|  |  |
| --- | --- |
| Группа М3217 \_\_ | К работе допущен |
| Студент \_ Бессонов Борис\_\_\_\_\_\_\_\_\_\_ | Работа выполнена 30.05.24 |
| Преподаватель Писарева Юлия | Отчет принят |

**Рабочий протокол и отчет по  
моделированию работе №4**

**Связанные маятники**

1. Цель работы.

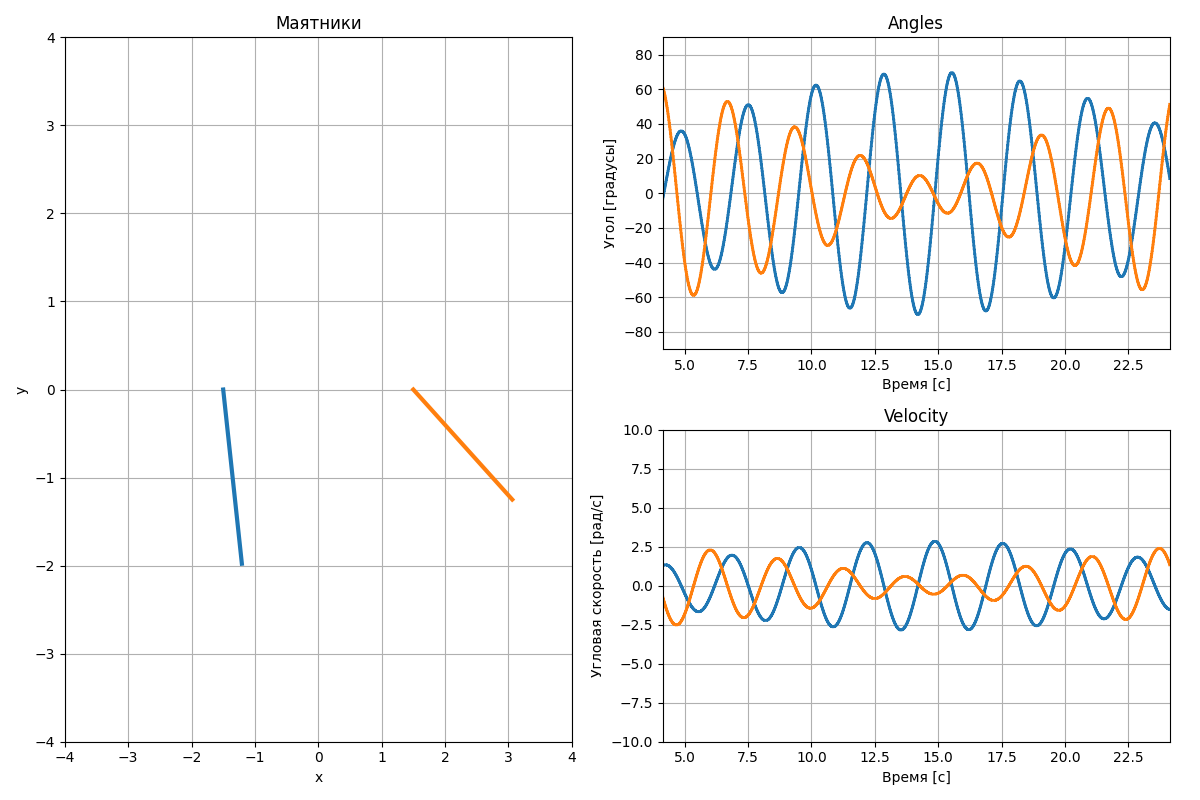
Написать программу, моделирующую два маятника, связанных пружиной с коэффициентом жесткости k, на расстоянии L1 от точки крепления.

1. Задачи, решаемые при выполнении работы

Найти уравнение углов связанных маятников

1. Рабочие формулы и исходные данные

*Конечная формула:*

1. Графики (*перечень графиков, которые составляют Приложение 2*).
2. Результат работы

Программа:

Pendulum.py

import numpy as np

from numpy import sin, cos

from scipy.integrate import solve\_ivp

class Pendulum:

    def \_\_init\_\_(self, angle\_1, angle\_2, g, l, m, k, stop\_force, dist\_between\_pendulum):

        self.angles = [[angle\_1, 0.0], [angle\_2, 0.0]]  # угол, скорость, угол2, скорость2

        self.dist\_between\_pendulum = dist\_between\_pendulum

        self.g = g

        self.L = l

        self.m = m

        self.K = k

        self.stop\_force = stop\_force

        self.elapsed\_time = 0

        self.graph\_data\_1 = [[], [], []]    # время, угол, скорость

        self.graph\_data\_2 = [[], [], []]

    def get\_angle(self):

        return self.angles[0][0], self.angles[1][0]

    def get\_velocity(self):

        return self.angles[0][1], self.angles[1][1]

    def equations(self, t, angles):

        self.graph\_data\_1[0].append(self.elapsed\_time)

        self.graph\_data\_1[1].append(np.degrees(angles[0]))

        self.graph\_data\_1[2].append(angles[1])

        self.graph\_data\_2[0].append(self.elapsed\_time)

        self.graph\_data\_2[1].append(np.degrees(angles[2]))

        self.graph\_data\_2[2].append(angles[3])

        dtheta1\_dt = angles[1]

        domega1\_dt = (- (self.g / self.L) \* angles[0] - self.stop\_force \* angles[1]

