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-208Б-23

Ганяк Александр Олегович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

Зав. каф. 802, Волков Е.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

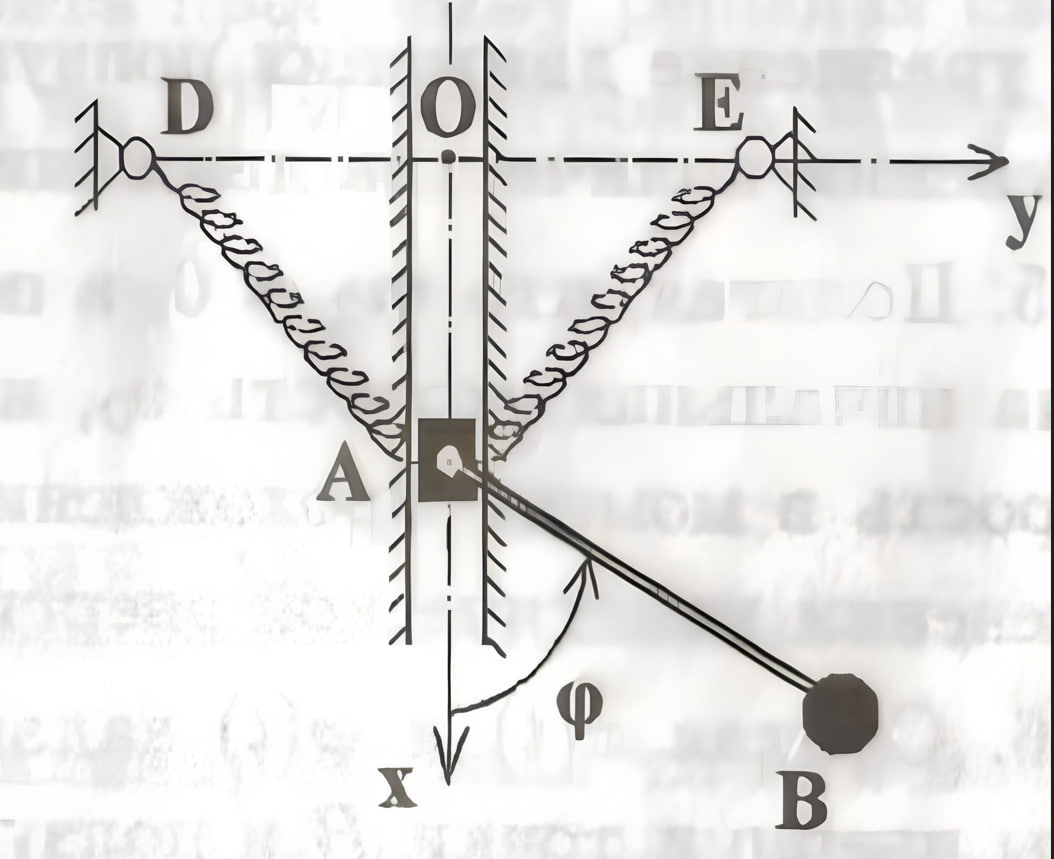
подпись, дата

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.

**Система:**



**Текст программы:**

**import sympy as sp**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from matplotlib.animation import FuncAnimation**

