ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**«ДИНАМИКА СИСТЕМЫ»**

**ПО ДИСЦИПЛИНЕ «ТЕОРЕТИЧЕСКАЯ МЕХАНИКА И ОСНОВЫ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ»**

**ВАРИАНТ ЗАДАНИЯ № 19**

Выполнил(а) студент группы М8О-208Б-21

Попов Николай Александроич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

Зав. каф. 802, Бардин Б.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Симуляция движения системы с двумя степенями свободы**

Задание: проинтегрировать систему дифференциальных уравнений движения системы с двумя степенями свободы при помощи средств Python. Построить анимацию движения системы, а также графики законов движения системы. А также составить уравнение Лагранжа второго рода для заданной системы.

Расчет уравнения Лагранжа:

1) Определить количество степеней свободы системы 𝑛.

2) В соответствии с количеством степеней свободы системы, полученным в п. 1, ввести соответствующее число обобщенных координат (𝑞1,𝑞2,…,𝑞𝑛).

3) Посчитать кинетическую энергию в зависимости от обобщенных координат и обобщенных скоростей 𝑇(𝑞1,…,𝑞𝑛,𝑞̇1,…,𝑞̇𝑛).

4) Найти обобщенные силы, которые соответствуют обобщенным координатам (𝑄1,…,𝑄𝑛).

5) Составить 𝑛 уравнений Лагранжа второго рода. Общий вид -го уравнения Лагранжа второго рода выглядит следующим образом:

()− =𝑄𝑖.

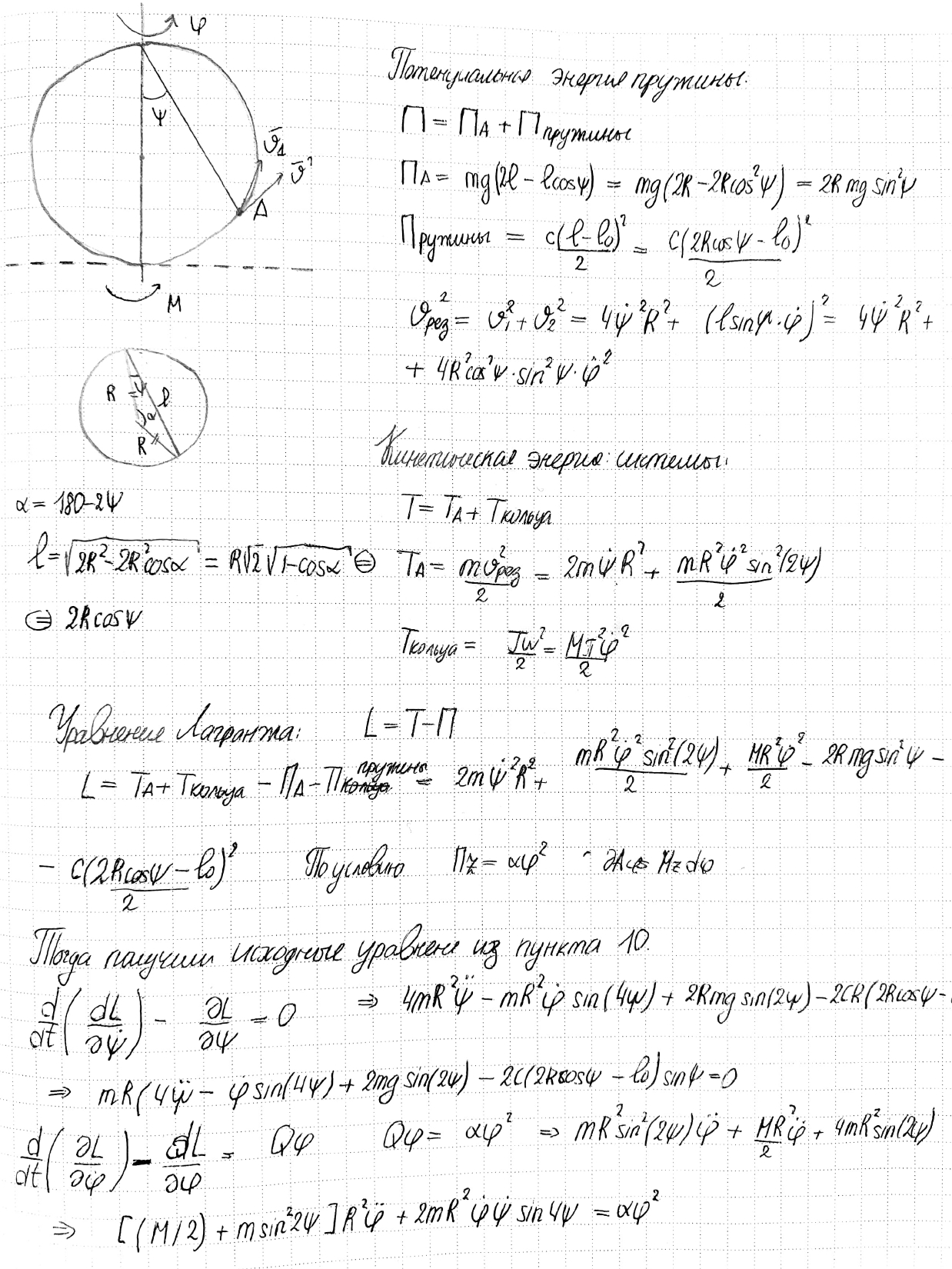
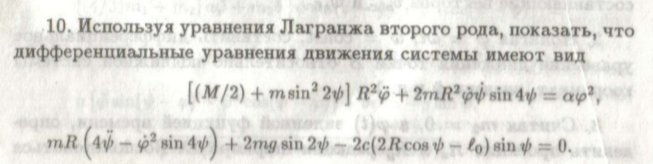
Если система консервативна, то можно упростить вид уравнения, используя функцию Лагранжа

