COLUMBIA UNIVERSITY

MECE 4510 EVOLUTIONARY COMPUTATION AND DESIGN AUTOMATION

Assignment3 Phase B

Yifan Gui
UNI: yg2751

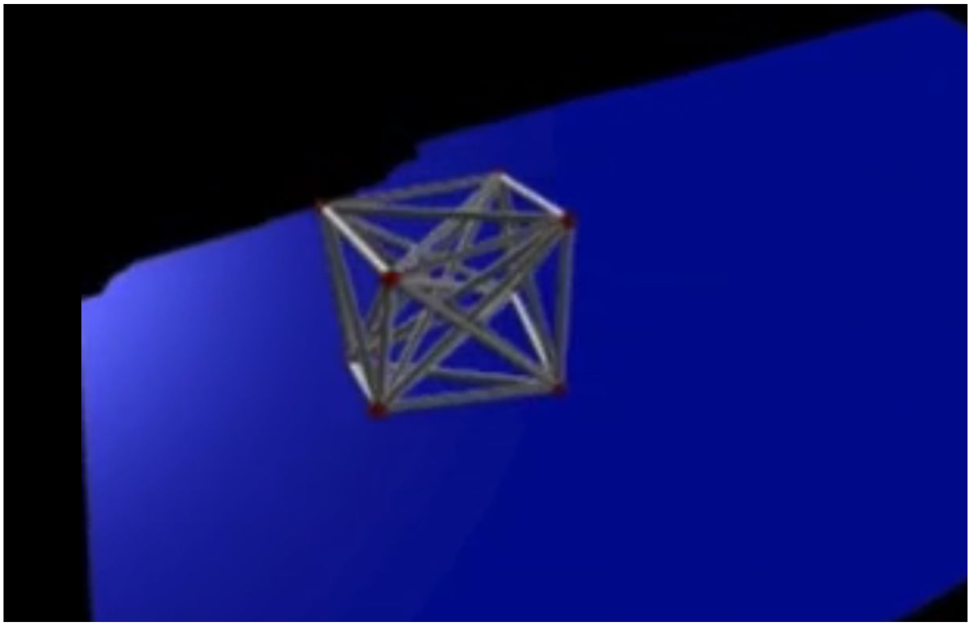
Instructor:
Dr. Hod Lipson

Grace Hour Used: 0
Grace Hour Gained: 0
Grace Hour Remaining: 92

Nov 24 , 2020

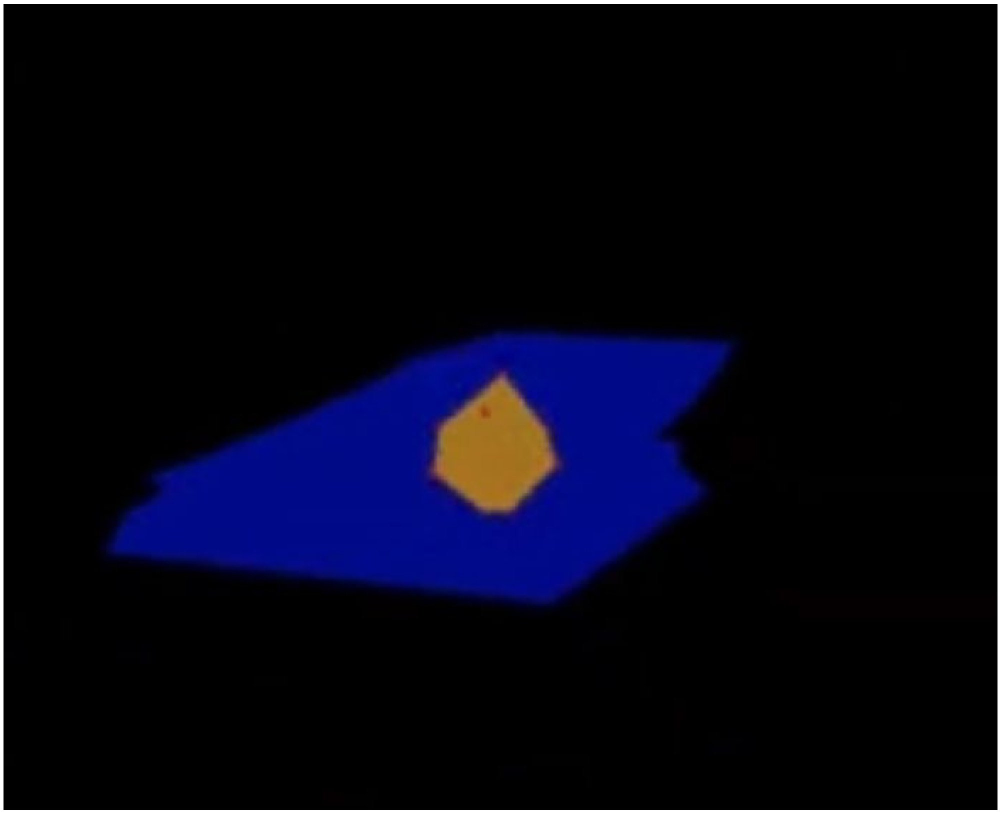
Result Summary

Fastest robot running cycles



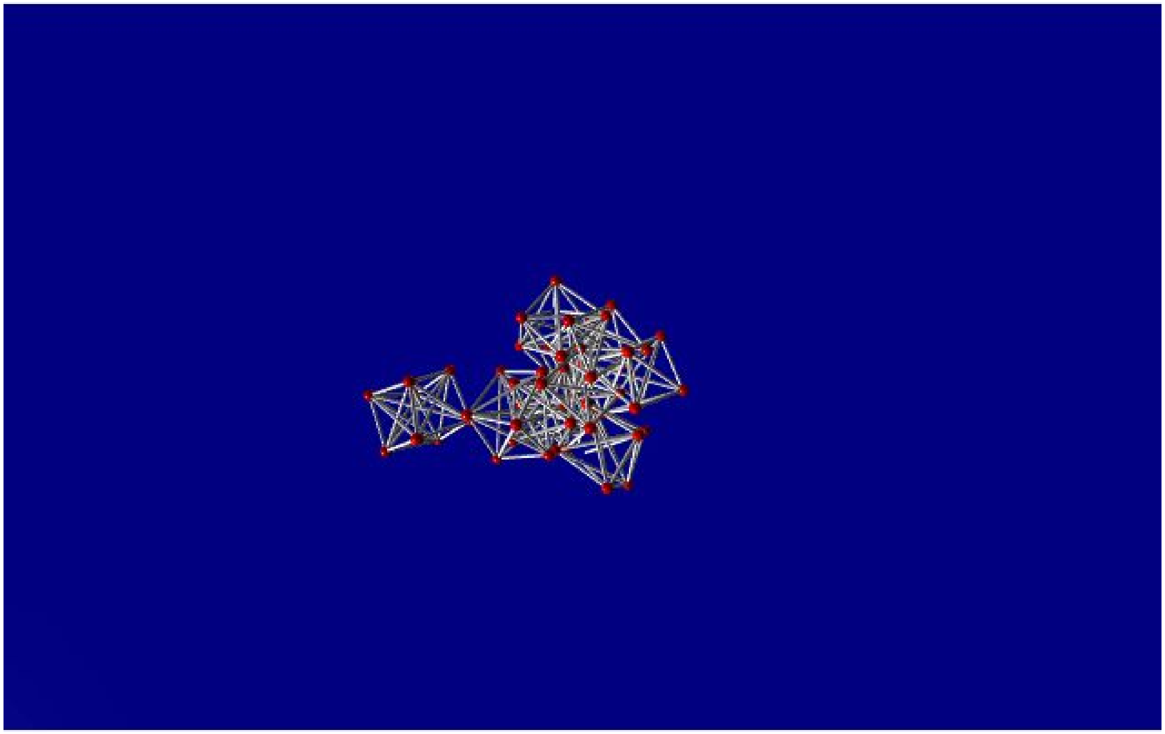
https://youtu.be/03SL64Pb0M4

Fastest robot bouncing test



https://youtu.be/J_RXgzODi4k

Multiple Robots



https://youtu.be/MChHPKd2zVg

Description of Design

In Phase B, the goal is to make a fixed morphology robot by one or more cubes from Phase A and evolve it to obtain a robot that moves the fastest. Since the simulation involves complex structures and lots of physical behaviors which need massive computation power, we eventually decided to implement it in python with a single cube structure.

GA Representation

The rest length of springs are controlled by the following equation:

$$L = L_0 + A * sin(B * t + C)$$

The genetic algorithm evolves parameters A, B and C with the purpose of finding the fastest moving robot.