**from matplotlib.patches import Rectangle, Circle**

**from scipy.integrate import odeint**

***# Функция для вычисления системы дифференциальных уравнений***

**def diff\_eq(y, t, func\_v, func\_w):**

**y1, y2, y3, y4 = y**

**dydt = [y3, y4, func\_v(y1, y2, y3, y4), func\_w(y1, y2, y3, y4)]**

**return dydt**

***# Символьное объявление переменных***

**time\_sym = sp.Symbol("t")**

**x\_func = sp.Function('x')(time\_sym)**

**fi\_func = sp.Function('fi')(time\_sym)**

**v\_func = sp.Function('v')(time\_sym)**

**w\_func = sp.Function('w')(time\_sym)**

***# КОНФИГ***

**rect\_width = 1  *# ширина прямоугольника***

**rect\_length = 2  *# длина прямоугольника***

**circle\_radius = 0.2  *# радиус круга***

**spring\_distance = 1  *# расстояние до пружин***

**mass\_rect = 1  *# масса прямоугольника***

**mass\_circle = 1  *# масса груза***

**gravity = 9.8  *# ускорение свободного падения***

**rod\_length = 1  *# длина стержня***

**spring\_stiffness = 5  *# жесткость пружин***

**initial\_conditions = [0, 1, 0.5, 1]  *# начальные условия: x0, φ0, v0, w0***

***# Вычисление уравнений Лагранжа***

***# Определение коэффициентов матрицы инерции***

**a11 = mass\_rect + mass\_circle**

**a12 = -mass\_circle \* rod\_length \* sp.sin(fi\_func)**

**a21 = sp.sin(fi\_func)**

**a22 = -rod\_length**

***# Определение правой стороны уравнений движения***

**b1 = -2 \* spring\_stiffness \* x\_func \* (1 - (spring\_distance\*\*2 / sp.sqrt(x\_func\*\*2 + spring\_distance\*\*2))) + (mass\_rect + mass\_circle) \* gravity - w\_func\*\*2 \* sp.cos(fi\_func) \* mass\_circle \* rod\_length**

**b2 = gravity \* sp.sin(fi\_func)**

***# Вычисление детерминантов для решения системы уравнений***

**detA = a11 \* a22 - a12 \* a21**

**detA1 = b1 \* a22 - b2 \* a21**

**detA2 = a11 \* b2 - b1 \* a21**

***# Определение ускорений по x и φ***

**dvdt = detA1 / detA  *# Ускорение по x***

**dwdt = detA2 / detA  *# Ускорение по φ***

***# Временная сетка для решения уравнений***

**time\_grid = np.linspace(0, 50, 500)**

***# Преобразование символьных выражений в функции***

**func\_v = sp.lambdify([x\_func, fi\_func, v\_func, w\_func], dvdt, "numpy")**

**func\_w = sp.lambdify([x\_func, fi\_func, v\_func, w\_func], dwdt, "numpy")**

***# Решение системы дифференциальных уравнений***

**solution = odeint(diff\_eq, initial\_conditions, time\_grid, args=(func\_v, func\_w))**

***# Вычисление реакции вертикальных направляющих***

**solution\_ra = odeint(diff\_eq, initial\_conditions, time\_grid, args=(func\_v, func\_w))**

**reaction\_force = [mass\_circle \* rod\_length \* (solution\_ra[i, 3] \* np.cos(solution[i, 1]) - solution\_ra[i, 1]\*\*2 \* np.sin(solution[i, 1])) for i in range(len(solution))]**

***# Координаты точки A (центр прямоугольника)***

**AX = np.zeros\_like(solution[:, 0])**

**AY = -solution[:, 0]**

***# Координаты точки B (центр груза)***

**BX = rod\_length \* np.sin(solution[:, 1])**

**BY = -(rod\_length \* np.cos(solution[:, 1]) + solution[:, 0])**

***# Начало построения графиков и анимации***

**fig = plt.figure()**

**plt.suptitle('Графики и Симуляция', fontsize=16)**

***# Настройка оси для симуляции***

**ax\_sim = fig.add\_subplot(1, 2, 1)**

**ax\_sim.axis("equal")**

**ax\_sim.set\_title('Симуляция', fontsize=14)**

***# Подготовка данных для окружения***

**wall\_left\_x = [-rect\_width / 2, -rect\_width / 2]**

**wall\_right\_x = [rect\_width / 2, rect\_width / 2]**

**wall\_y = [min(AY) - rect\_length, max(AY) + rect\_length]**

***# Построение окружения***

**ax\_sim.plot(0, 0, marker=".", color="red")  *# красная точка***

**ax\_sim.plot(wall\_left\_x, wall\_y, linestyle='--', color="grey")  *# левая стена***

**ax\_sim.plot(wall\_right\_x, wall\_y, linestyle='--',  color="grey")  *# правая стена***

**spring\_left, = ax\_sim.plot([-spring\_distance, -rect\_length / 2], [0, AY[0] + rect\_width / 2], color="grey")  *# левая пружина***