𝐿=𝑇−𝛱.

Тогда уравнение Лагранжа второго рода для консервативной системы будет выглядеть следующим образом:

()− =0.

Рассмотрим вывод всех уравнений, которые нам необходимы для решения данной задачи.

  
  
  
  
Как мы можем убедиться, выведенные собственноручно формулы полностью совпадают с формулами, приведенными задании:

**Исходный код программы:**

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from scipy.integrate import odeint

import sympy as sp

import math

def formY(y, t, fV, fOm):

y1, y2, y3, y4 = y

dYdT = [y3, y4, fV(y1, y2, y3, y4), fOm(y1, y2, y3, y4)]

return dYdT

# defining parameters

alpha = math.pi / 6

M = 1

m = 2

R = 0.3

c = 20

l0 = 1

g = 9.81

# start

y0 = [0, np.pi / 15, -0.5, 0]

# defining t as a symbol (it will be the independent variable)

t = sp.Symbol('t')

# phi, ksi, v\_phi = d(phi) / dt and v\_psi = d(psi) / dt as functions of t

phi = sp.Function('phi')(t)

psi = sp.Function('psi')(t)

v\_phi = sp.Function('v\_phi')(t)

v\_psi = sp.Function('v\_psi')(t)

length = 2 \* R \* sp.cos(phi)

# 1 defining the kinetic energy

TT1 = M \* R \*\* 2 \* v\_phi \*\* 2 / 2

V1 = 2 \* v\_psi \* R

V2 = v\_phi \* R \* sp.sin(2 \* psi)

