ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**«АНИМАЦИЯ СИСТЕМЫ»**

**ПО ДИСЦИПЛИНЕ «Основы Компьютерного Моделирования Математических Систем»**

**ВАРИАНТ ЗАДАНИЯ №17**

Выполнил(а) студент группы М8О-209Б-23

Мазепа И.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

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

подпись, дата

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

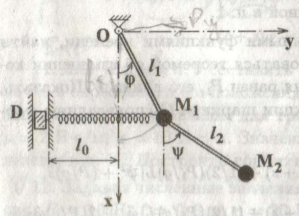
Москва, 2024

**Вариант № 17**

**Задание:**

Реализовать анимацию движения механической системы используя язык программирования Python.

**Механическая система:**

****

**Текст программы**

import numpy as np

from scipy.integrate import solve\_ivp

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

# Параметры системы

mass = 1.0 # масса материальных точек

rod\_length1 = 1.0 # длина первого стержня

rod\_length2 = 1.0 # длина второго стержня

spring\_constant = 0.1 # жёсткость пружины

gravity = 9.81 # ускорение свободного падения

damping\_coefficient = 0.1 # Дампинг

# Начальные условия

initial\_phi = np.pi / 6 # Начальное отклонение первого маятника

initial\_psi = np.pi / 2 # Начальное отклонение второго маятника

initial\_omega1 = 0.0 # Начальная угловая скорость первого маятника

initial\_omega2 = 0.0 # Начальная угловая скорость второго маятника

# Временной интервал

dt = 0.005 # Шаг интегрирования

time\_span = (0, 20.0)

time\_eval = np.arange(time\_span[0], time\_span[1], dt)

# Уравнения движения

def equations(t, y):

"""

y = [phi, psi, omega1, omega2]

phi, psi: углы отклонения

omega1, omega2: угловые скорости

"""

phi, psi, omega1, omega2 = y

# Координаты точек M1 и M2

x1 = rod\_length1 \* np.sin(phi)

y1 = -rod\_length1 \* np.cos(phi)

x2 = x1 + rod\_length2 \* np.sin(psi)

y2 = y1 - rod\_length2 \* np.cos(psi)

# Длина пружины

spring\_length = np.sqrt(x1\*\*2 + y1\*\*2)

# Сила пружины и её проекции

if spring\_length == 0:

fx\_spring = 0

fy\_spring = 0

else:

spring\_force = -spring\_constant \* (spring\_length - rod\_length1) / spring\_length

fx\_spring = spring\_force \* x1

fy\_spring = spring\_force \* y1

# Угловые ускорения

domega1 = (

-mass \* gravity \* rod\_length1 \* np.sin(phi)

+ fx\_spring \* rod\_length1 \* np.cos(phi)

+ fy\_spring \* rod\_length1 \* np.sin(phi)

- damping\_coefficient \* omega1

- spring\_constant \* omega1

) / (mass \* rod\_length1\*\*2)

domega2 = (

-mass \* gravity \* rod\_length2 \* np.sin(psi)

- damping\_coefficient \* omega2

- spring\_constant \* omega2

) / (mass \* rod\_length2\*\*2)

return [omega1, omega2, domega1, domega2]

# Начальные условия

initial\_conditions = [initial\_phi, initial\_psi, initial\_omega1, initial\_omega2]

# Численное решение

solution = solve\_ivp(equations, time\_span, initial\_conditions, t\_eval=time\_eval, method='RK45')

# Анимация

fig, ax = plt.subplots(figsize=(10, 6))

# Создаем объекты для анимации

# "брусок"

block\_width, block\_height = 0.1, 0.2

block = plt.Rectangle((-block\_width/2, -block\_height/2),

block\_width, block\_height,

color='gray', zorder=5)

ax.add\_patch(block)

# Рисуем "стенки"

left\_wall = plt.plot([-1.5 - block\_width / 2 - 0.03, -1.5 - block\_width / 2 - 0.03], [-0.5, -1.2], color='k', linewidth=1)

right\_wall = plt.plot([-1.5 + block\_width / 2 + 0.03, -1.5 + block\_width / 2 + 0.03], [-0.5, -1.2], color='k', linewidth=1)

rod1, = ax.plot([], [], 'o-', lw=2, markersize=10)

rod2, = ax.plot([], [], 'o-', lw=2, markersize=10)

spring\_line, = ax.plot([], [], 'b-', lw=2)

ax.set\_xlim(-rod\_length1 - rod\_length2 - 0.5, rod\_length1 + rod\_length2 + 0.5)

ax.set\_ylim(-rod\_length1 - rod\_length2 - 0.5, rod\_length1 + rod\_length2 + 0.5)

ax.set\_aspect('equal')

ax.grid(True)

