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 №31

Выполнил(а) студент группы М8О-208Б-23

Пинчук Михаил Сергеевич ______________________________

подпись, дата

Проверил и принял

Ст. преп. каф. 802 Волков Е.В.______________________________

подпись, дата

с оценкой ______________________________

Москва, 2024

Вариант №31

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.

Механическая система:

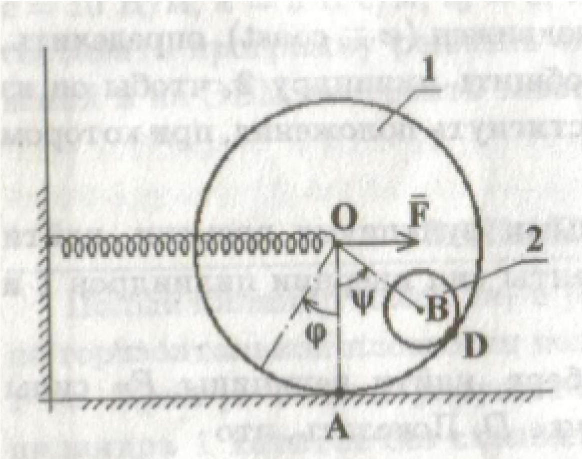


Рис. 31

Текст программы

```
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from scipy.integrate import odeint

# ======================
# Функция для системы дифференциальных уравнений
# ======================
def system_of_equations(state, time, F_amplitude, mass_main, mass_sub,
damping_coef, frequency, gravity):
    """
    Система дифференциальных уравнений для модели.
    """
    derivatives = np.zeros(4)
    angle_main, angle_sub, velocity_main, velocity_sub = state
```

```
    derivatives[0] = velocity_main  # d(angle_main)/dt
    derivatives[1] = velocity_sub   # d(angle_sub)/dt

    # Коэффициенты уравнений (можно интерпретировать как элементы
матрицы масс/моментов)
    matrix_a11 = mass_sub * (radius_main - radius_sub) * (1 +
np.cos(angle_main))
    matrix_a12 = 2 * radius_main * (mass_main + mass_sub)
    matrix_a21 = 2 * (radius_main - radius_sub)
    matrix_a22 = radius_main * (1 + np.cos(angle_main))

    # Правая часть системы уравнений
    rhs1 = (F_amplitude * np.sin(time * frequency)
            - damping_coef * radius_main * velocity_sub
            + mass_sub * (radius_main - radius_sub) * velocity_main**2 *
np.sin(angle_main))
    rhs2 = -gravity * np.sin(angle_main)

    # Решение системы линейных уравнений A * d(velocity)/dt = rhs
    det = (matrix_a11 * matrix_a22 - matrix_a12 * matrix_a21)
    derivatives[2] = (rhs1 * matrix_a22 - rhs2 * matrix_a12) / det
    derivatives[3] = (rhs2 * matrix_a11 - rhs1 * matrix_a21) / det

    return derivatives

# =======================
# Константы системы
# =======================
mass_main = 1.0
mass_sub  = 1.0
radius_main = 1.0
radius_sub  = 0.5
F_amplitude = 4.0
frequency   = 3 * math.pi   # Ещё выше частота
damping_coef = 2.0
gravity = 9.81

# Параметры времени
time_steps = 1000
time_final = 20
time_array = np.linspace(0, time_final, time_steps)

# Начальные условия (углы в радианах и соответствующие скорости)
initial_state = [2.0, -1.0, 5.0, -2.0]

# =======================
# Решение системы
# =======================
solution = odeint(system_of_equations, initial_state, time_array,
                  args=(F_amplitude, mass_main, mass_sub, damping_coef,
```

```
frequency, gravity))

angle_main = solution[:, 0]  # Угол основного цилиндра
angle_sub  = solution[:, 1]  # Угол дополнительного цилиндра

# Предположим, что "длина пружины" (или нечто аналогичное) меняется как
2.5 + angle_main + radius_main
spring_length = 2.5 + angle_main + radius_main

# Координаты основного цилиндра
X_main = spring_length