Design parameters

Simulation parameters

- mass = 0.5
- length = 1
- gravity = 9.81
- time step = 0.001

Robot parameters

- mass = 0.5
- length of cube = 1
- soft spring constant = 2000
- hard spring constant = 20000

Evolutionary Parameters

- population size = 2000
- mutation rate = 0.1
- crossover rate = 0.2

Analysis

Simulation:

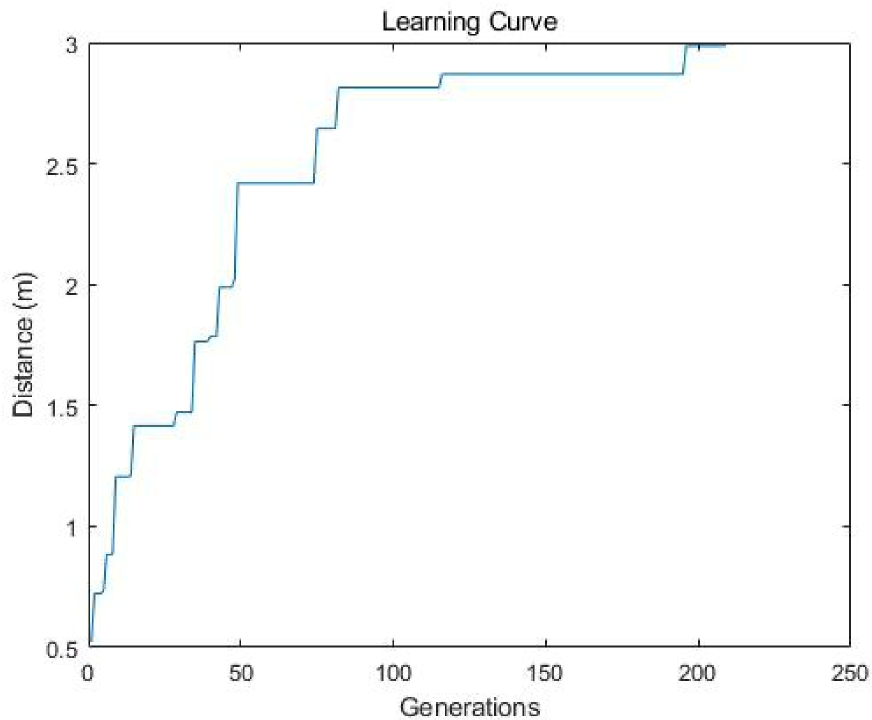
In general, the simulation of a soft moving cube robot in Python runs well and the speed of simulation is satisfactory. The main issue is that the ending position of the robot is not the same and the reason could be the calculation error in length change of the robot length.

Robot:
A cube is simple enough to start with, however, it is hard to observe the motion since the robot "wriggles" instead of "walking". This could be improved if more GPU calculation power can be accessed and a robot with more cubes that can evolve into different shapes.

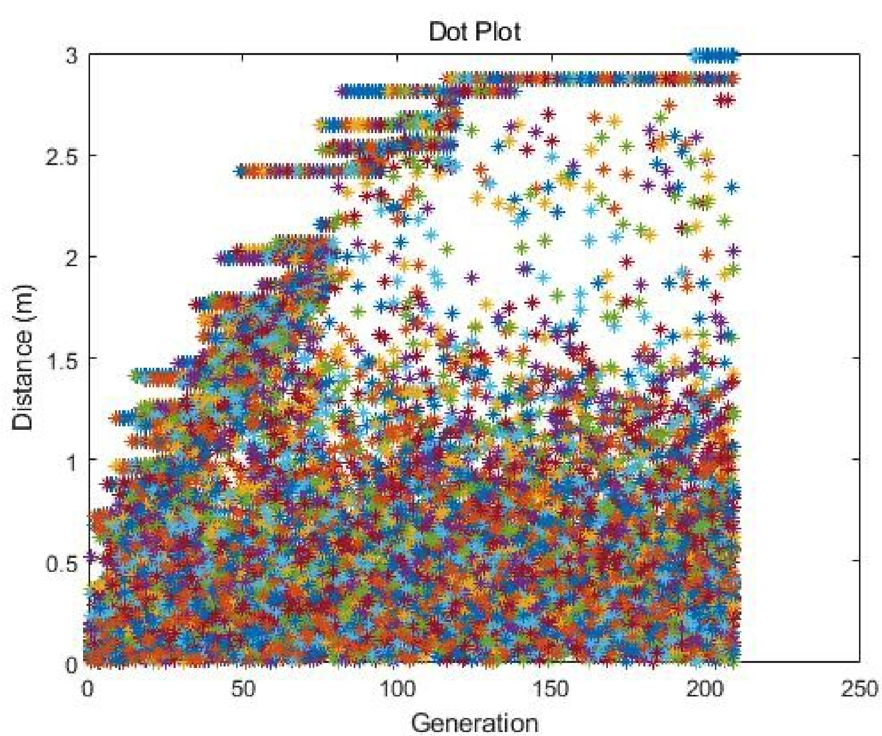
Genetic Algorithm:
The GA is used to evolve A, B, C parameters in the locomotion pattern. In a cube robot, there are 8 vertices and 28 springs. The fitness of selecting robots is by choosing the 50% of the fastest solutions from the offspring. Each individual cube robot is treated as a solution, the mutation works on randomly changing values of A, B, C parameters and the crossover is between two different solutions. The fitness of selecting robots is by choosing the 50% of the fastest solutions from the offspring.

