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 №21**

Выполнил(а) студент группы М8О-207Б-22

Кострюков Евгений Сергеевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

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

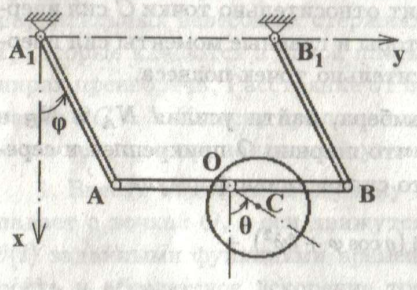
подпись, дата

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

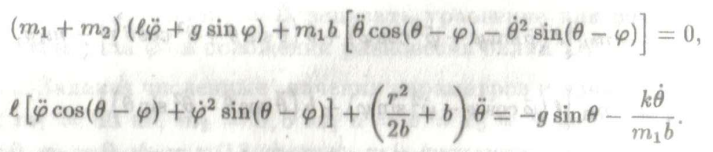
Москва, 2023

*Задание:* проинтегрировать систему дифференциальных уравнений движения системы с двумя степенями свободы с помощью средств Python. Построить анимацию движения системы, а также графики законов движения системы и указанных в задании реакций для разных случаев системы.

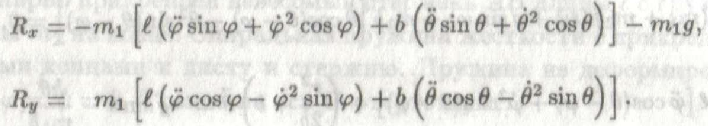
*1) Система:*



2) Уравнения движения системы:



*3) Проекции реакции шарнира:*

**

*4) Код программы:*

import math

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import odeint

from matplotlib.animation import FuncAnimation

# Функция составления дифференциальных уравнений

def odesys(y, t, g, m1, m2, r, l, b, k):

dy = np.zeros(4)

dy[0] = y[2] # phi с точкой

dy[1] = y[3] # thetta с точкой

a11 = (m1+m2)\*l

a12 = m1\*b\*np.cos(y[1]-y[0])

a21 = l\*np.cos(y[1]-y[0])

a22 = ((r\*\*2)/(2\*b))+b

b1 = -(m1+m2)\*g\*np.sin(y[0])+m1\*b\*((y[3]\*\*2)\*np.sin(y[1]-y[0]))

b2 = -g\*np.sin(y[1])-(k\*y[3])/(m1\*b)-l\*(y[2]\*\*2)\*np.sin(y[1]-y[0])

dy[2] = (b1\*a22 - b2\*a12)/(a11\*a22 - a12\*a21)

dy[3] = (b2\*a11 - b1\*a21)/(a11\*a22 - a12\*a21)

return dy

Steps = 1001

t\_fin = 20

t = np.linspace(0, t\_fin, Steps) # Равномерно распределяем Steps элементов на участке (0; t\_fin)

# Характеристики различных предметов системы

g = 9.81

m1 = 5 # Масса диска

m2 = 5 # Масса прикреплённого стержня

r = 0.25 # Радиус диска

l = 1 # Длины стержней

b = 0.125 # Расстояние от шарнира O до центра масс диска

k = 10 # Константа, пропорциональная угловой скорости

# Начальные положения

phi\_0 = math.pi / 2

thetta\_0 = math.pi / 2

dphi\_0 = 0

dthetta\_0 = 0

y0 = [phi\_0, thetta\_0, dphi\_0, dthetta\_0]

# Интегрируем систему, которую создаёт odeint

Y = odeint(odesys, y0, t, args=(g, m1, m2, r, l, b, k))

phi = Y[:,0]

thetta = Y[:,1]

# Для графиков реакции

dphi = Y[:,2]

dthetta = Y[:,3]

ddphi = [odesys(y, t, g, m1, m2, r, l, b, k)[2] for y,t in zip(Y, t)]

ddthetta = [odesys(y, t, g, m1, m2, r, l, b, k)[3] for y,t in zip(Y, t)]

Rx = -m1\*(l\*(ddphi\*np.sin(phi)+dphi\*\*2\*np.cos(phi))+b\*(ddthetta\*np.sin(thetta)+dthetta\*\*2\*np.cos(thetta)))-m1\*g

