ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**«АНИМАЦИЯ СИСТЕМЫ»**

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

**ВАРИАНТ ЗАДАНИЯ №4**

Выполнил(а) студент группы M8O-201Б-23

Вельма А.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

Ст. преп. каф. 802 Волков Е.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

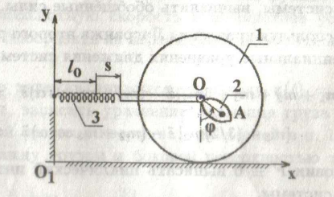
с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Задание:**

Реализовать анимацию движения механической системы используя язык программирования Python.

**Механическая система:**



**Текст программы**

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

import sympy as sp

import math

# Функция для поворота координат вокруг точки (XC, YC) на угол Alpha

def Rot(X, Y, Alpha, XC, YC):

    RX = (X - XC) \* np.cos(Alpha) - (Y - YC) \* np.sin(Alpha) + XC  # Новая координата X

    RY = (X - XC) \* np.sin(Alpha) + (Y - YC) \* np.cos(Alpha) + YC  # Новая координата Y

    return RX, RY

# Ввод символьной переменной времени t и радиуса R

t = sp.Symbol("t")  # Время

R = 1  # Радиус большой окружности

# Параметрические уравнения

Alpha = sp.cos(6 \* t) / 2  # Угол поворота шарика на маятнике

xc = sp.sin(t) + 2  # Координата X центра большой окружности

xa = xc - 0.05 \* sp.sin(Alpha)  # Координата X маятника

ya = 0.9 + 0.05 \* sp.cos(Alpha)  # Координата Y маятника

# Производные для вычисления скоростей

Vx = sp.diff(xc, t)  # Скорость по X для центра окружности

Vy = 0 \* t  # Скорость по Y для центра окружности (постоянная)

omega = sp.diff(Alpha, t)  # Угловая скорость маятника

Vxa = Vx - omega \* R \* sp.cos(Alpha)  # Скорость по X для маятника

Vya = Vy - omega \* R \* sp.sin(Alpha)  # Скорость по Y для маятника

# Генерация временного интервала и массивов для хранения данных

T = np.linspace(0, 10, 1000)  # Временной интервал

XC = np.zeros\_like(T)  # Массив координат X центра окружности

XA = np.zeros\_like(T)  # Массив координат X маятника

YA = np.zeros\_like(T)  # Массив координат Y маятника

ALPHA = np.zeros\_like(T)  # Массив углов Alpha

VX = np.zeros\_like(T)  # Массив скоростей по X для центра окружности

VY = np.zeros\_like(T)  # Массив скоростей по Y для центра окружности

VXA = np.zeros\_like(T)  # Массив скоростей по X для маятника

VYA = np.zeros\_like(T)  # Массив скоростей по Y для маятника

# Заполнение массивов через цикл

for i in np.arange(len(T)):

    XC[i] = sp.Subs(xc, t, T[i])  # Координата X центра окружности

    VX[i] = sp.Subs(Vx, t, T[i])  # Скорость по X для центра окружности

    VY[i] = sp.Subs(Vy, t, T[i])  # Скорость по Y для центра окружности

    XA[i] = sp.Subs(xa, t, T[i])  # Координата X маятника

    YA[i] = sp.Subs(ya, t, T[i])  # Координата Y маятника

    VXA[i] = sp.Subs(Vxa, t, T[i])  # Скорость по X для маятника

    VYA[i] = sp.Subs(Vya, t, T[i])  # Скорость по Y для маятника

    ALPHA[i] = sp.Subs(Alpha, t, T[i])  # Угол Alpha

# Построение графиков

fig = plt.figure(figsize=(17, 8))

ax1 = fig.add\_subplot(1, 2, 1)