Performance Plots

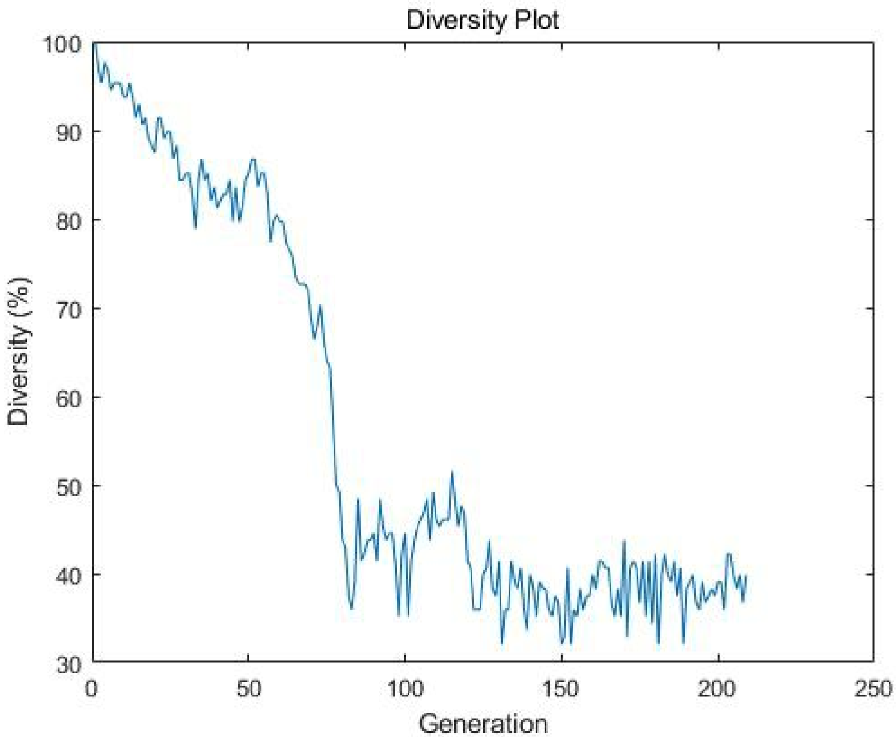
Learning Curve



Dot Plot



Diversity Plot



Appendix

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


scene = vp.canvas()
floor = vp.box(pos=vp.vector(0, 0, 0), length=100, height=0.001,
width=100, 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)]

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 = "texture.jpg" )   # texture