Y_main = radius_main

# Координаты дополнительного цилиндра
X_sub = X_main + (radius_main - radius_sub) * np.sin(angle_sub)
Y_sub = Y_main - (radius_main - radius_sub) * np.cos(angle_sub)

# =======================
# Графики решений
# =======================
fig_graphs, axes = plt.subplots(2, 2, figsize=(13, 7))

axes[0, 0].plot(time_array, angle_main, color='blue')
axes[0, 0].set_title("Угол основного цилиндра (psi)")
axes[0, 0].grid(True)

axes[1, 0].plot(time_array, angle_sub, color='red')
axes[1, 0].set_title("Угол дополнительного цилиндра (phi)")
axes[1, 0].grid(True)

axes[0, 1].plot(time_array, np.sin(angle_main), color='orange')
axes[0, 1].set_title("Синус основного угла")
axes[0, 1].grid(True)

axes[1, 1].plot(time_array, np.cos(angle_sub), color='black')
axes[1, 1].set_title("Косинус дополнительного угла")
axes[1, 1].grid(True)

# =======================
# Анимация системы
# =======================
fig_anim, ax_anim = plt.subplots(figsize=(8, 6))
ax_anim.axis('equal')
ax_anim.set(xlim=[0, 8], ylim=[-1, 5])
ax_anim.set_title("Анимация двух цилиндров с пружиной")

# Статические элементы (земля, опоры и т.д.)
ground = ax_anim.plot([0, 0, 6], [2, 0, 0], color='black', linewidth=2)[0]

# Переобъявим "холостые" объекты, которые будет обновлять анимация
```

```
main_cylinder, = ax_anim.plot([], [], color='blue', linewidth=3)  # Основной цилиндр
sub_cylinder,  = ax_anim.plot([], [], color='red',  linewidth=2)  # Дополнительный цилиндр
spring_line,   = ax_anim.plot([], [], color='green',linewidth=2)  # Пружина (будет синусоидой)
point_main,    = ax_anim.plot([], [], 'bo', markersize=8)         # Точка центра осн. цилиндра

# Координаты кругов для отрисовки цилиндров
circle_theta = np.linspace(0, 2 * np.pi, 100)
X_circle_main = radius_main * np.cos(circle_theta)
Y_circle_main = radius_main * np.sin(circle_theta)
X_circle_sub  = radius_sub  * np.cos(circle_theta)
Y_circle_sub  = radius_sub  * np.sin(circle_theta)

def create_spring(x1, y1, x2, y2, n_coils=6, amplitude=0.1, resolution=100):
    """
    Генерирует массив координат X, Y, рисующий синусоидальную "пружину"
    между точками (x1, y1) и (x2, y2).

    Параметры:
    ----------
    x1, y1 : float
        Координаты начальной точки.
    x2, y2 : float
        Координаты конечной точки.
    n_coils : int
        Количество полувитков (или число «горбов») синусоиды.
    amplitude : float
        Амплитуда отклонения пружины (в единицах координат).
    resolution : int
        Количество точек вдоль пружины (чем больше, тем плавнее).

    Возвращает:
    -----------
    X_spring, Y_spring : ndarray
        Координаты точек вдоль синусоиды (пружины).
    """
    # Параметр t идёт от 0 до 1
    t = np.linspace(0, 1, resolution)

    # Линейная интерполяция координат (по оси X и Y)
    X_straight = x1 + (x2 - x1) * t
    Y_straight = y1 + (y2 - y1) * t

    # Найдём длину прямой между (x1,y1) и (x2,y2)
    dx = x2 - x1
    dy = y2 - y1
    length = np.hypot(dx, dy)
```