Vr2 = V1 \*\* 2 + V2 \*\* 2

TT2 = m \* Vr2 / 2

TT = TT1 + TT2

# 2 defining the potential energy

Pi1 = 2 \* R \* m \* g \* sp.sin(psi) \*\* 2

Pi2 = (c \* (length - l0) \*\* 2) / 2

Pi = Pi1 + Pi2

# 3 Not potential force

M = alpha \* phi \*\* 2

# Lagrange function

L = TT - Pi

# equations

ur1 = sp.diff(sp.diff(L, v\_phi), t) - sp.diff(L, phi) - M

ur2 = sp.diff(sp.diff(L, v\_psi), t) - sp.diff(L, psi)

a11 = ur1.coeff(sp.diff(v\_phi, t), 1)

a12 = ur1.coeff(sp.diff(v\_psi, t), 1)

a21 = ur2.coeff(sp.diff(v\_phi, t), 1)

a22 = ur2.coeff(sp.diff(v\_psi, t), 1)

b1 = -(ur1.coeff(sp.diff(v\_phi, t), 0)).coeff(sp.diff(v\_psi, t), 0).subs(

[(sp.diff(phi, t), v\_phi), (sp.diff(psi, t), v\_psi)])

b2 = -(ur2.coeff(sp.diff(v\_phi, t), 0)).coeff(sp.diff(v\_psi, t), 0).subs(

[(sp.diff(phi, t), v\_phi), (sp.diff(psi, t), v\_psi)])

detA = a11 \* a22 - a12 \* a21

detA1 = b1 \* a22 - b2 \* a21

detA2 = a11 \* b2 - b1 \* a21

# Constructing the system of differential equations

T = np.linspace(0, 25, 2000)

fv\_phi = sp.lambdify([phi, psi, v\_phi, v\_psi], detA1 / detA, "numpy")

fv\_psi = sp.lambdify([phi, psi, v\_phi, v\_psi], detA2 / detA, "numpy")

sol = odeint(formY, y0, T, args=(fv\_phi, fv\_psi))

t = sp.Symbol('t')

phi = sol[:, 0]

psi = sol[:, 1]

v\_phi = sol[:, 2]

v\_psi = sol[:, 3]

fig = plt.figure()

ax = fig.add\_subplot(projection='3d')

ax.set(xlim=[-8, 8], ylim=[-8, 8], zlim=[0, 8])

y = 3.5 \* np.cos(psi)

z = 3.5 \* np.sin(psi)

Z\_PointCentral = 8.5

Y\_PointCentral = 0

X\_PointCentral = 0

PointCentral = ax.plot(X\_PointCentral, Y\_PointCentral, Z\_PointCentral, color='blue', marker='o', markeredgewidth=1)[0]

Z\_PointM = 5

X\_PointM = 0

Y\_PointM = 0

circle = ax.plot(y \* np.cos(phi), y \* np.sin(phi), z + 5, linewidth=5, color='green', alpha=.2)[0]

PointM = ax.plot(X\_PointM, Y\_PointM, Z\_PointM, color='orange', marker='o', markeredgewidth=4)[0]

def get\_spring\_line(coils, diameter, start, end):

x = np.linspace(start[0], end[0], coils \* 2)

y = np.linspace(start[1], end[1], coils \* 2)

z = np.linspace(start[2], end[2], coils \* 2)

for i in range(1, len(z) - 1):

z[i] = z[i] + diameter \* 1 \* (-1) \*\* i

return np.array([x, y, z], dtype=object)

ax.plot([0, 0], [0, 0], [0, 10.80], linewidth=2, color='black', alpha=.8) # stick

# spring

spring\_xyz = get\_spring\_line(30, 0.1, [0, 0, 8.5], [1, 3, 2])

spring = ax.plot(spring\_xyz[0], spring\_xyz[1], spring\_xyz[2], linewidth=2, color='black')[0]

def animation(i):

circle.set\_data\_3d(y \* np.cos(psi[i]), y \* np.sin(psi[i]), z + 5)

current\_z = (Z\_PointM + 3.5 \* np.cos(psi[i]))

if current\_z >= 5:

current\_z = current\_z - 2 \* (current\_z - 5)

new\_x = (X\_PointM + 3.5 \* np.sin(psi[i])) \* np.cos(psi[i])

new\_y = (Y\_PointM + 3.5 \* np.sin(psi[i])) \* np.sin(psi[i])

new\_z = current\_z

PointM.set\_data\_3d(new\_x, new\_y, new\_z)

sprint\_coordinates = get\_spring\_line(30, 0.2, [0, 0, 8.5], [new\_x, new\_y, new\_z])

