ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ОТЧЕТ**

**О ВЫПЛОНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**«ДИНАМИКА СИСТЕМЫ»**

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

**ВАРИАНТ ЗАДАНИЯ № 14**

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

Марков И. И. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

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

подпись, дата

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

Москва, 2022

**Лабораторная работа 3.**

*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. Исследовать на устойчивость.

**1. Схема программы**

Для решения поставленных задач требуется сделать следующие шаги:

1. Отдельно от основной программы с помощью уравнений движения системы требуется сформировать функцию, которая будет принимать в себя значения , а на выход вернёт значения .
2. В основной программе требуется задать значения всех параметров, начальное положение системы, и запустить процедуру численного интегрирования системы.
3. Результаты численного интегрирования системы далее следует использовать при построении анимации движения системы.

**2. Составление функции уравнений движения**

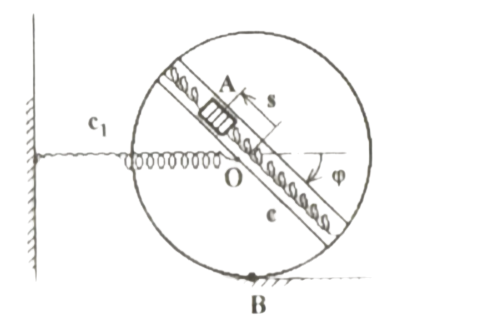


Рис. 1

Для того, чтобы узнать законы движения по координатам , необходимо определить функцию , которая будет принимать на вход вектор состояния и момент времени . Значением такой функции будет вектор .

Для определения такой функции воспользуемся уравнением движения системы, являющейся уравнением Лагранжа II рода:





Функция имеет вид:

def odesys(y, t, m1, m2, c, c1, R, g):

dy = np.zeros(4)

dy[0] = y[2]

dy[1] = y[3]

a11 = -m2 \* R \* np.cos(y[1])

a12 = ((2 \* m1 + m2) \* R\*\*2 + m2 \* y[0] + 2 \* R \* m2 \* y[0] \* np.sin(y[1]))

a21 = 1

a22 = -R \* np.cos(y[1])

b1 = -m2 \* R \* y[0] \* y[3]\*\*2 \* np.cos(y[1]) - 2 \* m2 \* (y[0] + R \* np.sin(y[1])) \* y[2] \* y[3] - c1 \* R\*\*2 \* y[1] - m2 \* g \* y[0] \* np.cos(y[1])

b2 = y[0] \* y[3]\*\*2 - 2 \* (c/m2) \* y[0] - g \* np.sin(y[1])

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

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

return dy

На вход функции odesys приходит вектор y, представленный в виде столбца, реализованного в библиотеке numpy. В качестве параметра t передается момент времени. Параметры m1, m2, c, c1, R, g являются константами системы и обозначают массу кольца, груза, жесткость пружин, радиус кольца и ускорение свободного падения.

**3. Численное интегрирование системы уравнений**

Следующим шагом является численное интегрирование функции odesys. Для этого я использовал функцию из библиотеки scipy под названием odeint. Она производит численное интегрирование функции по указанным параметрам. Итого, численное интегрирование выглядит следующим образом:

m1 = 1

m2 = 0.5

R = 1

g = 9.81

c = c1 = 5

t\_fin = 20

t = np.linspace(0, t\_fin, 1001)

s0 = 0

phi0 = np.pi/2

ds0 = 0

dphi0 = 1

y0 = [s0, phi0, ds0, dphi0]

Y = odeint(odesys, y0, t, (m1, m2, c, c1, R, g))

s = Y[:, 0]

phi = Y[:, 1]

ds = Y[:, 2]

dphi = Y[:, 3]

Где y0 содержит начальное состояние системы.

