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 12**

Выполнил(а) студент группы М8О-201Б-23

Ермеков Георгий Александрович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

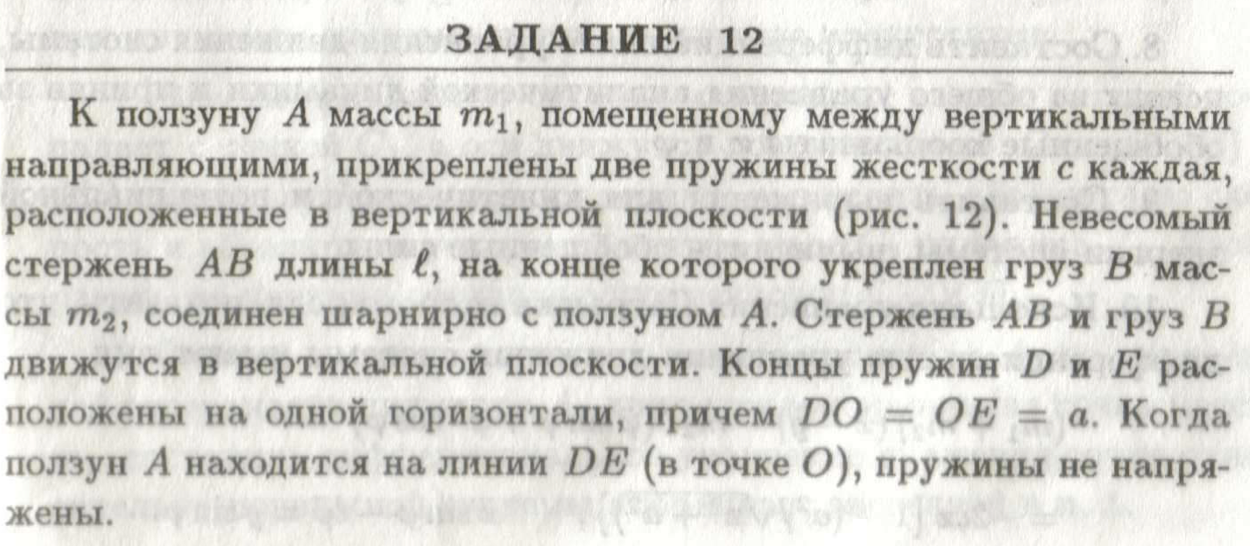
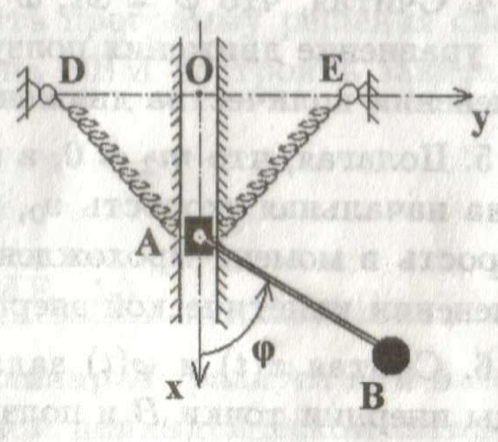
Проверил и принял

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

подпись, дата

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

Москва, 2024

**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.

**Программа**

import sympy as sp

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

# from matplotlib.patches import Rectangle

# from matplotlib.patches import Circle

# import math

from scipy.integrate import odeint

def formY(y, t, fv, fw):

    y1, y2, y3, y4 = y

    dydt = [y3, y4, fv(y1, y2, y3, y4), fw(y1, y2, y3, y4)]

    return dydt

Frames = 500

t = sp.Symbol("t")  # символ времени

x = sp.Function("x")(t)  # положение прямоугольника

fi = sp.Function("fi")(t)  # угол маятника

v = sp.Function("v")(t)  # скорость прямоугольника

w = sp.Function("w")(t)  # угловая скорость маятника

width = 1  # ширина ползуна

length = 2  # длина ползуна

circle\_radius = 0.2  # радиус шара

a = 1  # (DO и OE)

m1 = 1  # масса ползуна

m2 = 1  # масса шара

g = 9.8

pendulum\_l = 2  # длина маятника

k = 10  # коэф. жесткости

y0 = [-5, sp.rad(45), 0, 0]  # x(0), fi(0), v(0), w(0)

# кинетическая энергия ползуна

Ekin1 = (m1 \* v \* v) / 2

# квадратичная скорость центра масс

Vsquared = v \* v + w \* w \* pendulum\_l \* pendulum\_l - 2 \* v \* w \* pendulum\_l \* sp.sin(fi)

# кинетическая энергия маятника

pend\_kin = m2 \* Vsquared / 2

# кин. энергия системы

system\_kin = Ekin1 + pend\_kin

# потенциальная энергия

spring\_dx = sp.sqrt(a \* a + x \* x) - a  # изменение длины пружины

springs\_pot = k \* spring\_dx \* spring\_dx

system\_pot = -m1 \* g \* x - m2 \* g \* (x + pendulum\_l \* sp.cos(fi)) + springs\_pot

# generalized forces

Qx = -sp.diff(system\_pot, x)

Qfi = -sp.diff(system\_pot, fi)