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

```
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']

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

g_vector = vp.vector(0,g,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)/ 16
    return COM

pa1 = [[-0.0999684941118836, -2.436063476356824, 2296.7581897299874],
    [-0.07967391969199222, -0.08634880798548794, 2361.5826862217555],
    [0.09067974114260546, -0.21159318157779383, 4755.324973113422],
    [0.18380205176538172, -2.0565408755726033, 4409.389811699451 5],
    [-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, 2.391407246169980 5, 2474.754155099656],
    [-0.11773307676195617, -0.3926900646112008, 1067.4545888208283],
    [0.1889606677054101, 1.5196174154857083, 5192.421195053015],
    [0.03581610519811348 5, -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, -0.975609402032411 3, 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)
L0rate = np.zeros((8,8))
t = 0.001
c = 1
w = 10 * np.pi
eta = 1
while True:
    vp.rate(100)
    floor =
vp.box(pos=vp.vector(ballvectors[5].x,0,ballvectors[1].z),length=5,he
ight = 0.001,width=5,color=vp.color.blue)
    scene.forward = vp.vector(-1,-1,1.5)
    scene.center.y = ballvectors[1].y
    scene.center.z = ballvectors[1].z
    L0rate[0][1] = L0[0][1] + pa1[0][0] * sin(w * t + pa1[0][1])
    L0rate[1][0] = L0[1][0] + pa1[0][0] * sin(w * t + pa1[0][1])
    L0rate[0][2] = L0[0][2] + pa1[1][0] * sin(w * t + pa1[1][1])
    L0rate[2][0] = L0[2][0] + pa1[1][0] * sin(w * t + pa1[1][1])
    L0rate[0][3] = L0[0][3] + pa1[2][0] * sin(w * t + pa1[2][1])
    L0rate[3][0] = L0[3][0] + pa1[2][0] * sin(w * t + pa1[2][1])
    L0rate[0][4] = L0[0][4] + pa1[3][0] * sin(w * t + pa1[3][1])
    L0rate[4][0] = L0[4][0] + pa1[3][0] * sin(w * t + pa1[3][1])
    L0rate[0][5] = L0[0][5] + pa1[4][0] * sin(w * t + pa1[4][1])
    L0rate[5][0] = L0[5][0] + pa1[4][0] * sin(w * t + pa1[4][1])
    L0rate[0][6] = L0[0][6] + pa1[5][0] * sin(w * t + pa1[5][1])
    L0rate[6][0] = L0[6][0] + pa1[5][0] * sin(w * t + pa1[5][1])
    L0rate[0][7] = L0[0][7] + pa1[6][0] * sin(w * t + pa1[6][1])
    L0rate[7][0] = L0[7][0] + pa1[6][0] * sin(w * t + pa1[6][1])
    L0rate[1][2] = L0[1][2] + pa1[7][0] * sin(w * t + pa1[7][1])
    L0rate[2][1] = L0[2][1] + pa1[7][0] * sin(w * t + pa1[7][1])
    L0rate[1][3] = L0[1][3] + pa1[8][0] * sin(w * t + pa1[8][1])
    L0rate[3][1] = L0[3][1] + pa1[8][0] * sin(w * t + pa1[8][1])
```

```
    L0rate[1][4] = L0[1][4] + pa1[9][0] * sin(w * t + pa1[9][1])
    L0rate[4][1] = L0[4][1] + pa1[9][0] * sin(w * t + pa1[9][1])
    L0rate[1][5] = L0[1][5] + pa1[10][0] * sin(w * t + pa1[10][1])
    L0rate[5][1] = L0[5][1] + pa1[10][0] * sin(w * t + pa1[10][1])
    L0rate[1][6] = L0[1][6] + pa1[11][0] * sin(w * t + pa1[11][1])
    L0rate[6][1] = L0[6][1] + pa1[11][0] * sin(w * t + pa1[11][1])
    L0rate[1][7] = L0[1][7] + pa1[12][0] * sin(w * t + pa1[12][1])
    L0rate[7][1] = L0[7][1] + pa1[12][0] * sin(w * t + pa1[12][1])
    L0rate[2][3] = L0[2][3] + pa1[13][0] * sin(w * t + pa1[13][1])
    L0rate[3][2] = L0[3][2] + pa1[13][0] * sin(w * t + pa1[13][1])
    L0rate[2][4] = L0[2][4] + pa1[14][0] * sin(w * t + pa1[14][1])
    L0rate[4][2] = L0[4][2] + pa1[14][0] * sin(w * t + pa1[14][1])
    L0rate[2][5] = L0[2][5] + pa1[15][0] * sin(w * t + pa1[15][1])
    L0rate[5][2] = L0[5][2] + pa1[15][0] * sin(w * t + pa1[15][1])
    L0rate[2][6] = L0[2][6] + pa1[16][0] * sin(w * t + pa1[16][1])