Ry = m1\*(l\*(ddphi\*np.cos(phi)-dphi\*\*2\*np.sin(phi))+b\*(ddthetta\*np.cos(thetta)-dthetta\*\*2\*np.sin(thetta)))

# Создаём дополнительное окно для отриовки графиков зависимости обобщённых координат от времени

fig\_for\_graphs = plt.figure(figsize=[8, 7])

# phi(t)

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 2, 1)

ax\_for\_graphs.plot(t, phi, color='Blue')

ax\_for\_graphs.set\_title("phi(t)")

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True) # Добавляем сетку

# thetta(t)

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 2, 3)

ax\_for\_graphs.plot(t, thetta, color='Red')

ax\_for\_graphs.set\_title("thetta(t)")

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True)

# Rx

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 2, 2)

ax\_for\_graphs.plot(t, Rx, color='Black')

ax\_for\_graphs.set\_title("Rx")

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True)

# Ry

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 2, 4)

ax\_for\_graphs.plot(t, Ry, color='Gray')

ax\_for\_graphs.set\_title("Ry")

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True)

# Координаты подвижных частей стержней

# Конец стержня l

A\_X = l \* np.sin(phi)

A\_Y = - l \* np.cos(phi)

# Конец стержня l

B\_X = l \* np.sin(phi) + l

B\_Y = - l \* np.cos(phi)

# Центр масс диска

C\_X = A\_X + l / 2 + b \* np.sin(thetta)

C\_Y = A\_Y - b \* np.cos(thetta)

fig = plt.figure(figsize=[8, 7]) # Создаём окно для отрисовки

ax = fig.add\_subplot(1, 1, 1) # Добавляем ячейку (окно) для отрисовки

ax.axis('equal') # Делаем единичные отрезки осей равными

ax.set(xlim=[-3, 4], ylim=[-1.5, 1.5]) # Пределы по осям

# Координаты поверхности, на которой закрепелны стержни

X\_Ground = [-0.5, 0, 1.5]

Y\_Ground = [0.1, 0.1, 0.1]

# plot() рисует прямые линии, соединяя указанные точки

ax.plot(X\_Ground, Y\_Ground, color='black', linewidth=3) # Отрисовка поверхности

# Опоры, на которых вращаются стержни

Drawed\_Q1A1 = ax.plot([-0.1, 0], [0.1, 0], color='black')

Drawed\_Q2A1 = ax.plot([0.1, 0], [0.1, 0], color='black')

Drawed\_E1A1 = ax.plot([l - 0.1, l], [0.1, 0], color='black')

Drawed\_E2A1 = ax.plot([l + 0.1, l], [0.1, 0], color='black')

# Параметры, куда помещаем объект отрисовки чего-либо в начальный момент времени

# Стержени

Drawed\_A1A = ax.plot([0, A\_X[0]], [0, A\_Y[0]], color='green')[0]

Drawed\_B1B = ax.plot([1, B\_X[0]], [0, B\_Y[0]], color='green')[0]

Drawed\_AB = ax.plot([A\_X[0], B\_X[0]], [A\_Y[0], B\_Y[0]], color='blue')[0]

# Радиус-вектор

Drawed\_OC = ax.plot([A\_X[0] + l / 2, C\_X[0]], [A\_Y[0], C\_Y[0]], color='black')[0]

# Отрисовываем точки для стержней

Point\_A1 = ax.plot(0, 0, marker='o', color='black')

Point\_B1 = ax.plot(1, 0, marker='o', color='black')

Point\_A = ax.plot(A\_X[0], A\_Y[0], marker='o', color='green')[0]

Point\_B = ax.plot(B\_X[0], B\_Y[0], marker='o', color='green')[0]

Point\_O = ax.plot([A\_X[0] + l / 2], A\_Y[0], marker='o', color='black')[0]

Point\_C = ax.plot(C\_X[0], C\_Y[0], marker='o', markersize=70, markerfacecolor='none', markeredgewidth=2, color='red')[0]

# Обновление текущего кадра

def anima(i):

# Изменяем нулевые элементы на i-ые:

Drawed\_A1A.set\_data([0, A\_X[i]], [0, A\_Y[i]])

