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-20

Морозов Артем Борисович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

Доцент каф. 802, Чекина Е.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

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

Москва, 2021

**Лабораторная работа 3.**

**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 производится по следующему алгоритму:

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.

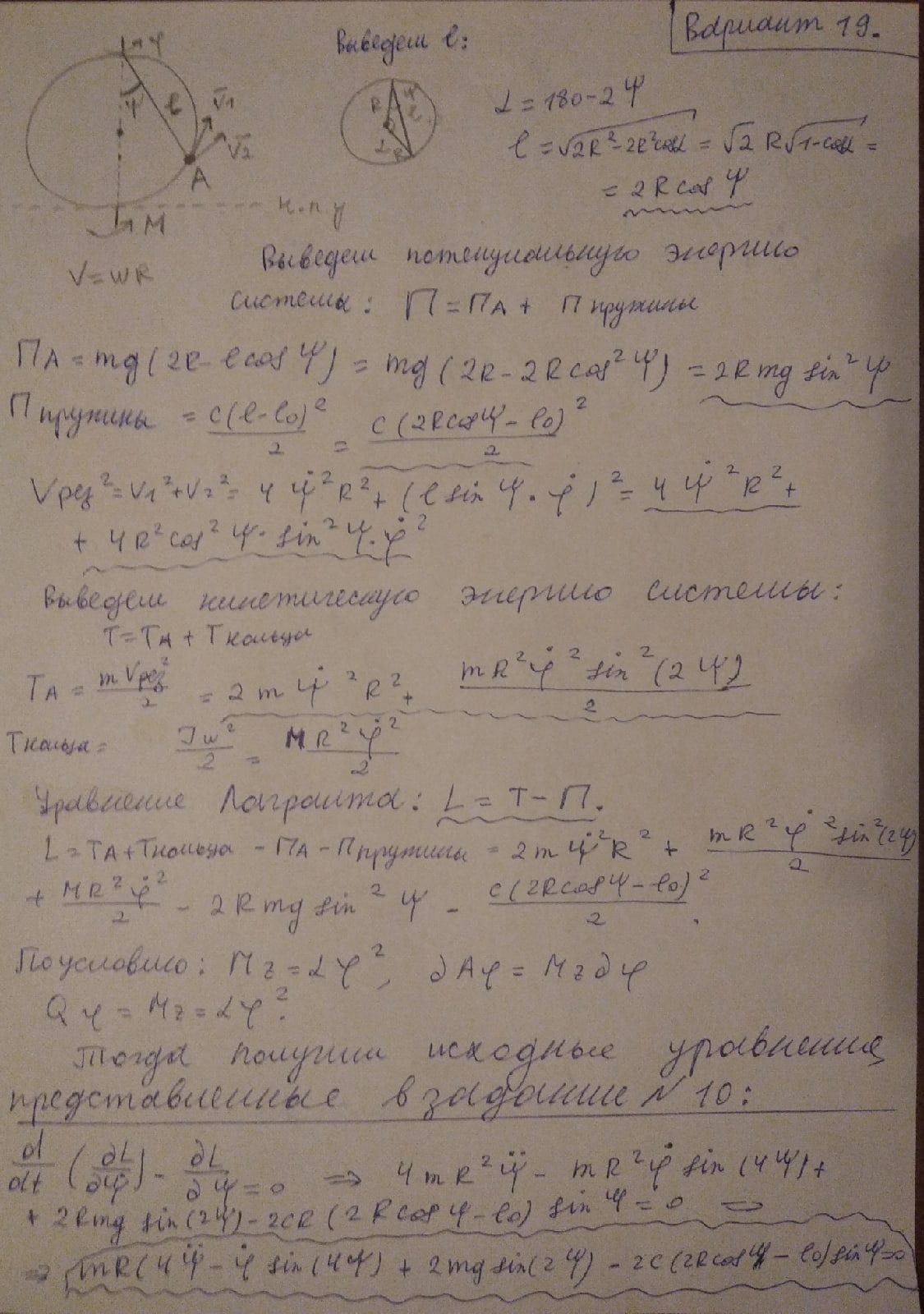
Если система консервативна, то можно упростить вид уравнения, используя функцию Лагранжа