**4. Построение графиков**

Построим графики решения решения , а также . Для построения графиков я использовал библиотеку pyplot. Итого, код программы, отрисовывающей графики, следующий

fig\_for\_graphs = plt.figure(figsize=[13,7])

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

ax\_for\_graphs.plot(t, s, color='blue')

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

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

ax\_for\_graphs.grid(True)

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

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

ax\_for\_graphs.set\_title('phi(t)')

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

ax\_for\_graphs.grid(True)

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

ax\_for\_graphs.plot(t,ds,color='green')

ax\_for\_graphs.set\_title("s'(t)")

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

ax\_for\_graphs.grid(True)

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

ax\_for\_graphs.plot(t,dphi,color='black')

ax\_for\_graphs.set\_title('phi\'(t)')

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

ax\_for\_graphs.grid(True)

**5. Построение анимации**

Теперь построим анимацию движения системы, используя полученные результаты. Оформим графическое окно, создадим нарисованные объекты в начальном положении и запустим цикл, переставляющий объект в положения, соответствующим новым моментам времени

def spring(k, h, w):

x = np.linspace(0, h, 100)

return np.array([

x,

np.sin(2 \* pi / (h / k) \* x) \* w

])

ring\_dots\_x\_tmp = R \* np.cos(angles)

ring\_dots\_y\_tmp = R \* np.sin(angles)

box\_x\_tmp = np.array([-box\_h / 2, -box\_h / 2, box\_h / 2, box\_h / 2, -box\_h / 2])

box\_y\_tmp = np.array([-box\_w / 2, box\_w / 2, box\_w / 2, -box\_w / 2, -box\_w / 2])

ring\_dots\_x = np.zeros([len(t), len(angles)])

ring\_dots\_y = np.zeros([len(t), len(angles)])

box\_dots\_x = np.zeros([len(t), 5])

box\_dots\_y = np.zeros([len(t), 5])

spring\_a\_x = np.zeros([len(t), 100])

spring\_a\_y = np.zeros([len(t), 100])

spring\_b\_x = np.zeros([len(t), 100])

spring\_b\_y = np.zeros([len(t), 100])

spring\_c\_x = np.zeros([len(t), 100])

spring\_c\_y = np.zeros([len(t), 100])

for i in range(len(t)):

ring\_x = x0 + phi[i] \* R

ring\_y = R

ring\_dots\_x[i] = np.cos(phi[i]) \* ring\_dots\_x\_tmp + np.sin(phi[i]) \* ring\_dots\_y\_tmp + ring\_x

ring\_dots\_y[i] = - np.sin(phi[i]) \* ring\_dots\_x\_tmp + np.cos(phi[i]) \* ring\_dots\_y\_tmp + ring\_y

bx = box\_x\_tmp - s[i]

by = box\_y\_tmp

box\_dots\_x[i] = np.cos(phi[i]) \* bx + np.sin(phi[i]) \* by + ring\_x

box\_dots\_y[i] = - np.sin(phi[i]) \* bx + np.cos(phi[i]) \* by + ring\_y

spring\_a\_x[i] = spring(5, ring\_x, 0.2)[0]

spring\_a\_y[i] = spring(5, ring\_x, 0.2)[1] + ring\_y

b\_x = R - spring(10, R + s[i] - box\_h / 2, 0.2)[0]

b\_y = spring(10, R - s[i], 0.2)[1]

spring\_b\_x[i] = np.cos(phi[i]) \* b\_x + np.sin(phi[i]) \* b\_y + ring\_x

spring\_b\_y[i] = -np.sin(phi[i]) \* b\_x + np.cos(phi[i]) \* b\_y + ring\_y

c\_x = spring(10, R - s[i] - box\_h / 2, 0.2)[0] - R

c\_y = spring(10, R - s[i], 0.2)[1]

spring\_c\_x[i] = np.cos(phi[i]) \* c\_x + np.sin(phi[i]) \* c\_y + ring\_x

spring\_c\_y[i] = -np.sin(phi[i]) \* c\_x + np.cos(phi[i]) \* c\_y + ring\_y

fig = plt.figure() # создаем холст, на котором будем рисовать фигуры

ax = fig.add\_subplot(1, 1, 1)

ax.axis("equal")