Drawed\_B1B.set\_data([1, B\_X[i]], [0, B\_Y[i]])

Drawed\_AB.set\_data([A\_X[i], B\_X[i]], [A\_Y[i], B\_Y[i]])

Point\_A.set\_data(A\_X[i], A\_Y[i])

Point\_B.set\_data(B\_X[i], B\_Y[i])

Point\_O.set\_data([A\_X[i] + l / 2], A\_Y[i])

Point\_C.set\_data(C\_X[i], C\_Y[i])

Drawed\_OC.set\_data([A\_X[i] + l / 2, C\_X[i]], [A\_Y[i], C\_Y[i]])

return [Point\_A, Point\_B, Point\_O, Point\_C, Drawed\_A1A, Drawed\_B1B, Drawed\_AB, Drawed\_OC]

anim = FuncAnimation(fig, anima, frames=len(t), interval=10, repeat=False) # Создаём анимацию

plt.show() # Отрисовываем

*5) Пояснение к коду:*

Сперва определяется функция odesys, которая описывает дифференциальные уравнения системы. Входные параметры функции: y - вектор обобщенных координат, t - время, g, m1, m2, r, l, b, k - параметры системы. Функция возвращает вектор производных обобщенных координат.

Затем задаются параметры системы, начальные положения обобщенных координат и интегрируется система дифференциальных уравнений с помощью функции odeint. Результаты интегрирования записываются в массив Y, который содержит значения обобщенных координат и их производных в зависимости от времени.

Далее происходит построение графиков зависимости обобщенных координат и их производных от времени с использованием библиотеки matplotlib.pyplot. Создается фигура fig\_for\_graphs и добавляются четыре подграфика с помощью функции add\_subplot. Затем на каждом подграфике вызывается функция plot, которая строит график, и устанавливаются заголовки и пределы по осям.

Далее определяются координаты различных частей системы для последующей визуализации. Создается новая фигура fig и добавляется одна ячейка для отрисовки с помощью функции add\_subplot. Затем на этой ячейке отображается поверхность, на которой закреплены стержни, а также опоры и подвижные части системы.