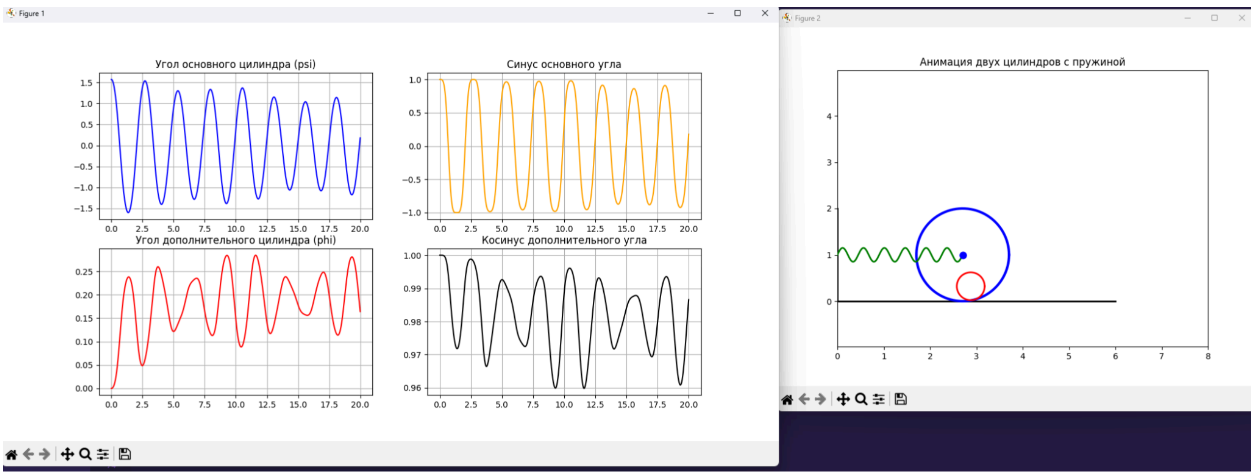
```
    # Пружина будет колебаться перпендикулярно прямой линии
    # Для этого найдём единичный вектор "перпендикуляр" к (dx, dy)
    # Поворот вектора (dx, dy) на 90 градусов => (-dy, dx)
    # Нормируем:
    perp_norm = np.array([-dy, dx])
    perp_len = np.hypot(perp_norm[0], perp_norm[1])
    if perp_len != 0:
        perp_norm /= perp_len
    else:
        # На случай, если x1,y1 == x2,y2
        perp_norm = np.array([0, 1])

    # Синусоида вдоль оси t, растянута на n_coils полуволн
    # full_waves = n_coils/2 (если говорить о «полуволнах» vs «полных
волнах» -- на вкус)
    # но будем считать n_coils полуволнами -- значит частота = 2*pi*n_coils
    wave = amplitude * np.sin(2 * np.pi * n_coils * t)

    # Координаты пружины = координаты прямой + колебания по
перпендикуляру
    X_spring = X_straight + wave * perp_norm[0]
    Y_spring = Y_straight + wave * perp_norm[1]

    return X_spring, Y_spring

def animate(frame):
    """
    Анимирует кадр с номером frame.
    """
    # Координаты основного цилиндра на данном кадре
    current_X_main = X_main[frame]
    current_Y_main = Y_main  # В коде Y_main не меняется во времени

    # Координаты дополнительного цилиндра на данном кадре
    current_X_sub = X_sub[frame]
    current_Y_sub = Y_sub[frame]

    # --- Обновляем основной цилиндр ---
    main_cylinder.set_data(X_circle_main + current_X_main,
                           Y_circle_main + current_Y_main)

    # --- Обновляем дополнительный цилиндр ---
    sub_cylinder.set_data(X_circle_sub + current_X_sub,
                          Y_circle_sub + current_Y_sub)

    # --- Обновляем точку центра основного цилиндра ---
    point_main.set_data([current_X_main], [current_Y_main])

    # --- Генерируем координаты «пружины» ---
```

```
    # В нашем случае она рисуется горизонтально от (0, Y_main) до (X_main,
Y_main).
    # Если хотите, чтобы она "шла" по тому же уровню, меняющемуся во
времени, можно
    # использовать разные Y-координаты, но ниже оставим её горизонтальной.
    spring_x, spring_y = create_spring(0, current_Y_main,
                                   current_X_main, current_Y_main,
                                   n_coils=6, amplitude=0.15,
resolution=150)
    spring_line.set_data(spring_x, spring_y)

    return main_cylinder, sub_cylinder, spring_line, point_main

# Создаем анимацию
anim = FuncAnimation(fig_anim, animate, frames=len(time_array), interval=30,
blit=True)

# Показываем графики и анимацию
plt.show()
```

Результат работы программы:

- mass_main = 5.0; mass_sub = 0.5; radius_main = 1.0; radius_sub = 0.3; F_amplitude = 3.0; frequency = math.pi; damping_coef = 5.0; gravity = 9.81;

initial_state = [math.pi / 2, 0, 0, 0]; – лёгкий дополнительный цилиндр, умеренное демпфирование:



Результат: Основной цилиндр (более тяжёлый) колеблется около вертикального положения, отклоняясь под действием внешней силы.