**spring\_right, = ax\_sim.plot([spring\_distance, rect\_length / 2], [0, AY[0] + rect\_width / 2], color="grey")  *# правая пружина***

**ax\_sim.plot(-spring\_distance, 0, marker=".", color="black")  *# левое соединение***

**ax\_sim.plot(spring\_distance, 0, marker=".", color="black")  *# правое соединение***

**ax\_sim.axhline(0, linestyle=':', color='k')  *# горизонтальная пунктирная линия***

***# Создание прямоугольника и круга***

**rect = Rectangle((-rect\_width / 2, AY[0]), rect\_width, rect\_length, color="black")  *# прямоугольник***

**circ = Circle((BX[0], BY[0]), circle\_radius, color="grey")  *# круг***

***# Построение радиус-вектора точки B***

**radius\_vector, = ax\_sim.plot([0, BX[0]], [0, BY[0]], color="grey")**

***# Построение графиков для x(t), φ(t) и RA(t)***

**ax\_x = fig.add\_subplot(4, 2, 2)**

**ax\_x.plot(time\_grid, solution[:, 0], color="black")**

**ax\_x.set\_title('График x(t)', fontsize=14)**

**ax\_fi = fig.add\_subplot(4, 2, 4)**

**ax\_fi.plot(time\_grid, solution[:, 1], color="black")**

**ax\_fi.set\_title('График φ(t)', fontsize=14)**

**ax\_ra = fig.add\_subplot(4, 2, 6)**

**ax\_ra.plot(time\_grid, reaction\_force, color="black")**

**ax\_ra.set\_title('График RA(t)', fontsize=14)**

**plt.subplots\_adjust(wspace=0.4, hspace=0.8)**

***# Добавление подписей к точкам, грузу и ползунку***

**ax\_sim.text(-spring\_distance - 0.2, 0, 'D', ha='right', va='bottom')**

**ax\_sim.text(spring\_distance + 0.2, 0, 'E', ha='left', va='bottom')**

**text\_A = ax\_sim.text(0, AY[0] - 0.2, 'A', ha='right', va='bottom')**

**text\_B = ax\_sim.text(BX[0] + 0.2, BY[0], 'B', ha='right', va='bottom')**

***# Функция для инициализации позиций***

**def init():**

**rect.set\_y(-rect\_length / 2)**

**ax\_sim.add\_patch(rect)**

**circ.center = (0, 0)**

**ax\_sim.add\_patch(circ)**

**return rect, circ**

***# Функция для создания пружины***

**def spring(start, end, num\_segments=6, amplitude=0.1):**

**x\_vals = np.linspace(start[0], end[0], num\_segments)**

**y\_vals = np.linspace(start[1], end[1], num\_segments)**

**dist = np.sqrt((end[0] - start[0])\*\*2 + (end[1] - start[1])\*\*2)**

**amp\_factor = amplitude \* (2 + 0.5 \* dist)  *# Изменение амплитуды пружины в зависимости от расстояния***

**for i in range(1, num\_segments, 2):**

**y\_vals[i] += amp\_factor**

**return x\_vals, y\_vals**

***# Функция для обновления позиций на каждом кадре анимации***

**def animate(i):**

**rect.set\_y(AY[i] - rect\_length / 2)**

**spring\_left\_x, spring\_left\_y = spring((-spring\_distance, 0), (-rect\_width / 2, AY[i]), num\_segments=12, amplitude=0.1)**

**spring\_right\_x, spring\_right\_y = spring((spring\_distance, 0), (rect\_width / 2, AY[i]), num\_segments=12, amplitude=0.1)**