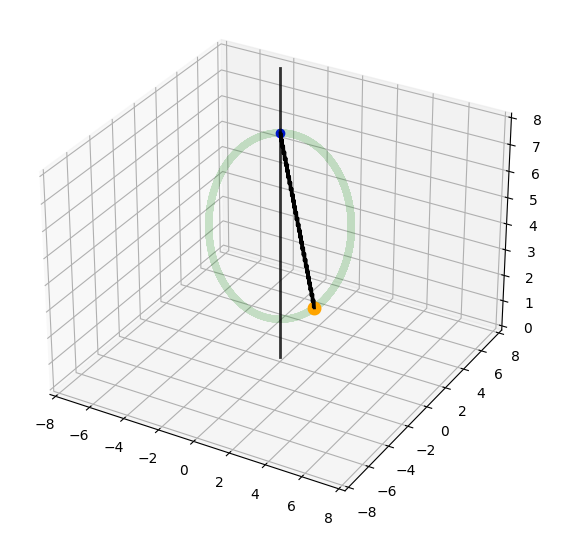
spring.set\_data\_3d(sprint\_coordinates[0], sprint\_coordinates[1], sprint\_coordinates[2])

return [circle, PointM, spring]

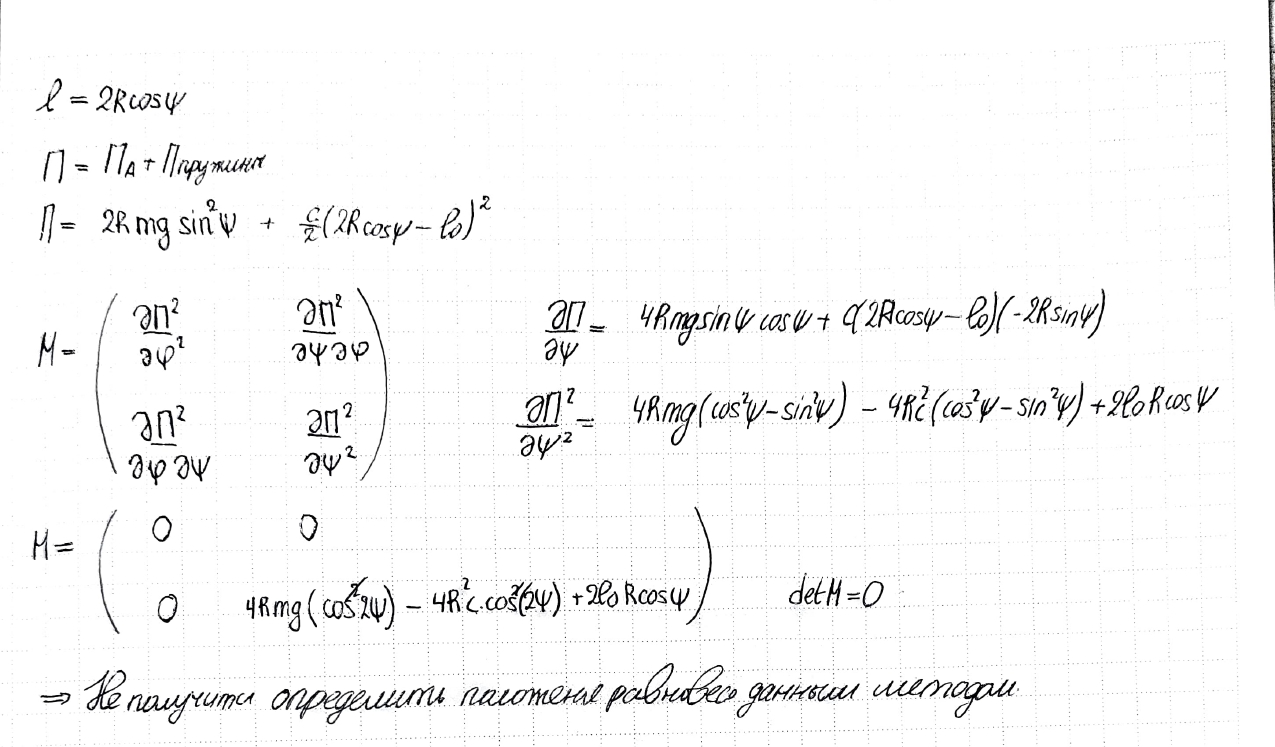
anima = FuncAnimation(fig, animation, frames=500, interval=20)