ax1.axis("equal")  # Выравнивание осей

# Построение осей координат

(liney,) = ax1.plot([0, 0], [0, 5], "black")  # Вертикальная ось

(linex,) = ax1.plot([0, 5], [0, 0], "black")  # Горизонтальная ось

ArrowX = np.array([-0.2, 0, -0.2])  # Стрелка оси X

ArrowY = np.array([0.1, 0, -0.1])  # Стрелка оси Y

(ArrowOY,) = ax1.plot(ArrowY, ArrowX + 5, "black")  # Стрелка для оси Y

(ArrowOX,) = ax1.plot(ArrowX + 5, ArrowY, "black")  # Стрелка для оси X

# Построение соединительной линии

# Построение окружности

(P,) = ax1.plot(XC[0], R, marker="o", color="black")  # Центр окружности

Phi = np.linspace(0, 2 \* math.pi, 200)  # Углы для построения окружности

(Circ,) = ax1.plot(XC[0] + R \* np.cos(Phi), R + R \* np.sin(Phi), "black")  # Окружность

# Построение маятника

Mayatnik = ax1.plot(

    XA[0] + 0.05 \* np.cos(Phi), YA[0] + 0.1 \* np.sin(Phi), color="black"

)[0]  # Маятник, прикрепленный к окружности

# Дополнительные графики скоростей

ax2 = fig.add\_subplot(4, 2, 2)

ax2.plot(T, VX)  # Скорость по X для центра окружности

ax2.set\_xlabel("T")

ax2.set\_ylabel("VX")

ax3 = fig.add\_subplot(4, 2, 4)

ax3.plot(T, VY)  # Скорость по Y для центра окружности

ax3.set\_xlabel("T")

ax3.set\_ylabel("VY")

ax4 = fig.add\_subplot(4, 2, 6)

ax4.plot(T, VXA)  # Скорость по X для маятника

ax4.set\_xlabel("T")

ax4.set\_ylabel("Vx маятника")

ax5 = fig.add\_subplot(4, 2, 8)

ax5.plot(T, VYA)  # Скорость по Y для маятника

ax5.set\_xlabel("T")

ax5.set\_ylabel("Vy маятника")

# Параметры пружины

spring\_segments = 20  # Количество сегментов пружины

spring\_height = 0.3  # Высота каждого сегмента

# Создаём линию для пружины

spring\_line, = ax1.plot([], [], color="purple", linewidth=2)  # Фиолетовая пружина

def create\_spring(start\_x, end\_x, start\_y, end\_y, segments, height):

    spring\_x = np.linspace(start\_x, end\_x, segments \* 2)

    spring\_y = np.zeros\_like(spring\_x)

    for i in range(len(spring\_x)):

        if i % 2 == 0:

            spring\_y[i] = start\_y + (i / (segments \* 2)) \* (end\_y - start\_y) - height

        else:

            spring\_y[i] = start\_y + (i / (segments \* 2)) \* (end\_y - start\_y) + height

    return spring\_x, spring\_y

# Функция анимации

def anima(i):

    NewX = []

    NewY = []

    # Обновление данных

    P.set\_data([XC[i]], [R])  # Центр окружности

    Circ.set\_data(XC[i] + R \* np.cos(Phi), R + R \* np.sin(Phi))  # Окружность

    spring\_x, spring\_y = create\_spring(0, XC[i], R, R, spring\_segments, spring\_height)

    spring\_line.set\_data(spring\_x, spring\_y)

    # Обновление маятника

    for phi in Phi:

        newx, newy = Rot(

            XC[i] + 0.15 \* math.cos(phi),

            0.1 + 0.3 \* math.sin(phi) + 0.6,

            ALPHA[i],

            XC[i],

            R,

        )

        NewX.append(newx)

        NewY.append(newy)

    Mayatnik.set\_data(NewX, NewY)

