf")
   # asdfff = 0
    for i in range(len(total_indices)):
        for z in total_indices[i]:
            L0rate[z[0]][z[1]] = L0[z[0]][z[1]] + pa1[i][0] * sin(w * t + pa1[i][1])
    #         asdfff += 1
   # print(asdfff, len(total_indices), len(total_indices[10]),"ASdfasdfasdfsf")

    for i in range(8):
        ballvectors[i] = ballnameC[i].pos
       # ballnameC[i].pos = ballname[i].pos
#
    springvecs = [[ballvectors[0], ballvectors[4]], [ballvectors[0], ballvectors[1]],
                  [ballvectors[0], ballvectors[2]],
                  [ballvectors[1], ballvectors[5]], [ballvectors[1], ballvectors[3]],
                  [ballvectors[2], ballvectors[6]],
                  [ballvectors[3], ballvectors[7]], [ballvectors[3], ballvectors[2]],
                  [ballvectors[4], ballvectors[5]],
                  [ballvectors[4], ballvectors[6]], [ballvectors[7], ballvectors[5]],
                  [ballvectors[7], ballvectors[6]]]




    for i in range(12):
        position = springvecs[i][1] - springvecs[i][0]  # - L0[i]
        spring[i].pos = springvecs[i][0]
        spring[i].axis = position
        spring[i].length = vp.mag(position)

    for i in range(4):
        ballvectors2[i] = ballnameC[i+8].pos
      #  ballnameC[i+4].pos = ballname2[i].pos


    springvecs2 = []
    for i in range(4):
        corner1 = ballvectors2[i]
        if i == 0:
            springvecs2.append(
                [[corner1, ballvectors[i]], [corner1, ballvectors[i + 1]], [corner1,
ballvectors[i + 2]],
                 [corner1, ballvectors[i + 4]]])
        elif i == 1:
            springvecs2.append(
                [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i + 2]],
                 [corner1, ballvectors[i + 4]]])
        elif i == 2:
            springvecs2.append(
                [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i + 1]],
                 [corner1, ballvectors[i + 5]]])
        elif i == 3:
            springvecs2.append(
                [[corner1, ballvectors[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i - 3]],
                 [corner1, ballvectors[i + 3]]])

    for i in range(4):
        for j in range(4):
            position2 = springvecs2[i][j][1] - springvecs2[i][j][0]
            spring2[j + i * 4].pos = springvecs2[i][j][0]
            spring2[j + i * 4].axis = position2
```

```
            spring2[j + i * 4].length = vp.mag(position2)

    for i in range(4):
        ballvectors3[i] = ballnameC[i+12].pos
#                ballnameC[i+8].pos = ballname3[i].pos

    springvecs3 = []
    for i in range(4):
        corner1 = ballvectors3[i]
        if i == 0:
            springvecs3.append(
                [[corner1, ballvectors2[i]], [corner1, ballvectors[i + 1]], [corner1,
ballvectors[i]],
                 [corner1, ballvectors[i + 2]]])
        elif i == 1:
            springvecs3.append(
                [[corner1, ballvectors2[i]], [corner1, ballvectors[i]], [corner1, ballvectors[i +
2]],
                 [corner1, ballvectors[i - 1]]])
        elif i == 2:
            springvecs3.append(
                [[corner1, ballvectors2[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i + 1]],
                 [corner1, ballvectors[i]]])
        elif i == 3:
            springvecs3.append(
                [[corner1, ballvectors2[i]], [corner1, ballvectors[i - 1]], [corner1,
ballvectors[i - 3]],
                 [corner1, ballvectors[i]]])

    for i in range(4):
        for j in range(4):
            position3 = springvecs3[i][j][1] - springvecs3[i][j][0]
            spring3[j + i * 4].pos = springvecs3[i][j][0]
            spring3[j + i * 4].axis = position3
            spring3[j + i * 4].length = vp.mag(position3)

    dampening = 0.9
    #ballvectorsC = ballvectors + ballvectors2 + ballvectors3
   # ballnameC = ballname + ballname2 + ballname3

    F_mat = np.zeros((16, 16))
    F_vec = []
    F_v = []
    a = np.array(np.zeros((16, 16)))
    for i in range(16):
        for k in range(16):
            if k == i:
                L = 0
                F_mat[i][k] = 0
                F_vec.append(vp.vector(0, 0, 0))
            else:
                L = vp.mag(ballnameC[k].pos - ballnameC[i].pos) - L0rate[k][i]
                F_mat[i][k] = L * k_sp
                pf0 = ballnameC[k].pos - ballnameC[i].pos
                F_vec.append(vp.norm(pf0) * L * k_sp)

    a = np.array(F_vec).reshape(16, 16)
    F = a.sum(axis=0)
   # print(pa1, "asdfasdfasdfasdf")
    for i in range(16):

        F[i] = F[i] + g_vector * mass
        if ballnameC[i].pos.y < floor.pos.y:
            F_N = ((floor.pos.y - ballnameC[i].pos.y) ** 2) * 1000
            F[i].y = 0.99 * (F[i].y - F_N)
            mu = 1
            F_st = mu * F_N
            F_horiz = (F[i].x ** 2 + F[i].z ** 2) ** 0.5
            v_xz = (ballnameC[i].velocity.x ** 2 + ballnameC[i].velocity.z ** 2) ** 0.5
            vx = ballnameC[i].velocity.x / v_xz
```

```
            vz = ballnameC[i].velocity.z / v_xz
            if F_st < F_horiz:
                F[i].x += F_horiz * vx - F_N * vx
                F[i].z += F_horiz * vz - F_N * vz
            else:
                F[i].x = F_horiz * vx
                F[i].z = F_horiz * vz
                ballnameC[i].velocity.x = 0
                ballnameC[i].velocity.z = 0

    for i in range(16):
        ballnameC[i].velocity -= F[i] / mass * dt
        ballnameC[i].pos += ballnameC[i].velocity * dt
    t += 0.001
    c += 1
    if c == 8000:
        break
#         COM = getCOM(ballnameC)
#         dvec = COM - OriginalCOM
#         dis = sqrt(dvec.x * 2 + dvec.z * 2)
#         total_dis.append(dis)
#     dis_index = np.argsort(total_dis)
#     sorted_dis = []
#     sorted_pa1 = []
#     for i in range(20):
#         sorted_dis.append(total_dis[dis_index[i]])
#         sorted_pa1.append(pa1[dis_index[i]])
#     good_dis = sorted_dis[-10:]
#     dots.append(good_dis)
#     print(dots)
#     print('GOODDIS', good_dis[-1])
#     # for i in good_dis:
#     #     if i <= 50:
#     #         take.append(i)
#     best_dis.append(good_dis[-1])
#     pa1 = sorted_pa1[-10:]
#     print('PA1END', len(pa1))

# print(best_dis)
# print(pa1[-1])
# evals = list(range(1, 3))

# plt.plot(evals, best_dis)
# plt.xlabel('Evaluations')
# plt.ylabel('Distance Moved')
# plt.title('Evolutionary Algorithm')
# plt.show()

# plt.plot(evals, dots)
# plt.xlabel('Evaluations')
# plt.ylabel('Distance Moved')
# plt.title('Evolutionary Algorithm Dot Plot')
# plt.show()
```