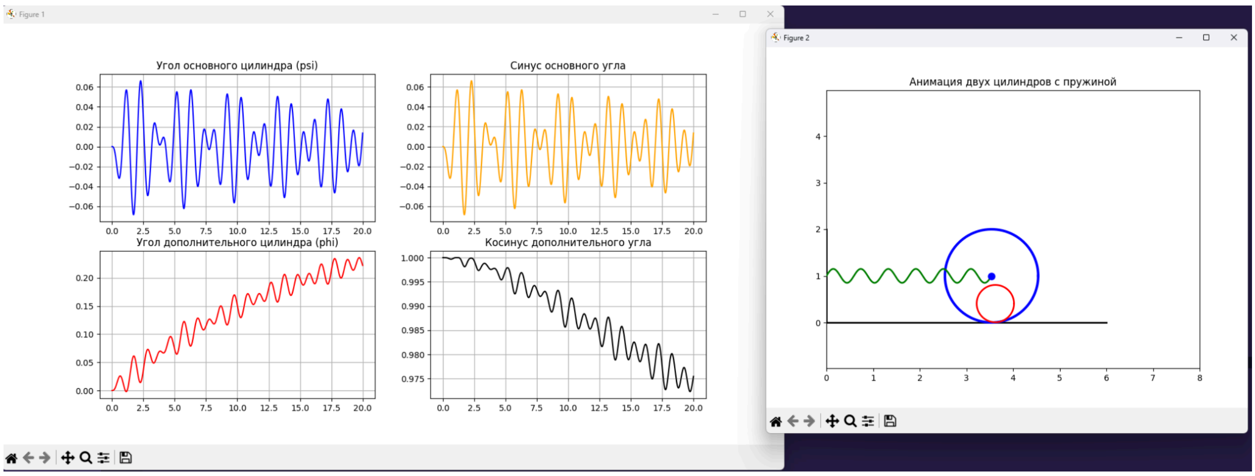
Лёгкий дополнительный цилиндр (mass_sub = 0.5) будет раскачиваться заметно сильнее и «гулять» по радиусу основного.

Демпфирование (damping_coef = 5) не слишком большое, поэтому колебания затухают медленно.

Пружина (модельно) чуть «трясет» основной цилиндр взад-вперед, создавая колебания в горизонтальном направлении.

- mass_main = 3.0; mass_sub = 5.0; radius_main = 1.0; radius_sub = 0.4; F_amplitude = 2.0; frequency = 2 * math.pi; damping_coef = 1.0; gravity = 9.81;

initial_state = [0, 0, 0, 0]; – тяжёлый «дополнительный» цилиндр, слабый демпфер:



Результат: Поскольку дополнительный цилиндр тяжелее основного, система становится нестабильной при минимальном внешнем воздействии.

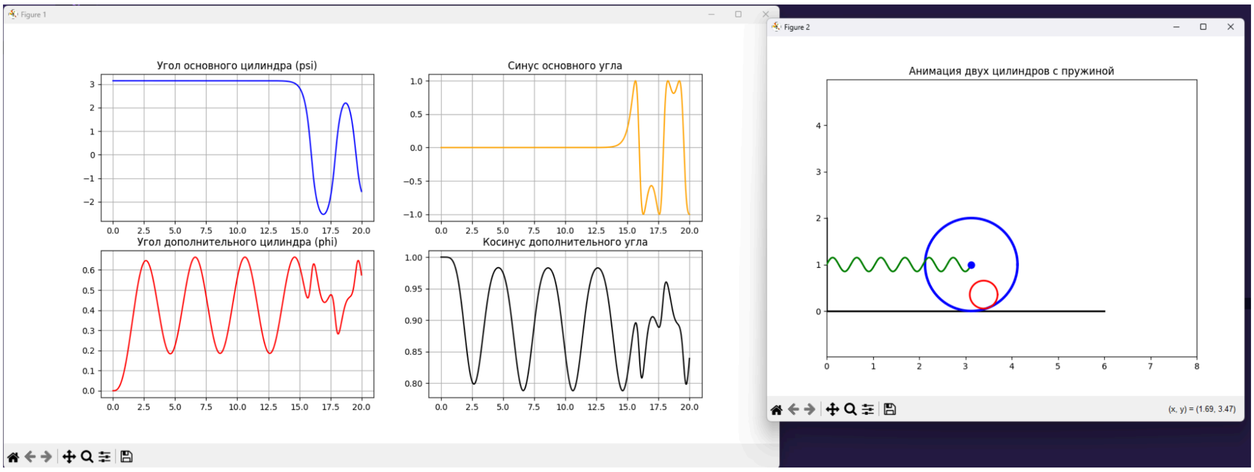
При слабом демпфировании (damping_coef = 1) пружинные колебания почти не гасятся, возможны крупные размахи.