surface = ax.plot([0, 0, 10], [10, 0, 0]) # пол и стена

ring, = ax.plot(ring\_dots\_x[0], ring\_dots\_y[0]) # кольцо

box, = ax.plot(box\_dots\_x[0], box\_dots\_y[0]) # груз

spring\_a, = ax.plot(spring\_a\_x[0], spring\_a\_y[0]) # горизонтальная пружина

spring\_b, = ax.plot(spring\_b\_x[0], spring\_b\_y[0]) # нижняя пружина

spring\_c, = ax.plot(spring\_c\_x[0], spring\_c\_y[0]) # верхняя пружина

def animate(i):

ring.set\_data(ring\_dots\_x[i], ring\_dots\_y[i])

box.set\_data(box\_dots\_x[i], box\_dots\_y[i])

spring\_a.set\_data(spring\_a\_x[i], spring\_a\_y[i])

spring\_b.set\_data(spring\_b\_x[i], spring\_b\_y[i])

spring\_c.set\_data(spring\_c\_x[i], spring\_c\_y[i])

return ring, box, spring\_a, spring\_b, spring\_c

animation = FuncAnimation(fig, animate, frames=len(t), interval=15)

plt.show()

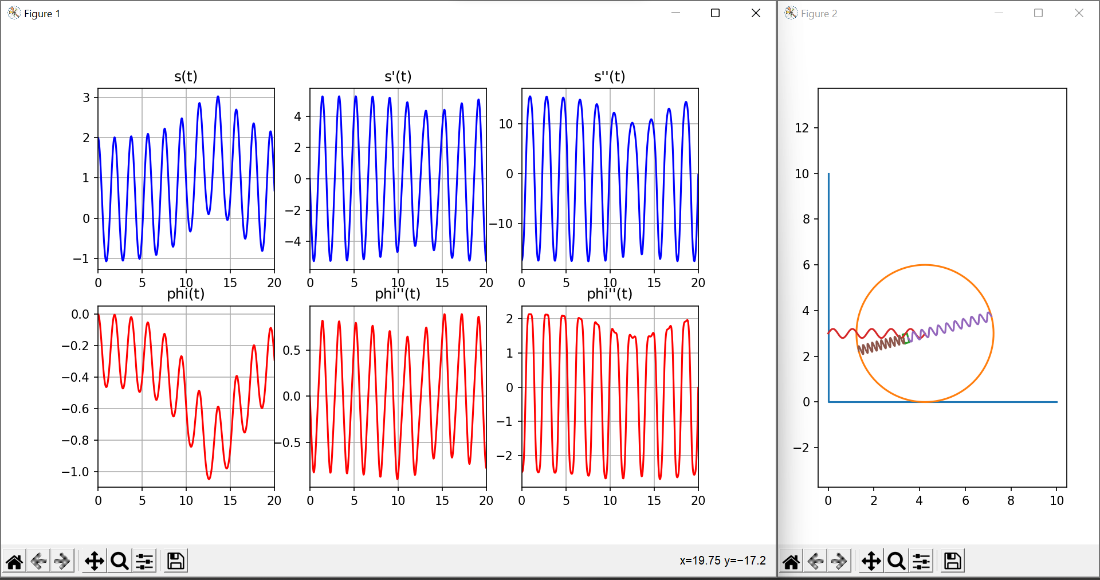
Функция spring является шаблоном для генерации точек пружины высотой w, шириной h и количеством витков k. Первым шагом я создал пустые массивы, которые в ходе выполнения цикла заполнял необходимыми значениями. После чего, используя заполненные массивы я создал графические объекты и анимировал их, реализовав функцию animate, выполняющую правила перехода к новому кадру анимации.