**spring\_left.set\_data(spring\_left\_x, spring\_left\_y)**

**spring\_right.set\_data(spring\_right\_x, spring\_right\_y)**

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

**circ.center = (BX[i], BY[i])**

***# Обновление позиций текстовых меток для ползунка и груза***

**text\_A.set\_position((-1, AY[i] - 0.2))  *# Ползунок A***

**text\_B.set\_position((BX[i] + 0.2, BY[i] + 0.2))  *# Груз B***

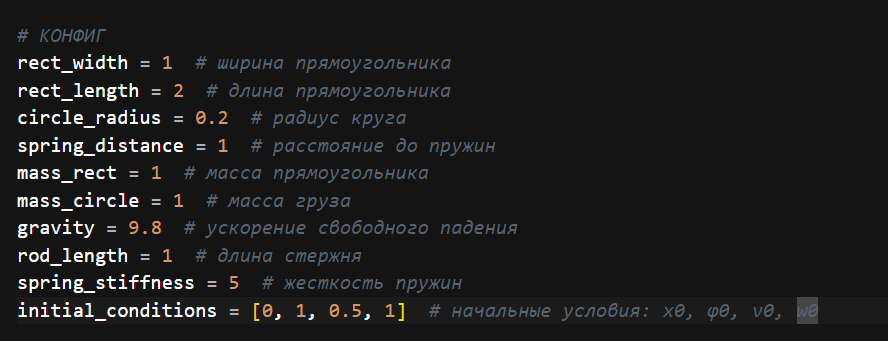
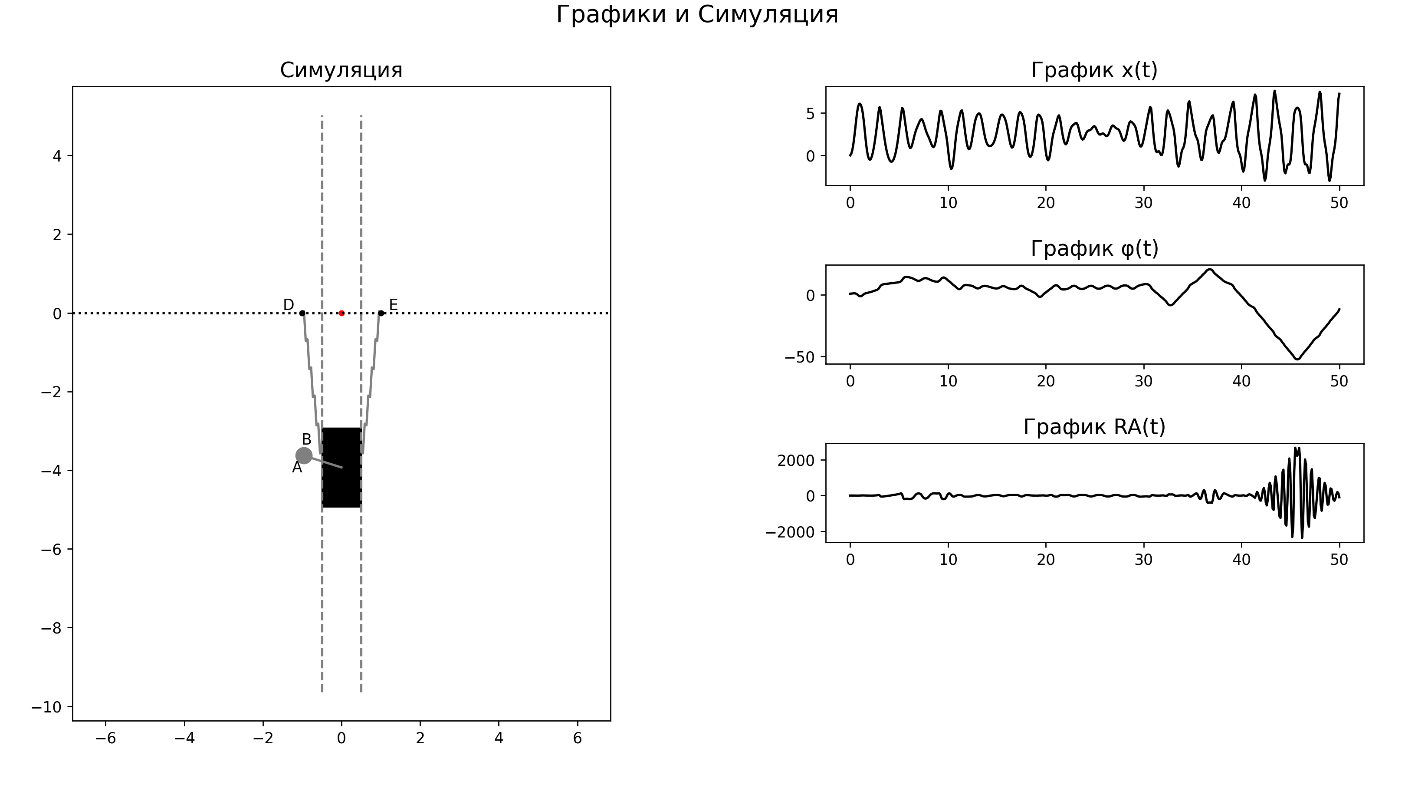
**return spring\_left, spring\_right, rect, radius\_vector, circ**