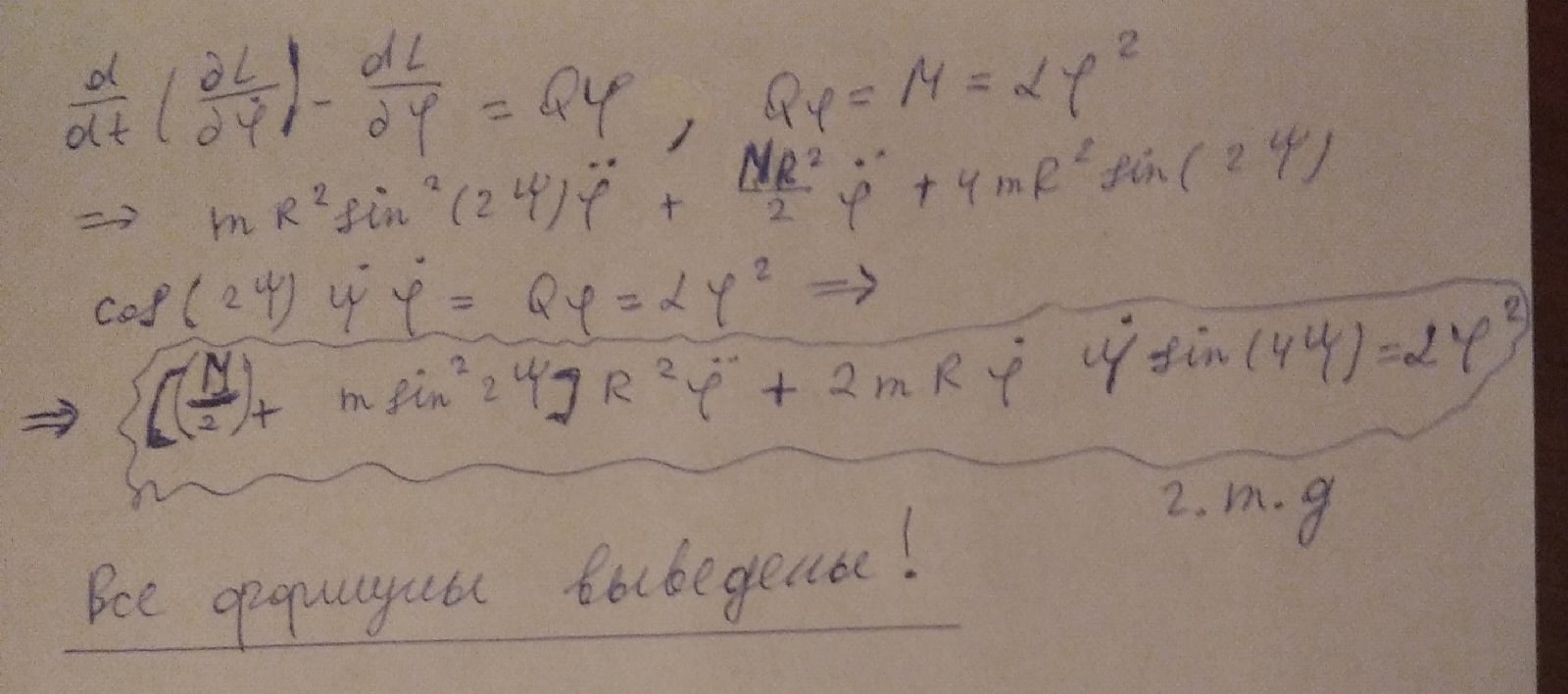
𝐿=𝑇−𝛱.

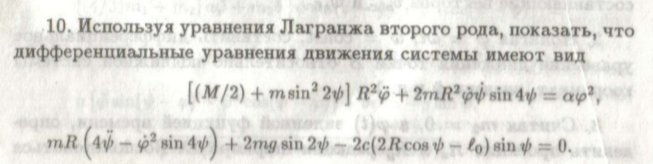
Тогда уравнение Лагранжа второго рода для консервативной системы будет выглядеть следующим образом:

()− =0.