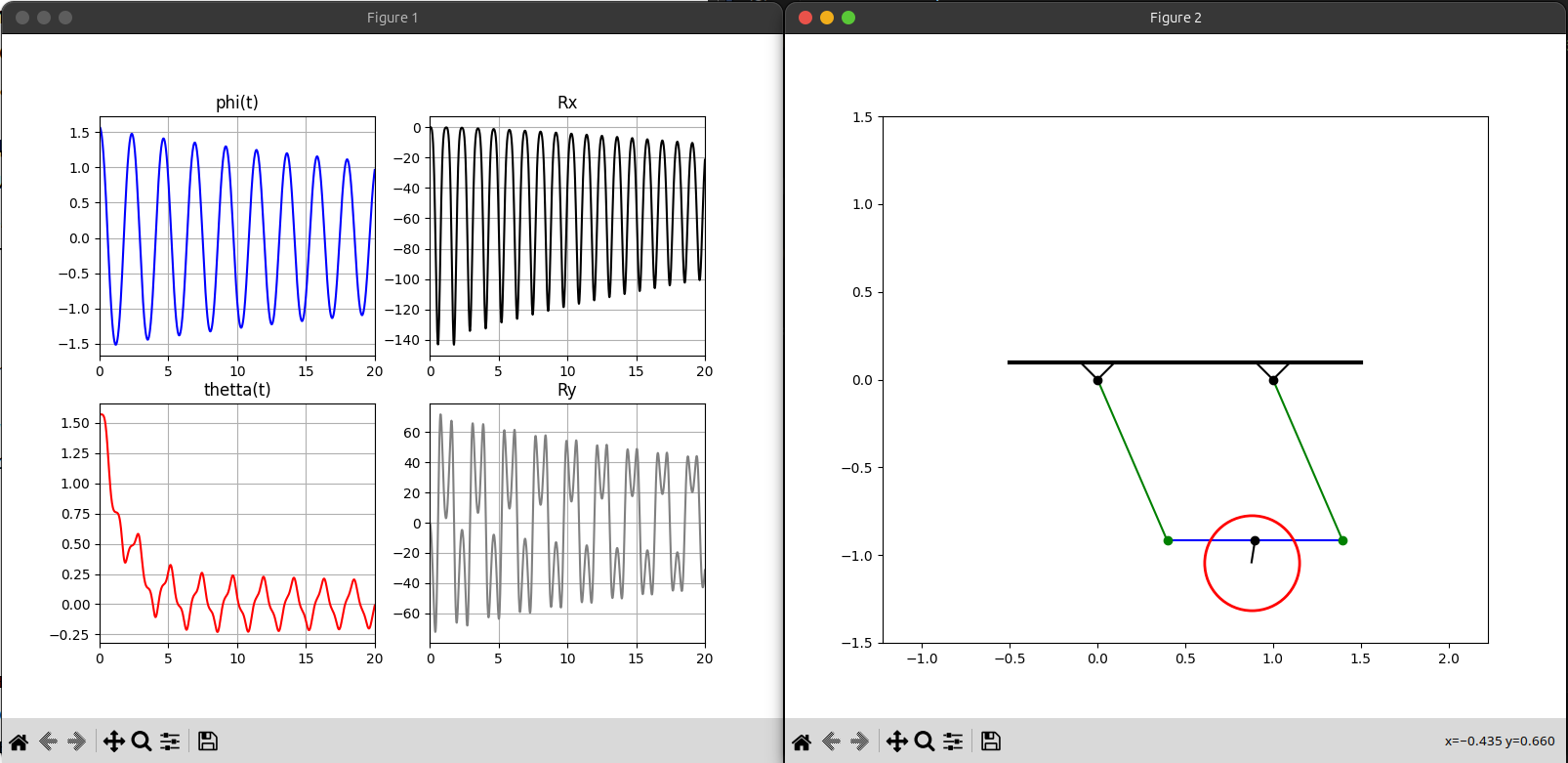
Далее определяется функция anima, которая обновляет текущий кадр анимации. Внутри функции происходит изменение координат и положений объектов системы в соответствии с текущим временным шагом i. Функция возвращает список объектов, которые нужно обновить на каждом кадре анимации.

Наконец, создается анимация с помощью функции FuncAnimation, где указывается фигура, функция обновления кадра anima, количество кадров, интервал между кадрами и параметр repeat, указывающий, нужно ли повторять анимацию после ее окончания. Затем вызывается функция show для отображения анимации.