***# Запуск анимации***

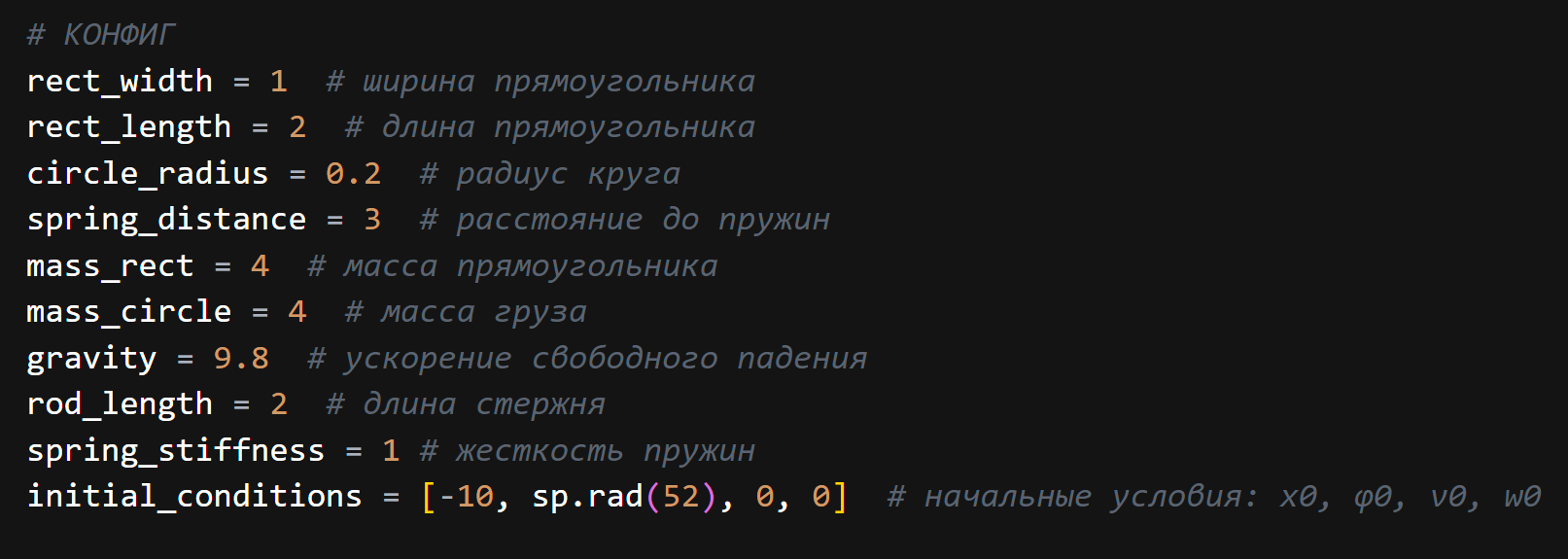
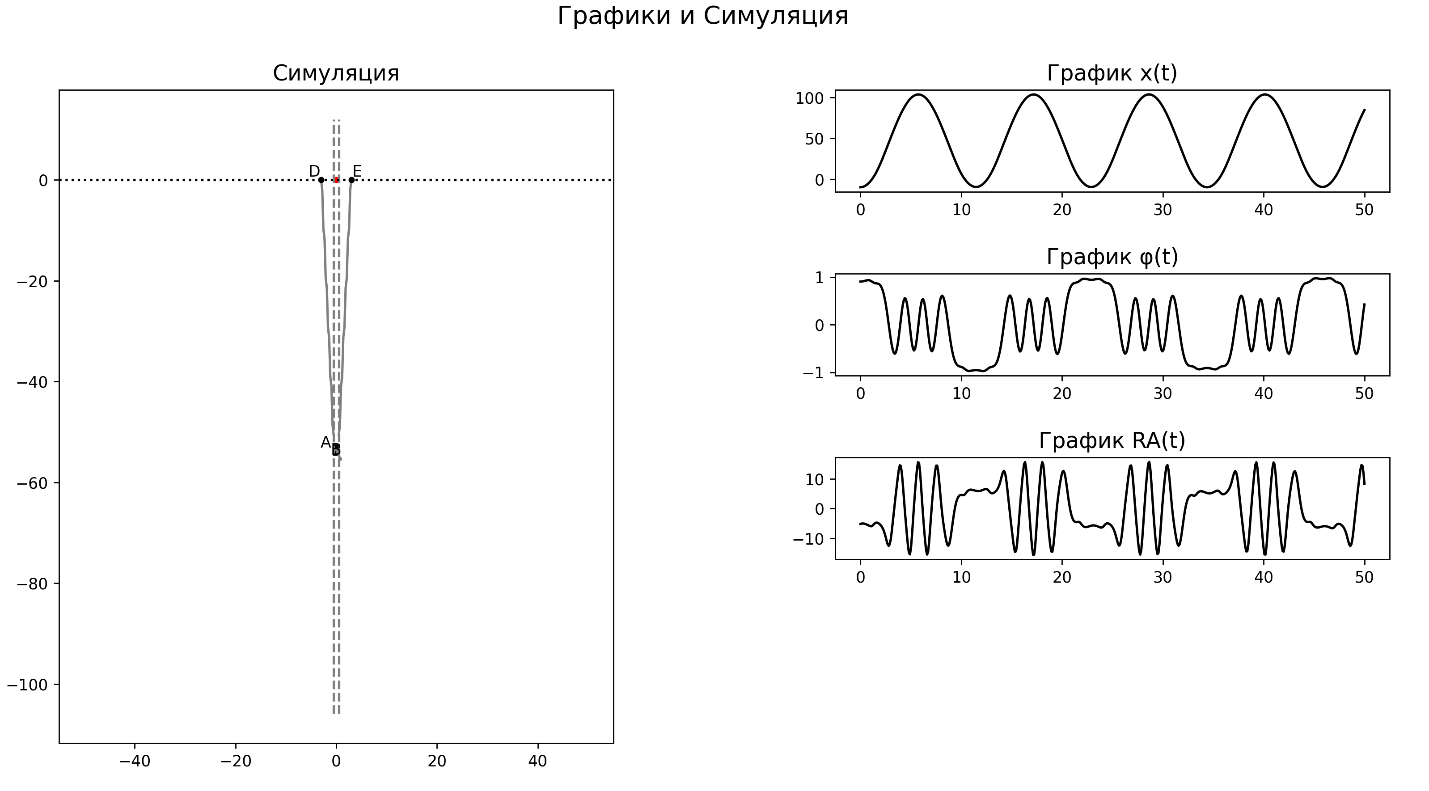
**anim = FuncAnimation(fig, animate, init\_func=init, frames=500, blit=False, repeat=True, repeat\_delay=0)**