Ниже привожу вывод всех нужных уравнений, проделанный собственноручно:





Как мы можем убедиться, выведенные собственноручно формулы полностью совпадают с формулами, приведенными в самой методичке:  
  
  
  
  
  
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

Steps = 1000

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 = 0.1

R = 0.3

c = 20

l0 = 0.2

g = 9.81

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

t = sp.Symbol('t')

# defining phi, ksi, Vphi=dphi/dt and Vpsi=dpsi/dt as functions of 't'

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

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

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

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

l = 2 \* R \* sp.cos(psi) # длина пружины

#1 defining the kinetic energy

TT1 = M \* R\*\*2 \* Vphi\*\*2 / 2

V1 = 2\*Vpsi \* R

V2 = Vphi \* 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 \* (l - 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,Vphi),t)-sp.diff(L,phi) - M

ur2 = sp.diff(sp.diff(L,Vpsi),t)-sp.diff(L,psi)

# isolating second derivatives(dV/dt and dom/dt) using Kramer's method

a11 = ur1.coeff(sp.diff(Vphi,t),1)

a12 = ur1.coeff(sp.diff(Vpsi,t),1)

a21 = ur2.coeff(sp.diff(Vphi,t),1)

a22 = ur2.coeff(sp.diff(Vpsi,t),1)

b1 = -(ur1.coeff(sp.diff(Vphi,t),0)).coeff(sp.diff(Vpsi,t),0).subs([(sp.diff(phi,t),Vphi), (sp.diff(psi,t), Vpsi)])

b2 = -(ur2.coeff(sp.diff(Vphi,t),0)).coeff(sp.diff(Vpsi,t),0).subs([(sp.diff(phi,t),Vphi), (sp.diff(psi,t), Vpsi)])

detA = a11\*a22-a12\*a21

detA1 = b1\*a22-b2\*a21

detA2 = a11\*b2-b1\*a21

dVdt = detA1/detA

domdt = detA2/detA

countOfFrames = 2000

# Constructing the system of differential equations

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

fVphi = sp.lambdify([phi,psi,Vphi,Vpsi], dVdt, "numpy")

fVpsi = sp.lambdify([phi,psi,Vphi,Vpsi], domdt, "numpy")

y0 = [0, np.pi/6, -0.5, 0]

sol = odeint(formY, y0, T, args = (fVphi, fVpsi))

t = sp.Symbol('t')

phi = sol[:,0]

psi = sol[:,1]

Vphi = sol[:,2]

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

PointM = ax.plot(X\_PointM, Y\_PointM, Z\_PointM, color='black', 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

ax.plot([-2.25, -1.25], [-0.25, 0.25], [0.3, 0.3], linewidth=2, color='black', alpha=.8)

ax.plot([-1.25, -1.25], [0.23, 0.25], [-0.3, 0.3], linewidth=2, color='black', alpha=.8)

ax.plot([-1.25, 0.5], [0.25, 1.25], [-0.4, -0.4], linewidth=2, color='black', alpha=.8)

ax.plot([0.5, 0.5], [1.23, 1.25], [-0.3, 0.3], linewidth=2, color='black', alpha=.8)

ax.plot([0.5, 1.40], [1.25, 1.80], [0.3, 0.3], linewidth=2, color='black', alpha=.8)

ax.plot([-0.25, -0.25], [0, 0], [9.75, 10.5], linewidth=1.5, color='black', alpha=.8)

ax.plot([0.25, 0.25], [0.15, 0.15], [9.77, 10.52], linewidth=1.5, color='black', alpha=.8) #right

ax.plot([0.25, 0.5], [0.15, 0.3], [10.52, 10.35], linewidth=1.5, color='black', alpha=.8)

ax.plot([0.25, 0.5], [0.15, 0.3], [10.35, 10.17], linewidth=1.5, color='black', alpha=.8)

ax.plot([0.25, 0.5], [0.15, 0.3], [10.17, 10], linewidth=1.5, color='black', alpha=.8)

ax.plot([0.25, 0.5], [0.15, 0.3], [10, 9.83], linewidth=1.5, color='black', alpha=.8)

ax.plot([-0.25, -0.5], [0, -0.15], [10.5, 10.33], linewidth=1.5, color='black', alpha=.8) #left

ax.plot([-0.25, -0.5], [0, -0.15], [10.33, 10.16], linewidth=1.5, color='black', alpha=.8)

ax.plot([-0.25, -0.5], [0, -0.15], [10.16, 9.99], linewidth=1.5, color='black', alpha=.8)

ax.plot([-0.25, -0.5], [0, -0.15], [9.99, 9.82], linewidth=1.5, color='black', alpha=.8)

#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 Kino(i):