*6) Результаты работы программы:*

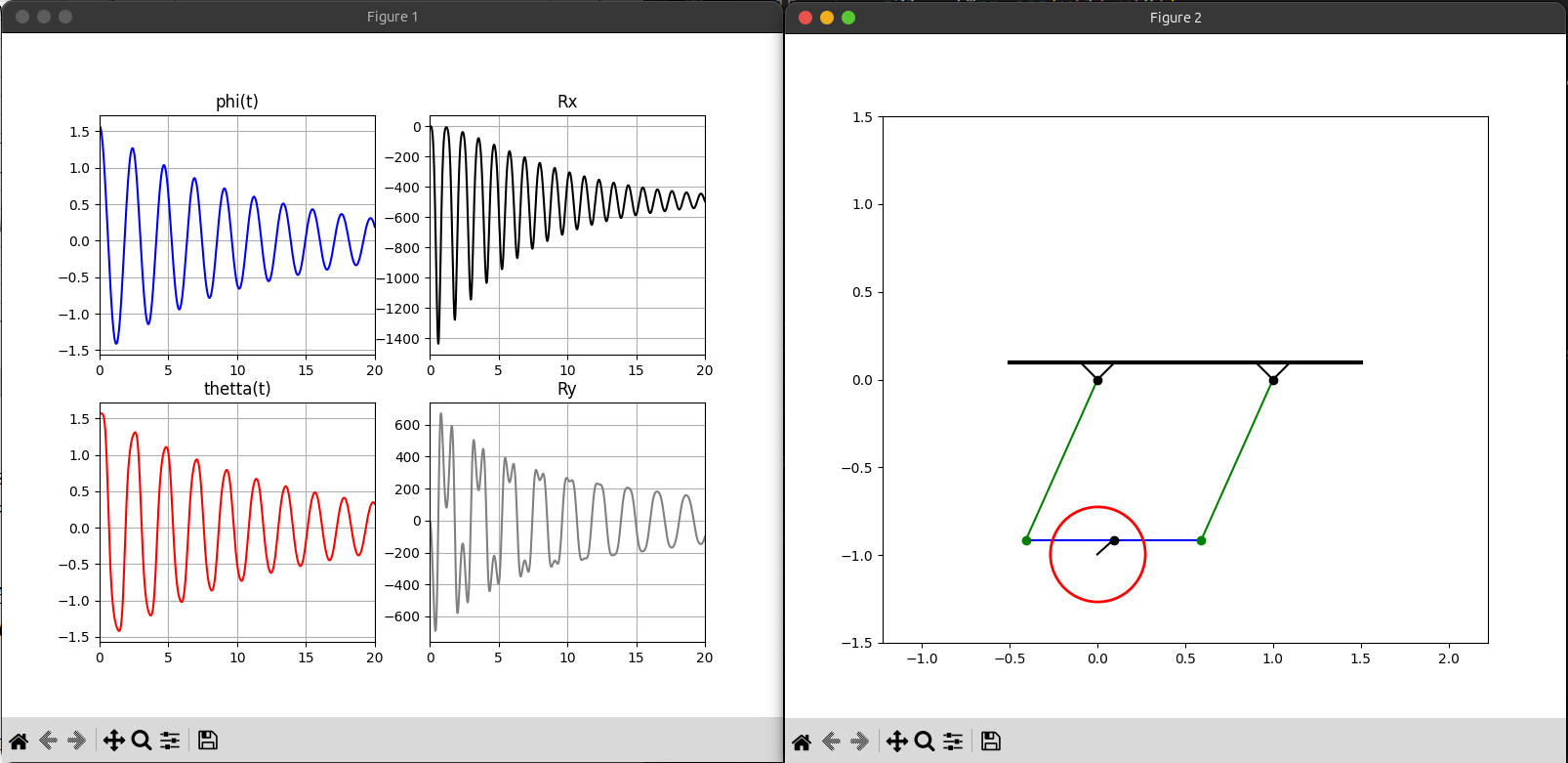
*1. g = 9.81; m1 = 5; m2 = 5; r = 0.25; l = 1; b = 0.125; k = 10; phi\_0 = math.pi / 2; thetta\_0 = math.pi / 2; dphi\_0 = 0; dthetta\_0 = 0; y0 = [phi\_0, thetta\_0, dphi\_0, dthetta\_0].*

*При данных значениях системы стержни совершают нормальные колебания, адекватно качаются, медленно останавливаясь, однако диск совершает не гармонические колебания.*