# считаем лагранжиану

Lagr = system\_kin - system\_pot

ur1 = sp.diff(sp.diff(Lagr, v), t) - sp.diff(Lagr, x)

ur2 = sp.diff(sp.diff(Lagr, w), t) - sp.diff(Lagr, fi)

# метод крамера

a11 = ur1.coeff(sp.diff(v, t), 1)

a12 = ur1.coeff(sp.diff(w, t), 1)

a21 = ur2.coeff(sp.diff(v, t), 1)

a22 = ur2.coeff(sp.diff(w, t), 1)

b1 = -(ur1.coeff(sp.diff(v, t), 0)).coeff(sp.diff(w, t), 0).subs([(sp.diff(x, t), v), (sp.diff(fi, t), w)])

b2 = -(ur2.coeff(sp.diff(v, t), 0)).coeff(sp.diff(w, t), 0).subs([(sp.diff(x, t), v), (sp.diff(fi, t), w)])

detA = a11 \* a22 - a12 \* a21

detA1 = b1 \* a22 - b2 \* a21

detA2 = a11 \* b2 - b1 \* a21

dvdt = detA1 / detA

dwdt = detA2 / detA

# массив кадров

T = np.linspace(0, 50, Frames)

# переводим в лямдба функции

fv = sp.lambdify([x, fi, v, w], dvdt, "numpy")

fw = sp.lambdify([x, fi, v, w], dwdt, "numpy")

sol = odeint(formY, y0, T, args=(fv, fw))

# sol[:,0] - x

# sol[:,1] - fi

# sol[:,2] - v (dx/dt)

# sol[:,3] - w (dfi/dt)

ax = sp.lambdify(x, 0)

ay = sp.lambdify(x, x)

AX = ax(sol[:, 0])

AY = -ay(sol[:, 0])

bx = sp.lambdify(fi, pendulum\_l \* sp.sin(fi))

by = sp.lambdify([x, fi], +pendulum\_l \* sp.cos(fi) + x)

BX = bx(sol[:, 1])

BY = -by(sol[:, 0], sol[:, 1])

# Расчет R\_a

phi\_prime = sp.diff(fi, t)

phi\_double\_prime = sp.diff(phi\_prime, t)

Ra\_expr = m2 \* pendulum\_l \* (phi\_double\_prime \* sp.cos(fi) - phi\_prime\*\*2 \* sp.sin(fi))

Ra\_func = sp.lambdify([fi, phi\_prime, phi\_double\_prime], Ra\_expr, "numpy")

# Вычисление значений R\_a

phi\_prime\_vals = np.gradient(sol[:, 1], T)

phi\_double\_prime\_vals = np.gradient(phi\_prime\_vals, T)

Ra\_vals = Ra\_func(sol[:, 1], phi\_prime\_vals, phi\_double\_prime\_vals)

fig = plt.figure()

ax0 = fig.add\_subplot(1, 2, 1)

ax0.axis("equal")

L1X = [-width / 2, -width / 2]

L2X = [width / 2, width / 2]

LY = [min(AY) - length, max(AY) + length]

ax0.plot(L1X, LY, color="grey")  # левая стенка

ax0.plot(L2X, LY, color="grey")  # правая стенка

(sl,) = ax0.plot([-a, -length / 2], [0, AY[0] + width / 2], color="brown")  # левая пружина

(sr,) = ax0.plot([a, length / 2], [0, AY[0] + width / 2], color="brown")  # правая пружина

ax0.plot(-a, 0, marker=".", color="black")  # левое крепление пружины

ax0.plot(a, 0, marker=".", color="black")  # правое крепление пружины

rect = plt.Rectangle((-width / 2, AY[0]), width, length, color="black")  # ползун

circ = plt.Circle((BX[0], BY[0]), circle\_radius, color="grey")  # шарик

(R\_vector,) = ax0.plot([0, BX[0]], [0, BY[0]], color="grey")

# графики

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

ax2.plot(T, sol[:, 0])

ax2.set\_xlabel("t")

ax2.set\_ylabel("x")

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

ax3.plot(T, sol[:, 1])

ax3.set\_xlabel("t")

ax3.set\_ylabel("fi")

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

ax4.plot(T, Ra\_vals)

ax4.set\_xlabel("t")

ax4.set\_ylabel("Ra")

plt.subplots\_adjust(wspace=0.3, hspace=0.7)

def init():

    rect.set\_y(-length / 2)

    ax0.add\_patch(rect)

    circ.center = (0, 0)

    ax0.add\_patch(circ)

    return rect, circ

def anima(i):

    rect.set\_y(AY[i] - length / 2)

    sl.set\_data([-a, -width / 2], [0, AY[i]])

    sr.set\_data([a, width / 2], [0, AY[i]])

    R\_vector.set\_data([0, BX[i]], [AY[i], BY[i]])

    circ.center = (BX[i], BY[i])

    return (sl, sr, rect, R\_vector, circ)

anim = FuncAnimation(

    fig,

    anima,

    init\_func=init,

    frames=Frames,

    interval=10,

    blit=False,

    repeat=True,

    repeat\_delay=0,

)

plt.show()

**Тесты**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Вывод:** С помощью языка программирования Python и библиотек matplotlib и numpy было схематично симулировано движение маятника прикрепленного к грузу, который зажат между двумя стенками и прикреплен пружинами.