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

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

if curz >= 5:

delta = curz - 5

curz = curz - 2 \* delta

PointM.set\_data\_3d((X\_PointM + 3.5 \* np.sin(psi[i])) \* np.cos(psi[i]),

(Y\_PointM + 3.5 \* np.sin(psi[i])) \* np.sin(psi[i]), curz)

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

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

newz = curz

spring\_xyz = get\_spring\_line(30, 0.2, [0, 0, 8.5], [newx, newy, newz])

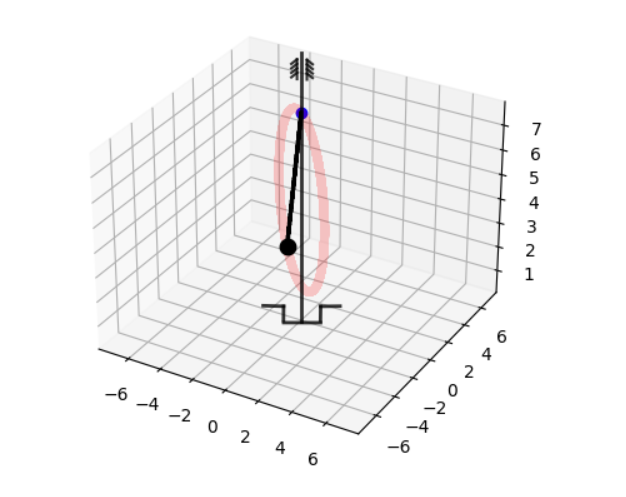
spring.set\_data\_3d(spring\_xyz[0], spring\_xyz[1], spring\_xyz[2])

return [Circle\_cir, PointM, spring]

anima = FuncAnimation(fig, Kino, frames=500, interval=80)