**

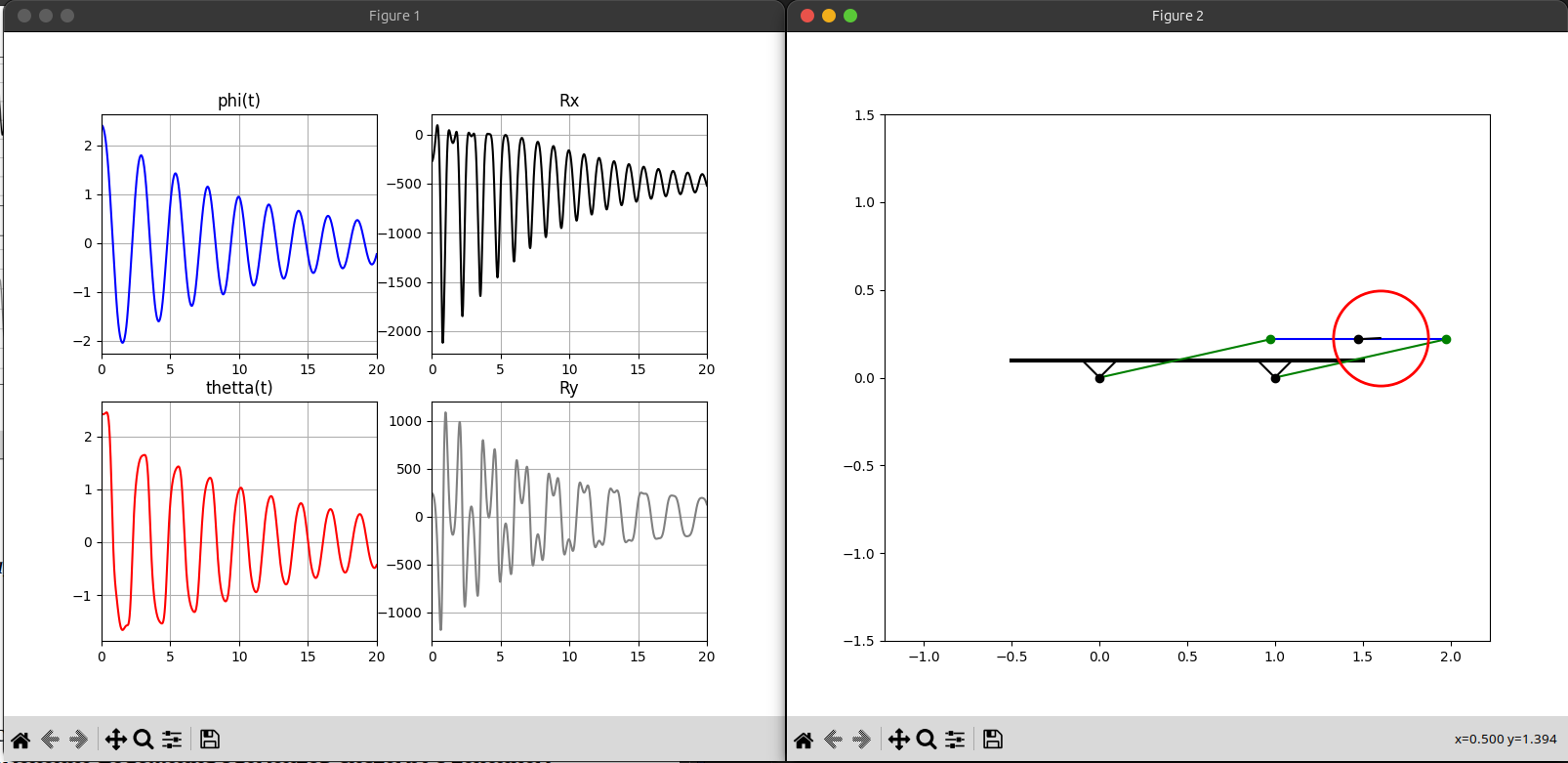
*2. g = 9.81; m1 = 50; m2 = 20; r = 0.25; l = 1; b = 0.125; k = 10; phi\_0 = math.pi / 2; thetta\_0 = math.pi / 2; dphi\_0 = 0; dthetta\_0 = 0; y0 = [phi\_0, thetta\_0, dphi\_0, dthetta\_0].*

*При небольшом увеличении масс стержня и диска система начинает двигаться более правдоподобно, колебания стержня затухают быстрее, а диск теперь совершает гармонические затухающие колебания.*