**plt.show()**

**Результаты**

1)**По условию задачи:**

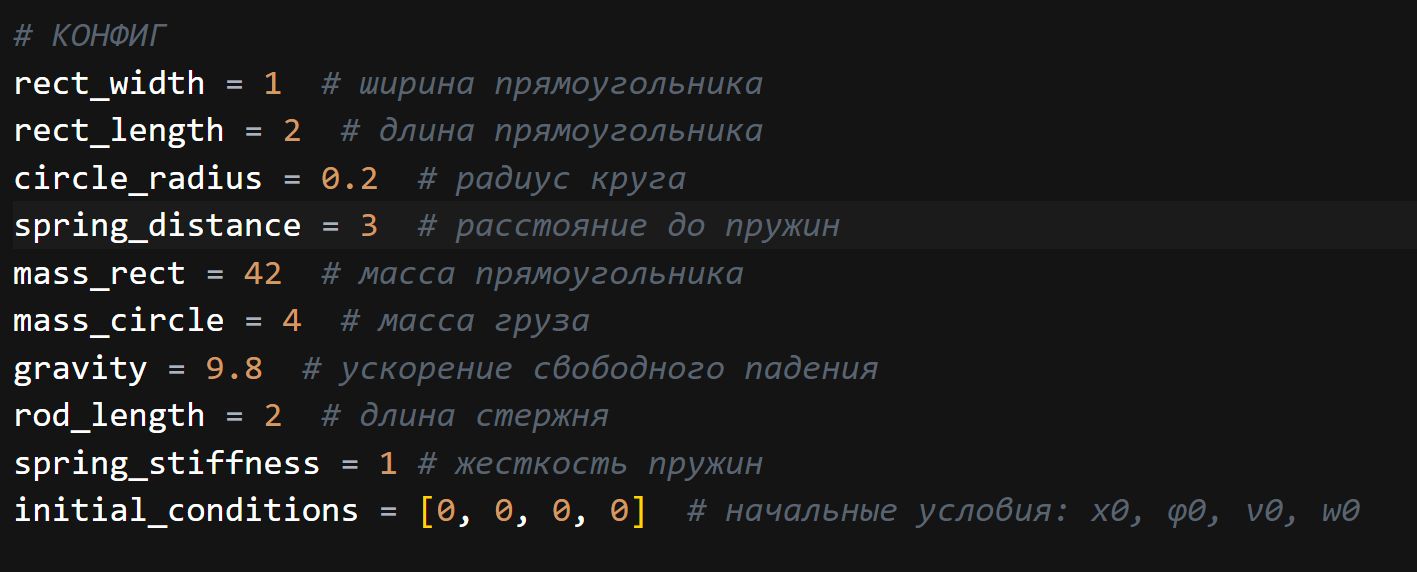
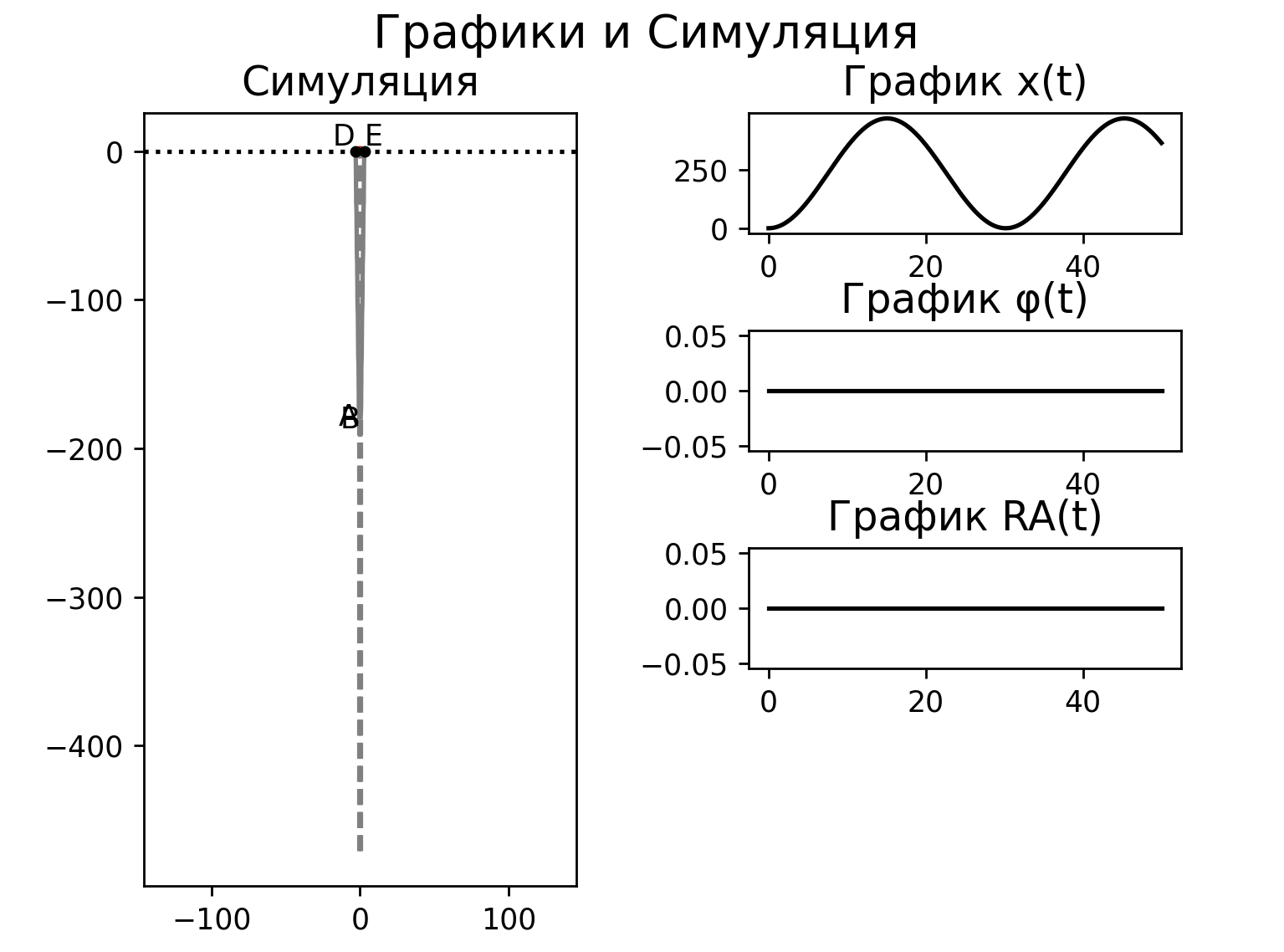
**Получили:** Ползунок A совершает вертикальные колебательные движения благодаря взаимодействию с двумя пружинами, закрепленными в точках D и E. Маятник B, прикрепленный к ползунку, совершает сложные вращательные движения относительно горизонтальной оси. При заданных начальных данных маятник колеблется около верхнего положения.