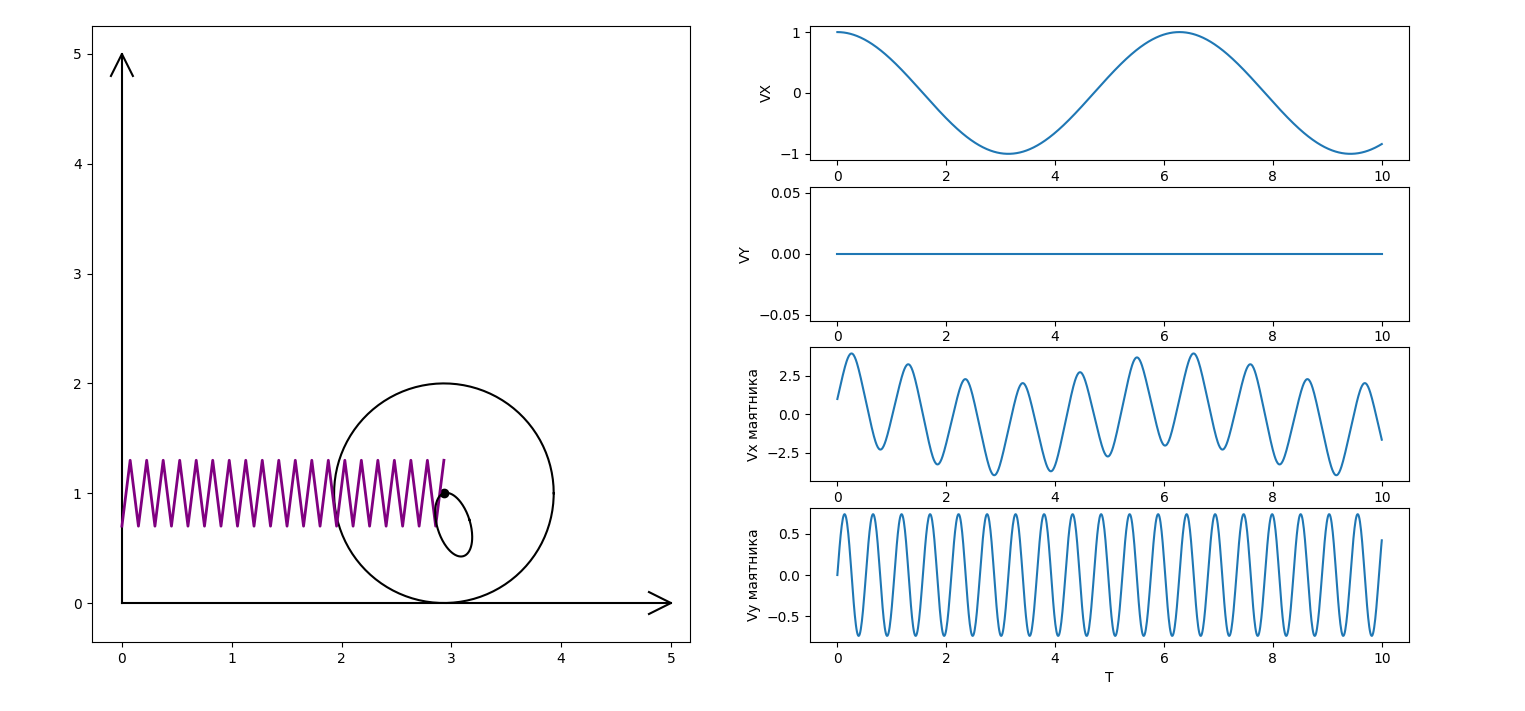
    # Возвращаем обновляемые объекты

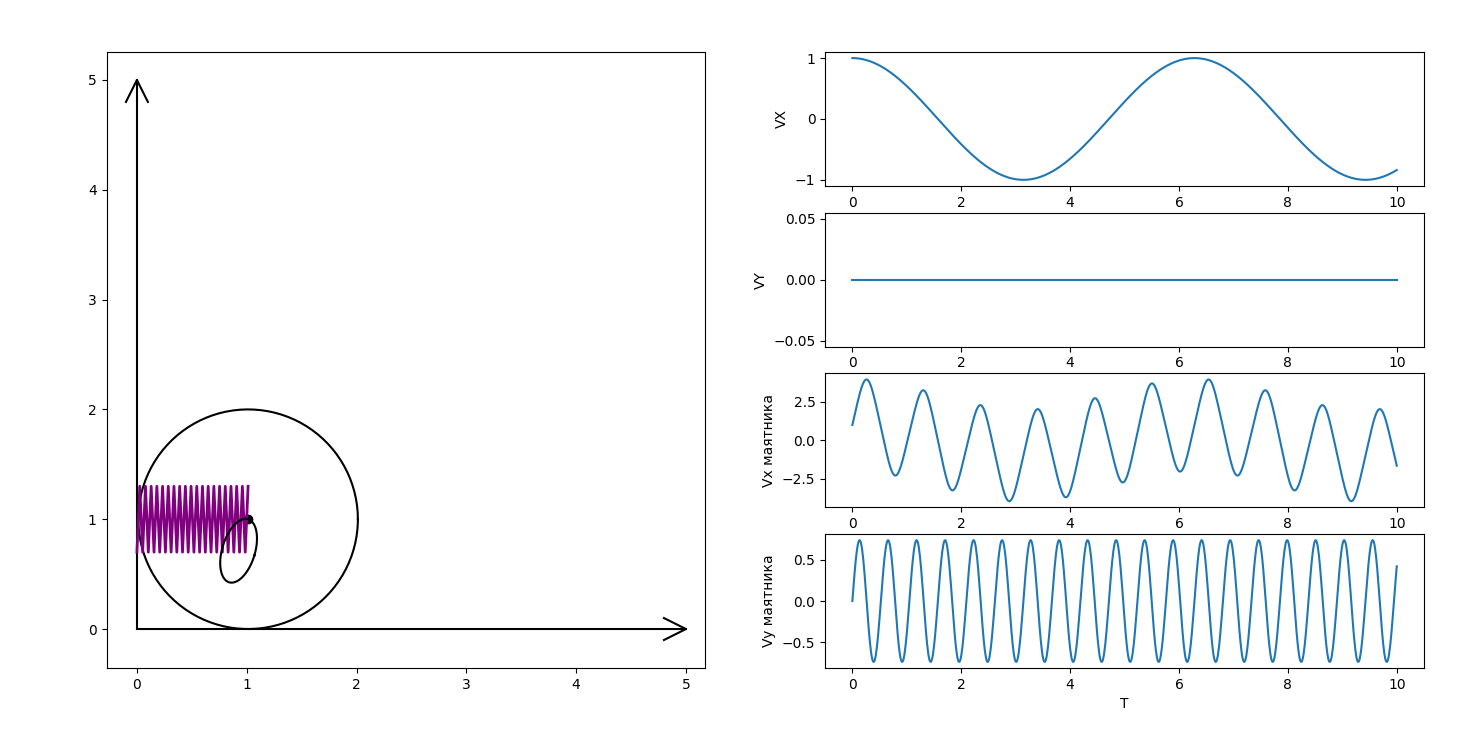
    return Circ, P, spring\_line, Mayatnik

# Создание анимации

anim = FuncAnimation(fig, anima, frames=500, interval=60, blit=True)

plt.show()

**Результат работы:** ****

****

**Вывод:**

В процессе выполнения второй лабораторной работы была разработана анимация механической системы, представляющей движение окружности и маятника, прикрепленного к её центру. Для реализации использовались библиотеки Python: sympy для параметризации системы и расчёта производных, а также matplotlib для визуализации и создания анимации. В ходе работы были определены ключевые параметры движения, рассчитаны координаты точек системы и построена анимация. Реализована динамика пружины, связанной с движением окружности. Эта работа позволила изучить основы символьных вычислений и построения анимационных моделей на языке программирования Python.