    L0rate[6][2] = L0[6][2] + pa1[16][0] * sin(w * t + pa1[16][1])
    L0rate[2][7] = L0[2][7] + pa1[17][0] * sin(w * t + pa1[17][1])
    L0rate[7][2] = L0[7][2] + pa1[17][0] * sin(w * t + pa1[17][1])
    L0rate[3][4] = L0[3][4] + pa1[18][0] * sin(w * t + pa1[18][1])
    L0rate[4][3] = L0[4][3] + pa1[18][0] * sin(w * t + pa1[18][1])
    L0rate[3][5] = L0[3][5] + pa1[19][0] * sin(w * t + pa1[19][1])
    L0rate[5][3] = L0[5][3] + pa1[19][0] * sin(w * t + pa1[19][1])
    L0rate[3][6] = L0[3][6] + pa1[20][0] * sin(w * t + pa1[20][1])
    L0rate[6][3] = L0[6][3] + pa1[20][0] * sin(w * t + pa1[20][1])
    L0rate[3][7] = L0[3][7] + pa1[21][0] * sin(w * t + pa1[21][1])
    L0rate[7][3] = L0[7][3] + pa1[21][0] * sin(w * t + pa1[21][1])
    L0rate[4][5] = L0[4][5] + pa1[22][0] * sin(w * t + pa1[22][1])
    L0rate[5][4] = L0[5][4] + pa1[22][0] * sin(w * t + pa1[22][1])
    L0rate[4][6] = L0[4][6] + pa1[23][0] * sin(w * t + pa1[23][1])
    L0rate[6][4] = L0[6][4] + pa1[23][0] * sin(w * t + pa1[23][1])
    L0rate[4][7] = L0[4][7] + pa1[24][0] * sin(w * t + pa1[24][1])
    L0rate[7][4] = L0[7][4] + pa1[24][0] * sin(w * t + pa1[24][1])
    L0rate[5][6] = L0[5][6] + pa1[25][0] * sin(w * t + pa1[25][1])
    L0rate[6][5] = L0[6][5] + pa1[25][0] * sin(w * t + pa1[25][1])
    L0rate[5][7] = L0[5][7] + pa1[26][0] * sin(w * t + pa1[26][1])
    L0rate[7][5] = L0[7][5] + pa1[26][0] * sin(w * t + pa1[26][1])
    L0rate[6][7] = L0[6][7] + pa1[27][0] * sin(w * t + pa1[27][1])
    L0rate[7][6] = L0[7][6] + pa1[27][0] * sin(w * t + pa1[27][1])
    ks = np.zeros((8, 8))
    ks[0][1] = pa1[0][2]
    ks[1][0] = pa1[0][2]
    ks[0][2] = pa1[1][2]
    ks[2][0] = pa1[1][2]
    ks[0][3] = pa1[2][2]
    ks[3][0] = pa1[2][2]
```

```
    ks[0][4] = pa1[3][2]
    ks[4][0] = pa1[3][2]
    ks[0][5] = pa1[4][2]
    ks[5][0] = pa1[4][2]
    ks[0][6] = pa1[5][2]
    ks[6][0] = pa1[5][2]
    ks[0][7] = pa1[6][2]
    ks[7][0] = pa1[6][2]
    ks[1][2] = pa1[7][2]
    ks[2][1] = pa1[7][2]
    ks[1][3] = pa1[8][2]
    ks[3][1] = pa1[8][2]
    ks[1][4] = pa1[9][2]
    ks[4][1] = pa1[9][2]
    ks[1][5] = pa1[10][2]
    ks[5][1] = pa1[10][2]
    ks[1][6] = pa1[11][2]
    ks[6][1] = pa1[11][2]
    ks[1][7] = pa1[12][2]
    ks[7][1] = pa1[12][2]
    ks[2][3] = pa1[13][2]
    ks[3][2] = pa1[13][2]
    ks[2][4] = pa1[14][2]
    ks[4][2] = pa1[14][2]
    ks[2][5] = pa1[15][2]
    ks[5][2] = pa1[15][2]
    ks[2][6] = pa1[16][2]
    ks[6][2] = pa1[16][2]
    ks[2][7] = pa1[17][2]
    ks[7][2] = pa1[17][2]
    ks[3][4] = pa1[18][2]
    ks[4][3] = pa1[18][2]
    ks[3][5] = pa1[19][2]
    ks[5][3] = pa1[19][2]
    ks[3][6] = pa1[20][2]
    ks[6][3] = pa1[20][2]
    ks[3][7] = pa1[21][2]
    ks[7][3] = pa1[21][2]
    ks[4][5] = pa1[22][2]
    ks[5][4] = pa1[22][2]
    ks[4][6] = pa1[23][2]
    ks[6][4] = pa1[23][2]
    ks[4][7] = pa1[24][2]
    ks[7][4] = pa1[24][2]
    ks[5][6] = pa1[25][2]
```

```
    ks[6][5] = pa1[25][2]
    ks[5][7] = pa1[26][2]
    ks[7][5] = pa1[26][2]
    ks[6][7] = pa1[27][2]
    ks[7][6] = pa1[27][2]

    t += 0.001
    for i in range(8):
        ballvectors[i] = ballname[i].pos
    springvecs = []
    for z in itertools.combinations(ballvectors, 2):
        springvecs.append(z)


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

    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 = 1




    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(10):
    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))
"""
```