plt.show()

Результат работы программы:



**Лабораторная работа 4**

Задание: Построить анимацию движения системы, а также графики законов движения системы. Исследовать на устойчивость. Показать правильность работы своей механической системы.  
  
**Вывод положений равновесия и исследование их на устойчивость:  
  
  
  
  
Результаты работы программы:**

Выведем полученные графики работы программы:

1)

alpha = math.pi / 6

M = 1

m = 0.1

R = 0.3

c = 20

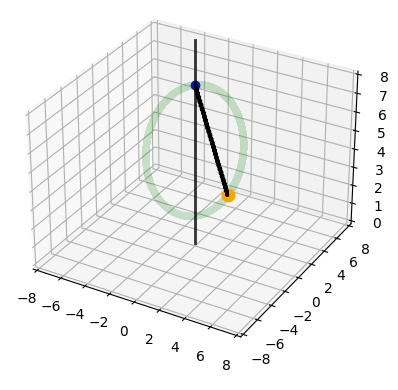
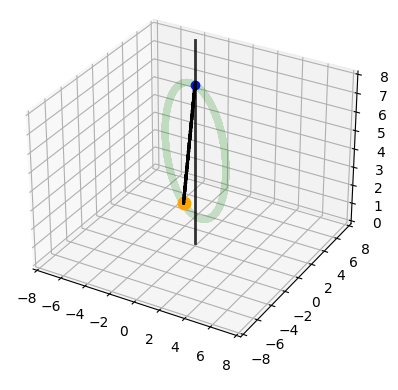
l0 = 0.2

g = 9.81

y0 = [0, np.pi/6, -0.5, 0] - система находится в положении заданном задачей (изначальные данные в лабораторной работе №3).

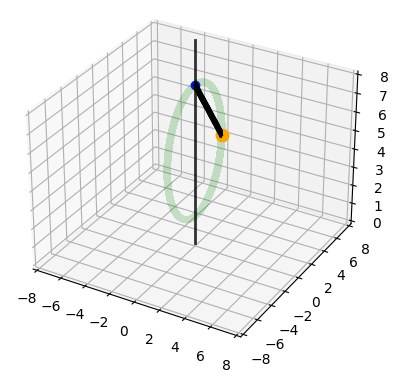
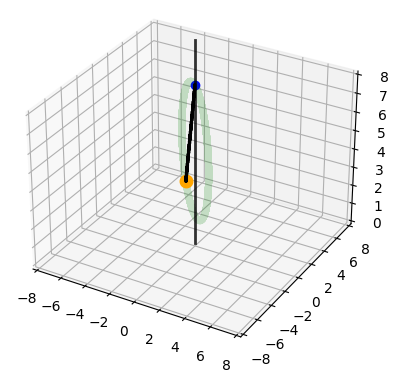
Результаты:

При заданных параметрах точка поднимается на относительно небольшую высоту. l0 = 0.2 - это довольно большое значение относительно радиуса, поэтому точка поднимается не быстро. При этом на кольцо по условию задачи действуют внешние силы.



2)

alpha = math.pi / 6  
M = 1  
m = 0.1  
R = 0.3  
c = 20  
l0 = 0  
g = 9.81  
y0 = [0, np.pi/6, -0.5, 0] - система находится так же в исходном положении.