**

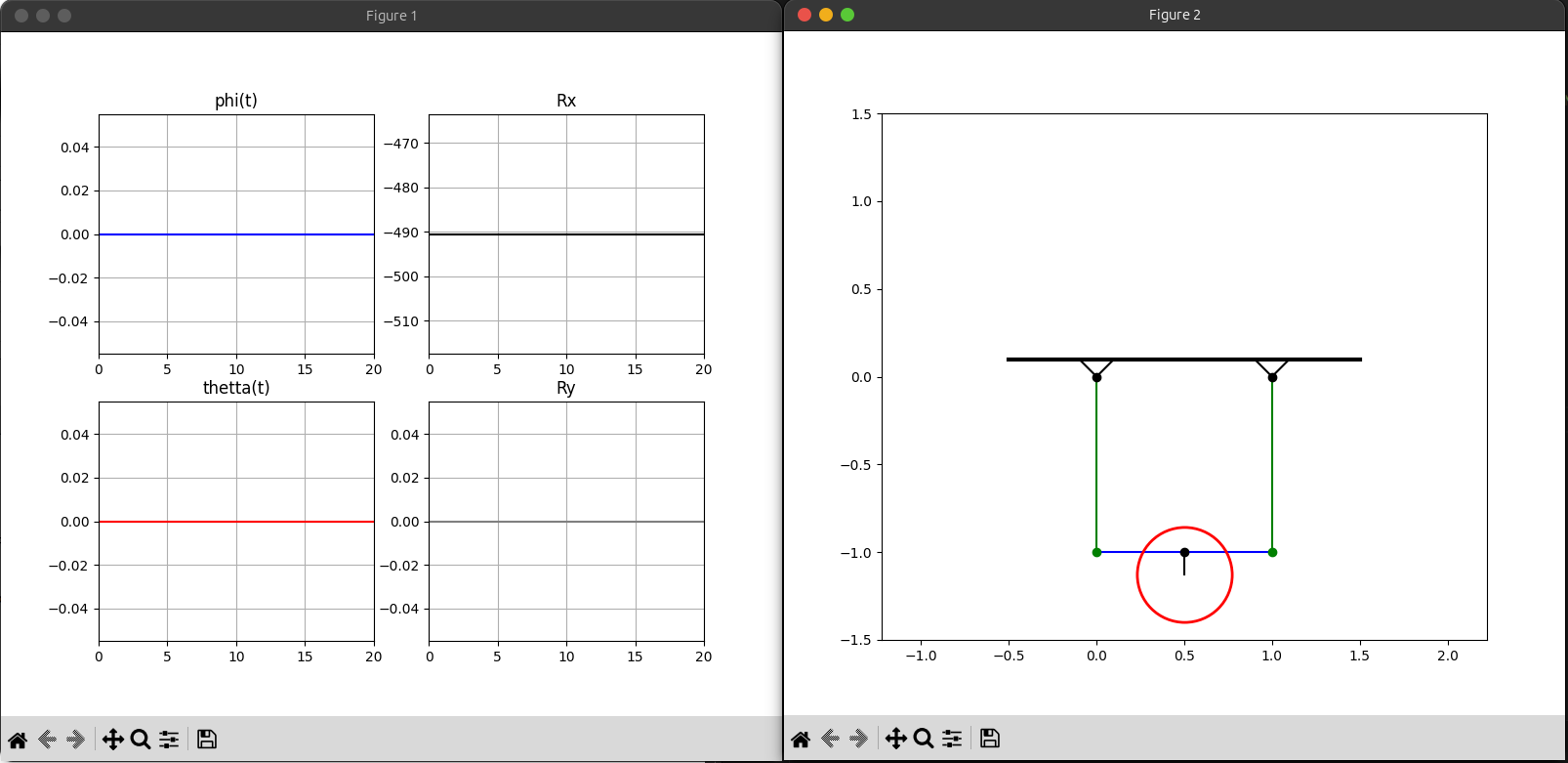
*3. g = 9.81; m1 = 50; m2 = 20; r = 0.25; l = 1; b = 0.125; k = 10; phi\_0 = math.pi / 1.3; thetta\_0 = math.pi / 1.3; dphi\_0 = 0; dthetta\_0 = 0; y0 = [phi\_0, thetta\_0, dphi\_0, dthetta\_0].*

*При увеличении начального угла отклонения стержней и диска система будет совершать колебания с большей скоростью и амплитудой, т. к. изначально она имеет большую потенциальную энергию для совершения колебаний.*

**

*4. g = 9.81; m1 = 50; m2 = 20; r = 0.25; l = 1; b = 0.125; k = 10; phi\_0 = 0; thetta\_0 = 0; dphi\_0 = 0; dthetta\_0 = 0; y0 = [phi\_0, thetta\_0, dphi\_0, dthetta\_0].*

*При нулевых значениях начальных углов отклонения стержней и диска система будет покоиться, т. к. не имеет энергии для совершения колебаний.*

**

*7) Вывод:*

При выполнении данной лабораторной работы помимо создания анимации механической системы мы научились интегрировать систему дифференциальных уравнений движения системы с двумя степенями свободы с помощью средств Python, строить графики законов движения системы и указанных в задании реакций для разных случаев системы. Также мы рассмотрели, как ведёт себя система при разных начальных значениях параметров.