                      - (self.K / self.m) \* (angles[0] - angles[2]))

        dtheta2\_dt = angles[3]

        domega2\_dt = (- (self.g / self.L) \* angles[2] - self.stop\_force \* angles[3]

                      + (self.K / self.m) \* (angles[0] - angles[2]))

        return [dtheta1\_dt, domega1\_dt, dtheta2\_dt, domega2\_dt]

    def update(self, dt):

        state = [self.get\_angle()[0], self.get\_velocity()[0],

                 self.get\_angle()[1], self.get\_velocity()[1]]

        solution = solve\_ivp(self.equations, [0, dt], state, t\_eval=[0, dt])

        # print()

        # print(state)

        # print(solution.t)

        # print(solution.y)

        self.angles = [[solution.y[0][1], solution.y[1][1]],

                       [solution.y[2][1], solution.y[3][1]]]

        self.elapsed\_time += dt

    def position(self):

        # print(self.get\_angle())

        x\_1 = np.cumsum([-self.dist\_between\_pendulum / 2, self.L \* sin(self.get\_angle()[0])])

        y\_1 = np.cumsum([0, - self.L \* cos(self.get\_angle()[0])])

        x\_2 = np.cumsum([self.dist\_between\_pendulum / 2, self.L \* sin(self.get\_angle()[1])])

        y\_2 = np.cumsum([0, - self.L \* cos(self.get\_angle()[1])])

        return [x\_1, y\_1], [x\_2, y\_2]

main.py

from Pendul import Pendulum

import numpy as np

import matplotlib.pyplot as plt

import matplotlib

import matplotlib.animation as animation

FPS = 60

dist\_between\_pendulum = 3

g = 9.81

L = 2.0

m = 1.0

K = 0.5

stop\_force = 0.05

theta1 = np.radians(30)

theta2 = np.radians(-30)

# Построение графиков

screen = plt.figure(figsize=(12, 8))

# График маятников

pendulum\_graph = plt.subplot(1, 2, 1)

pendulum\_graph.set\_title('Маятники')

pendulum\_graph.set\_xlabel('x')

pendulum\_graph.set\_ylabel('y')

pendulum\_graph.set\_xlim([-4, 4])

pendulum\_graph.set\_ylim([-4, 4])

# pendulum\_graph.legend()

pendulum\_graph.grid()

# График углов

angle\_graph = plt.subplot(2, 2, 2)

angle\_graph.set\_title('Angles')

angle\_graph.set\_xlabel('Время [с]')

angle\_graph.set\_ylabel('Угол [градусы]')

angle\_graph.set\_xlim([0, 20])

angle\_graph.set\_ylim([-90, 90])

# angle\_graph.legend()

angle\_graph.grid()

# График угловых скоростей

velocity\_graph = plt.subplot(2, 2, 4)

velocity\_graph.set\_title('Velocity')

velocity\_graph.set\_xlabel('Время [с]')

velocity\_graph.set\_ylabel('Угловая скорость [рад/с]')

velocity\_graph.set\_xlim([0, 20])

velocity\_graph.set\_ylim([-10, 10])

# velocity\_graph.legend()

velocity\_graph.grid()

############

pendulums = Pendulum(theta1, theta2, g, L, m, K, stop\_force, dist\_between\_pendulum)

############

# legend

line\_1, = pendulum\_graph.plot([], [], '-', lw=3, color=matplotlib.colors.TABLEAU\_COLORS['tab:blue'])

line\_2, = pendulum\_graph.plot([], [], '-', lw=3, color=matplotlib.colors.TABLEAU\_COLORS['tab:orange'])

line\_alpha\_1, = angle\_graph.plot([], [], lw=2)

line\_alpha\_2, = angle\_graph.plot([], [], lw=2)

line\_velocity\_1, = velocity\_graph.plot([], [], lw=2)

line\_velocity\_2, = velocity\_graph.plot([], [], lw=2)

def animate(i):

    pendulums.update(1/FPS)

    line\_1.set\_data(pendulums.position()[0])

    line\_alpha\_1.set\_data(pendulums.graph\_data\_1[0], pendulums.graph\_data\_1[1])

    line\_velocity\_1.set\_data(pendulums.graph\_data\_1[0], pendulums.graph\_data\_1[2])

    line\_2.set\_data(pendulums.position()[1])

    line\_alpha\_2.set\_data(pendulums.graph\_data\_2[0], pendulums.graph\_data\_2[1])

    line\_velocity\_2.set\_data(pendulums.graph\_data\_2[0], pendulums.graph\_data\_2[2])

    velocity\_graph.set\_xlim([pendulums.graph\_data\_1[0][-1]-20, pendulums.graph\_data\_1[0][-1]])

    angle\_graph.set\_xlim([pendulums.graph\_data\_1[0][-1]-20, pendulums.graph\_data\_1[0][-1]])

    return line\_alpha\_1, line\_alpha\_2, line\_1, line\_2, line\_velocity\_1, line\_velocity\_2

def init():

    line\_alpha\_1.set\_data([], [])

    line\_alpha\_2.set\_data([], [])

    line\_1.set\_data([], [])

    line\_2.set\_data([], [])

    line\_velocity\_1.set\_data([], [])

    line\_velocity\_2.set\_data([], [])

    return line\_alpha\_1, line\_alpha\_2, line\_1, line\_2, line\_velocity\_1, line\_velocity\_2

interval = 1/FPS

anim = animation.FuncAnimation(screen, animate, frames=200,

                               interval=interval, blit=True, init\_func=init)

plt.tight\_layout()

plt.show()