Основной цилиндр будет стремиться вращаться, но значительная масса дополнительного может вывести систему из равновесия.

Из-за более высокой частоты внешней силы (2π) могут появиться эффекты резонанса, если система колебаний попадёт в подходящий диапазон.

- mass_main = 5.0; mass_sub = 2.0; radius_main = 1.0; radius_sub = 0.3; F_amplitude = 10.0; frequency = 2 * math.pi; damping_coef = 15.0; gravity = 9.81;

initial_state = [math.pi, 0, 0, 0] – сильное демпфирование, большая внешняя сила:



Результат: Из-за большой внешней силы (F_amplitude = 10) цилиндры будут пытаться сильно раскачиваться.

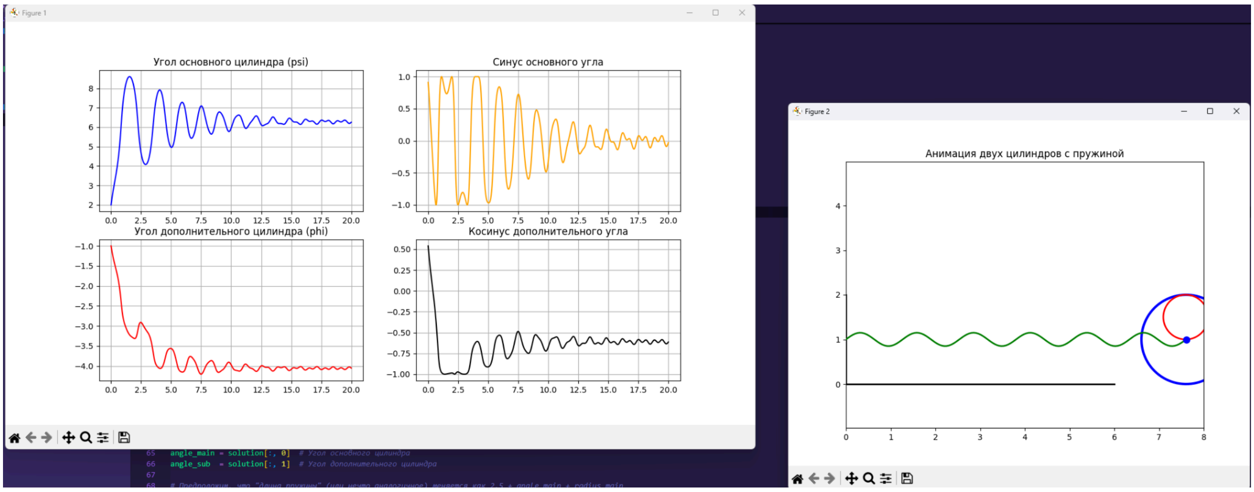
Однако высокое демпфирование (damping_coef = 15) будет быстро гасить любые колебания.

Основной цилиндр, начав с перевёрнутого положения (угол ~ 180°), будет пытаться вернуться в более устойчивое состояние (около 0 или π), совершая пару «рывков».

Дополнительный цилиндр может слегка «подвисать» или «подпрыгивать» в радиальной полости, но скоро тоже стабилизируется.

- mass_main = 1.0; mass_sub = 1.0; radius_main = 1.0; radius_sub = 0.5; F_amplitude = 4.0; frequency = 3 * math.pi; damping_coef = 2.0; gravity = 9.81;

initial_state = [2.0, -1.0, 5.0, -2.0]; – маленькая основная масса, сильные пружинные колебания, большие начальные углы:



Результат: Пример «бурных» колебаний: оба цилиндра легкие (одинаковые массы), начальные углы и скорости заданы так, что сразу возникает сильная динамика.

Высокая частота внешней силы (3π) может «разгонять» систему и приводить к резким пульсациям вокруг оси.

Небольшой коэффициент демпфирования (2.0) позволяет этим колебаниям довольно долго сохранять энергию, хотя и будет заметное затухание с течением времени.

Пружина визуально будет сильно «шевелиться», переходя из сжатого в растянутое состояние, цилиндры могут делать почти полный оборот вокруг своих центров.

Вывод:

В ходе выполнения этой лабораторной работы я написал Python код, строящий анимацию движения системы, уравнения движения которой могут быть модифицированы путем изменения коэффициентов (начальных значений).

Также я реализовал 4 графика, которые отображают изменение x(t), phi(t), FA(t) (силы трения) и NA(t) (силы давления системы на плоскость).