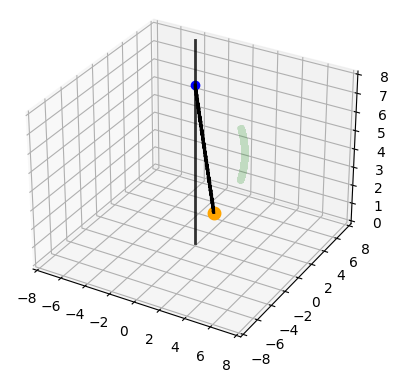
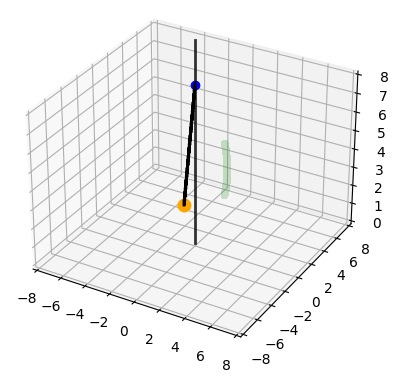


Результаты: шарик стал двигаться быстрее и подниматься по кольцу значительно выше. И это действительно так - ведь чем меньше длина пружины, тем, соответственно, выше потенциальная энергия пружины. Это видно из формулы для потенциальной энергии.

3)

alpha = math.pi / 8  
M = 1  
m = 2  
R = 0.3  
c = 20  
l0 = 0.5  
g = 9.81  
y0 = [0, np.pi/8, -0.5, 0]

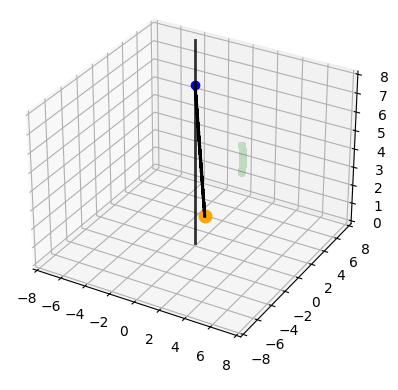
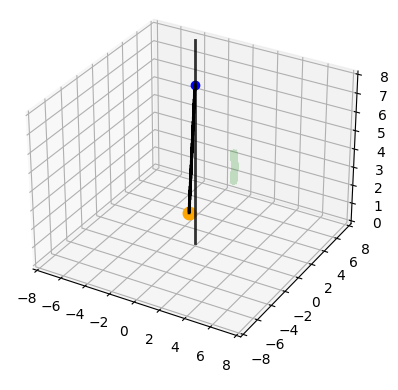
Угол отклонения — pi/8, m > M, R < l0 < 2R



Результат: Так как в этот раз l0 у нас больше радиуса, но меньше диаметра, масса шарика больше массы кольца, и при этом отклонение производится на меньший угол, то шарик поднимется на еще меньшую высоту. Чем больше длина нерастянутой пружины, тем меньше тяга вверх.

4)

alpha = math.pi / 15  
M = 1  
m = 2  
R = 0.3  
c = 20  
l0 = 1  
g = 9.81  
y0 = [0, np.pi/15, -0.5, 0] - теперь отклонение производится на угол pi/15,

M < m, 2R < l0  


Результат: Шарик практически не поднимается. Оно и понятно - масса больше массы кольца, а длина пружины больше диаметра. Именно поэтому шарик почти недвижим. Чем больше длина нерастянутой пружины, тем меньше тяга вверх.

**Вывод:**

В ходе этой лабораторной работы была создана анимация движения системы, было выведено уравнение Лагранжа второго порядка для данной системы, были найдены положения равновесия и проверены на устойчивость. Все эти четыре лабораторные работы помогли мне углубить изучение этой темы и овладеть нюансами работы со специальными библиотеками Python, с помощью которых можно создавать различные анимации, а также расширить мои знания в области теоретической механики.