# Функция для вычисления координат зигзага

def get\_spring\_zigzag(x\_start, y\_start, x\_end, y\_end, turns=8, amplitude=0.1):

# Делаем разбиение по длине

num\_points = 2 \* turns + 1

xs = np.linspace(x\_start, x\_end, num\_points)

ys = np.linspace(y\_start, y\_end, num\_points)

# Смещаем точки по нормали для зигзага

zigzag\_x = []

zigzag\_y = []

for i in range(num\_points):

# пропорция вдоль пружины

t = i / (num\_points - 1)

# направление

dx = x\_end - x\_start

dy = y\_end - y\_start

length = np.sqrt(dx\*\*2 + dy\*\*2)

# нормаль

nx = -dy / length

ny = dx / length

# смещение

offset = amplitude if i % 2 == 1 else -amplitude

# координаты

zx = xs[i] + offset \* nx

zy = ys[i] + offset \* ny

zigzag\_x.append(zx)

zigzag\_y.append(zy)

return zigzag\_x, zigzag\_y

def init():

rod1.set\_data([], [])

rod2.set\_data([], [])

spring\_line.set\_data([], [])

block.set\_xy((-block\_width/2, -block\_height/2))

return rod1, rod2, spring\_line, block

def update(frame):

phi, psi, omega1, omega2 = solution.y[:, frame]

x1 = rod\_length1 \* np.sin(phi)

y1 = -rod\_length1 \* np.cos(phi)

x2 = x1 + rod\_length2 \* np.sin(psi)

y2 = y1 - rod\_length2 \* np.cos(psi)

rod1.set\_data([0, x1], [0, y1])

rod2.set\_data([x1, x2], [y1, y2])

# Зигзаг от центра (там где брусок) до первой массы

zx, zy = get\_spring\_zigzag(-1.5, y1, x1, y1, turns=6, amplitude=0.05)

spring\_line.set\_data(zx, zy)

# Брусок в конце пружины

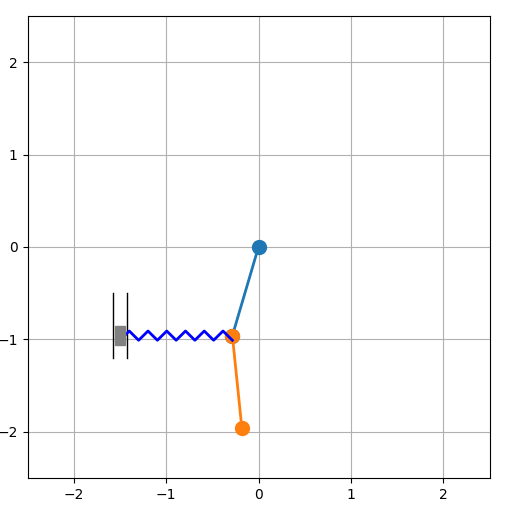
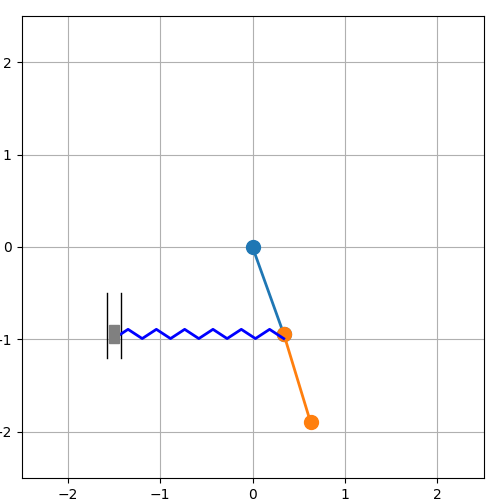
block.set\_xy((-1.5 - block\_width/2, y1 - block\_height/2))

return rod1, rod2, spring\_line, block

ani = FuncAnimation(fig, update, frames=len(time\_eval), init\_func=init, blit=True, interval=5)

plt.show()

**Результат работы:**



**Вывод:**  
В процессе выполнения лабораторной работы была разработана анимация механической системы с применением библиотек numpy и matplotlib. Основные этапы включали задание параметров системы, вычисление координат точек и создание визуализации движения.