2) **Условие:**

**Получили:** При увеличении масс прямоугольника и маятника, а также снижении жесткости пружины, наблюдается аналогичная тенденция. Тело (ползунок A) начинает колебаться преимущественно около нижнего положения равновесия. Маятник (В) также демонстрирует движение, ограниченное областью своего нижнего положения равновесия, изредка поднимаясь вверх под действием силы инерции, но в значительно меньшей амплитуде из-за увеличенной массы и сниженной упругости.

Общая динамика системы показывает затухающие, плавные движения, которые сосредоточены вокруг нижнего положения равновесия, что связано с перераспределением энергий из-за заданных начальных условий и параметров системы.

3) Условие



**Получили:** Существенное увеличение массы прямоугольника и сохранение малой жесткости пружины приводит к тому, что система становится практически статичной. Ползунок A остается неподвижным в нижнем положении, а маятник B вообще не колеблется. Это демонстрирует, что при таких параметрах инерция и гравитация полностью подавляют динамику системы.

**Вывод**

В процессе выполнения данной лабораторной работы я успешно реализовал все поставленные задачи. С использованием языка программирования Python и библиотек (matplotlib, numpy и scipy), мне удалось смоделировать и анимировать движение сложной механической системы, включающей прямоугольный ползунок, две пружины и маятник.

Работа включала решение системы дифференциальных уравнений, описывающих динамику системы, построение графиков зависимостей её параметров, а также создание анимации, которая позволила наглядно продемонстрировать поведение системы.

Данная лабораторная работа помогла мне лучше изучить matplotlib и scipy, а также освоить методы численного решения задач механики.

Мною был получен большой опыт для дальнейшего обучения.