**6. Результат работы программы**

Выведем полученные графики работы программы

**1.**

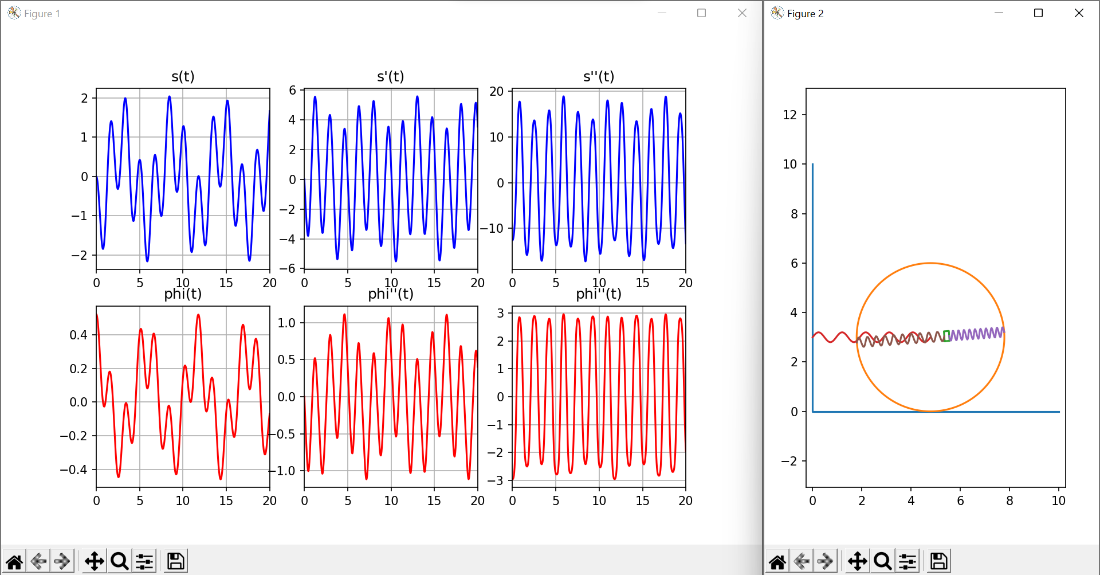
Кольцо имеет радиус 1, коэффициент упругости всех пружин одинаковый. Масса груза превышает массу кольца в 2 раза. В начальный момент времени кольцо не повернуто, горизонтальная пружина не деформирована, а груз отклонен.



**Результат:** Брусок совершает колебания, центр равновесия которых со временем изменяется (что видно по среднему значению графика . Также, колебания бруска затрагивают и само кольцо, которое начинает также совершает колебательное движение.

**2.**

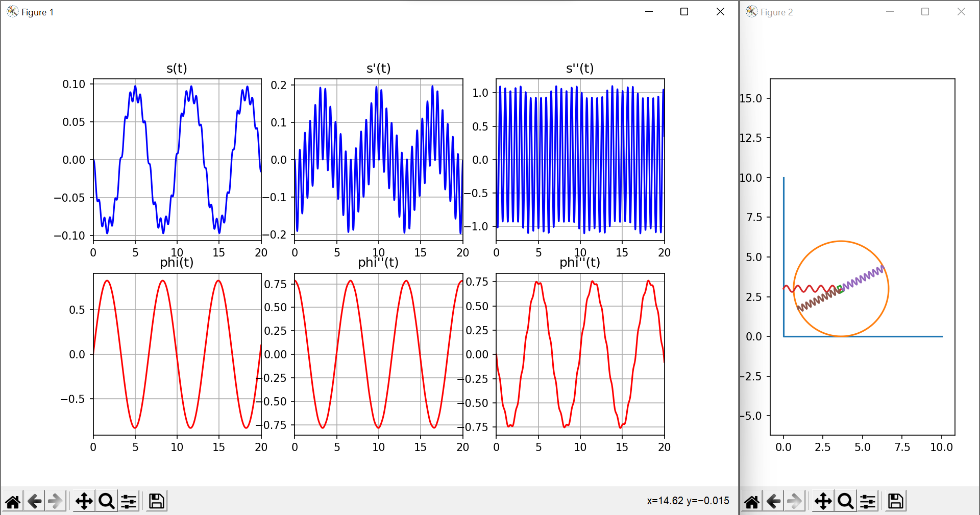
В начальный момент времени кольцо отклонено на малый угол. Обе пружины, находящие в кольце, в положении равновесия, Коэффициент упругости пружин в 4 раза больше по сравнению с прошлым опытом.