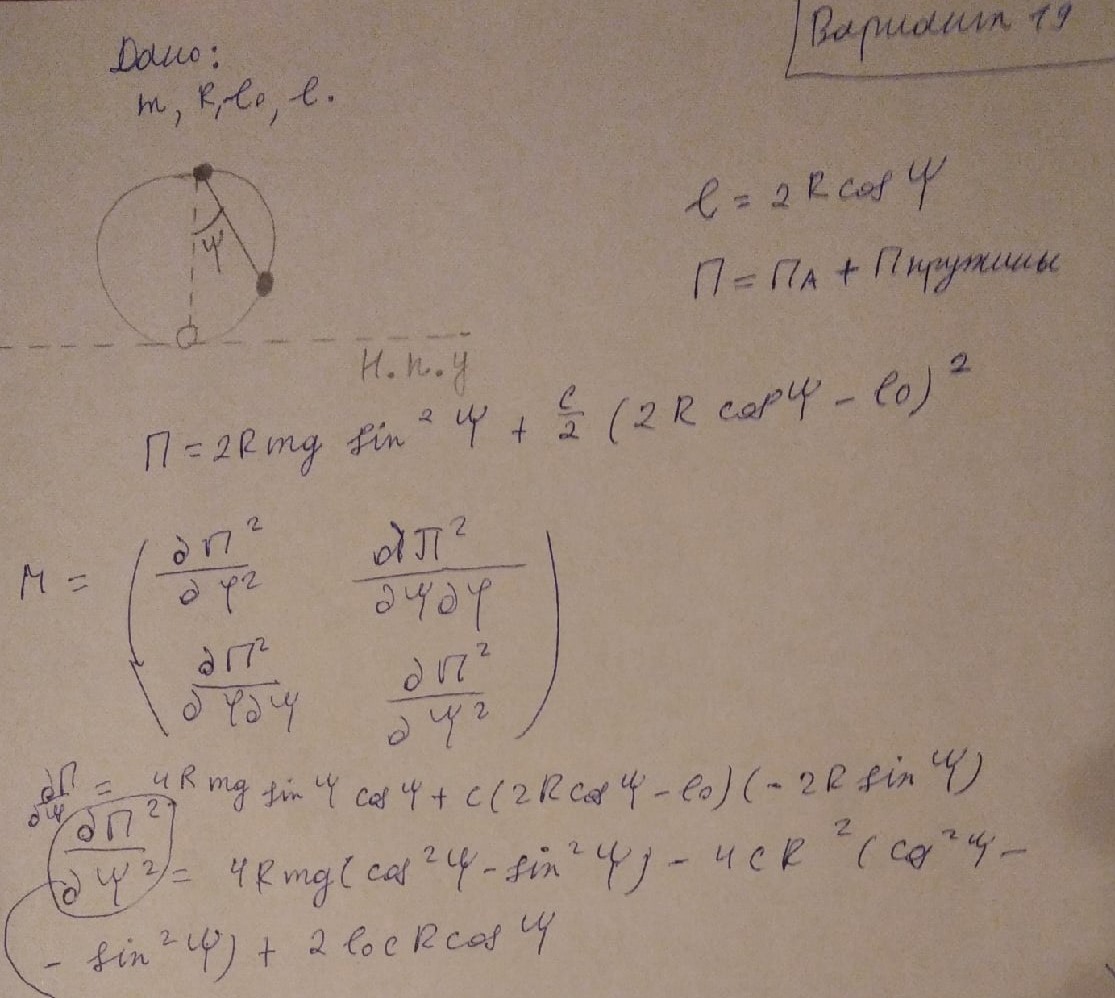
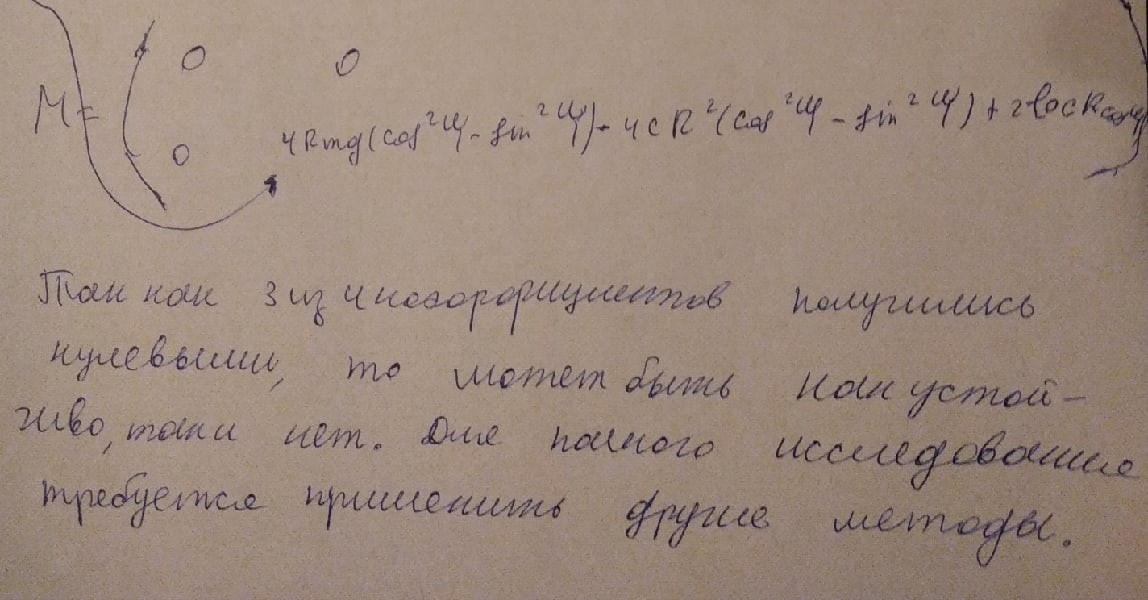
plt.show()

Результат работы программы:



**Лабораторная работа 4**

**Симуляция движения системы с двумя степенями свободы**

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:**

1. Выведем полученные графики работы программы:

alpha = math.pi / 6

M = 1

m = 0.1

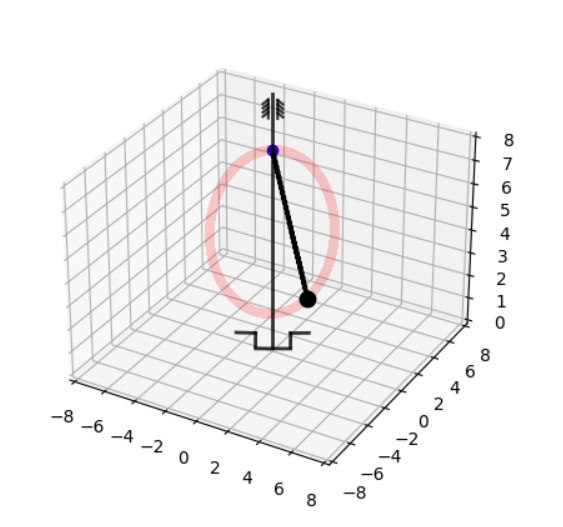
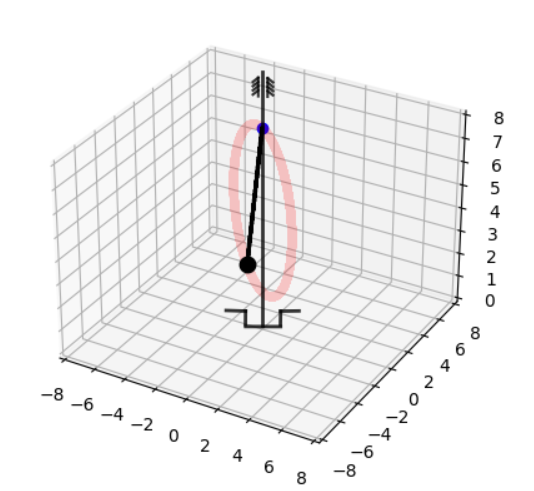
R = 0.3

c = 20

l0 = 0.2

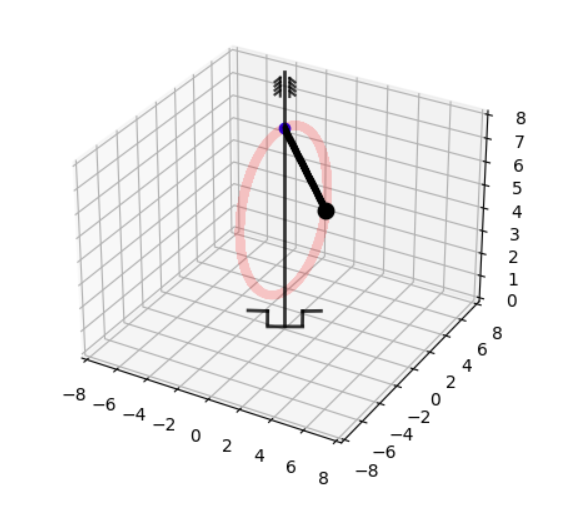
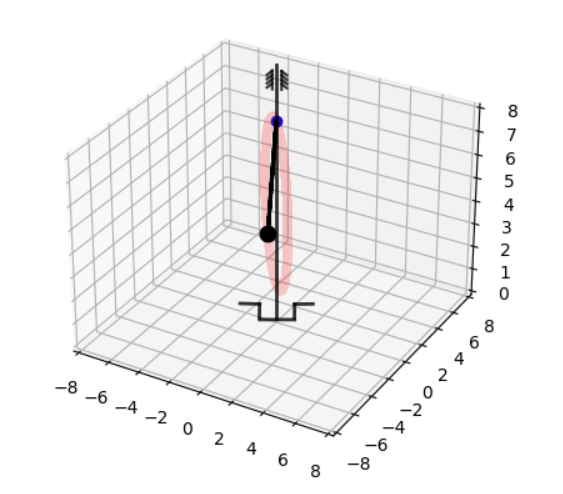
g = 9.81

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



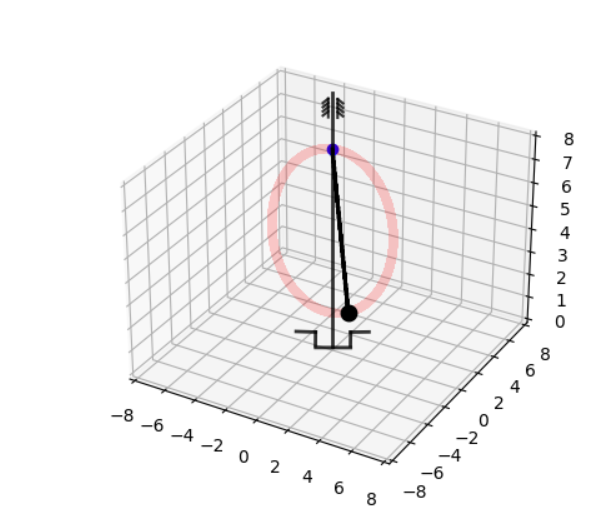
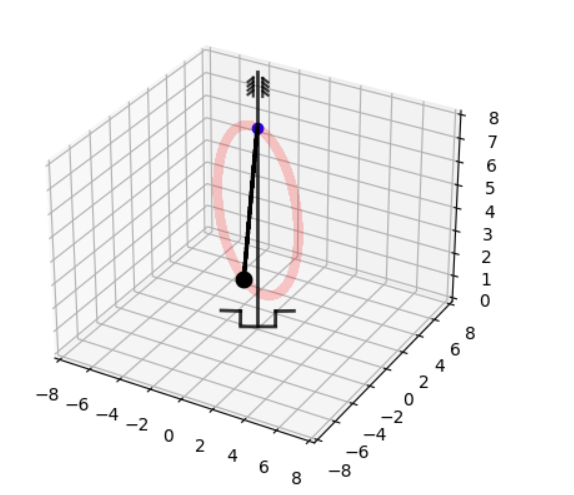
Результаты: Так как на точку действует сила тяжести, то она поднимается на лишь “среднюю” высоту. l0 = 0.2 - это довольно большое значение относительно радиуса, поэтому точка поднимается не быстро. При этом на кольцо по условию задачи действуют внешние силы.