**Результат:** Кольцо совершает колебательные движения. Однако, они затрагивает и груз, из-за чего он также совершает колебательные движения. В ходе движения, кольцо останавливается в один момент времени.

**3.**

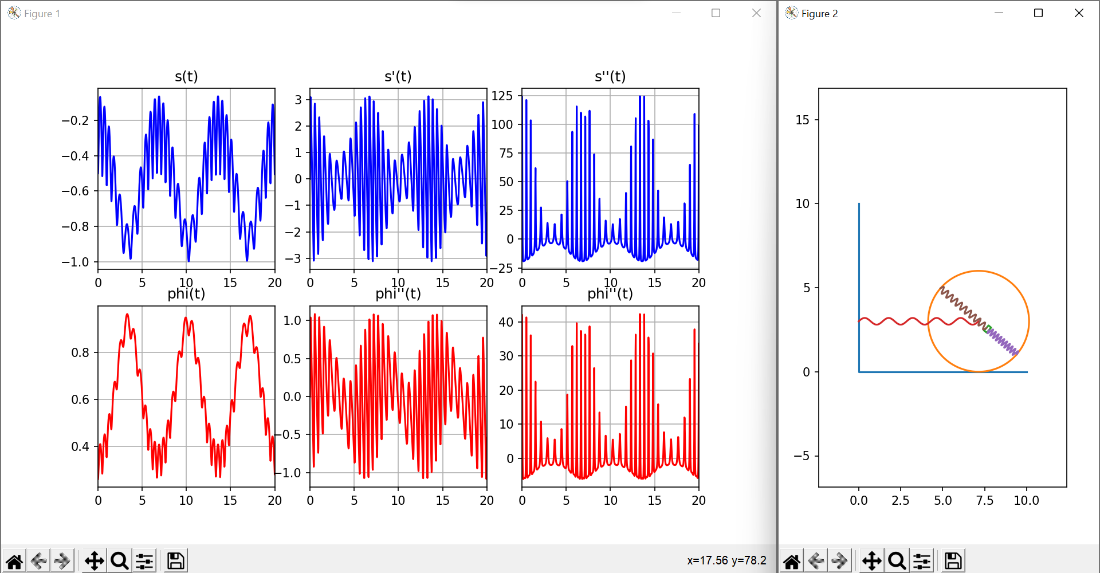
Массивное по сравнению с грузом кольцо, у которого ненулевая начальная угловая скорость. Коэффициенты упругости пружин внутри кольца имеют значительно большие значения.



**Результат:** Кольцо совершает колебательные движения. Груз же совершает колебания с крайне малой амплитудой, что практически не сказывается на движении кольца из-за большой массы последнего.

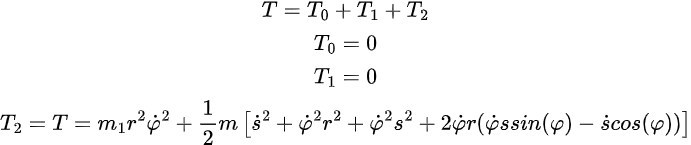
**4.**

Груз имеет значительно большую массу по сравнению с кольцом. Коэффициент упругости пружин внутри кольца имеет большое значение. В начальный момент времени кольцо и груз отклонены.



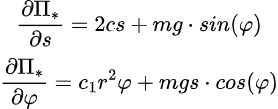
**Результат:** Вся система совершает прерывистые движения, циклически ускоряясь и резко останавливаясь.

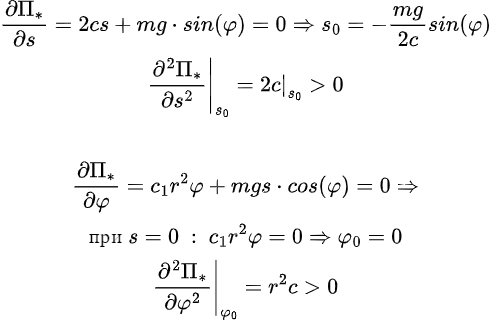
**7. Исследование на устойчивость**











Начальные положения - устойчивый