2) Выведем полученные графики работы программы:  
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.



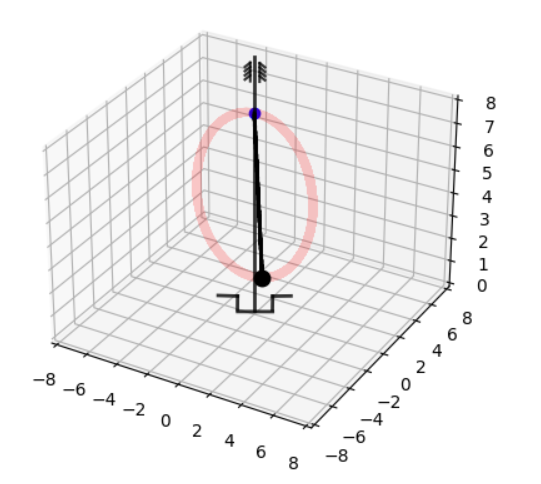
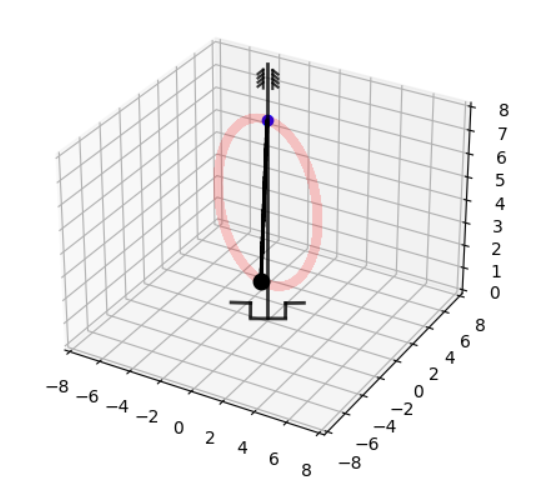
Результаты: При неизменных данных, кроме l0, произошли изменения: наш шарик стал двигаться быстрее и подниматься по кольцу значительно выше. И это действительно так - ведь чем меньше длина пружины, тем, соответственно, выше потенциальная энергия пружины. Это видно из формулы для потенциальной энергии. Именно поэтому наш шарик и может забраться на бОльшую высоту.

1. Выведем полученные графики работы программы:

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] - теперь отклонение производится на угол pi/8, масса шарика стала больше массы кольца, длина пружины больше радиуса, но меньше диаметра.  
  


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. Так оно и получилось. Чем больше длина нерастянутой пружины, тем меньше тяга вверх.

1. Выведем полученные графики работы программы:

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, масса шарика все так же больше массы кольца, длина пружины больше диаметра.  
  


Результат: В итоге при таких условиях шарик практически не поднимается. Оно и понятно - масса больше массы кольца, а длина пружины больше диаметра. Именно поэтому шарик почти недвижим. Опять же, чем больше длина нерастянутой пружины, тем меньше тяга вверх.

**Вывод:**

В результате данной лабораторной работы была составлена анимация движения системы, было выведено уравнение Лагранжа второго рода для заданной системы, были найдены положения равновесия и проверены на устойчивость.   
Все четыре лабораторные работы помогли мне глубже изучить данную тему и помогли освоить тонкости работы со специальными библиотеками языка Python, посредством которых можно создавать различные анимации, а также обогатили мои познания в сфере теоретической механики.