ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ОТЧЕТ
О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №2
ПО ДИСЦИПЛИНЕ
«ОСНОВЫ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ ДИНАМИЧЕСКИХ СИСТЕМ»
ВАРИАНТ ЗАДАНИЯ №25

Выполнил студент группы М8О-206Б-23
Сергеев В. С.______________________________
подпись, дата
Проверил и принял
Ст. преп. каф. 802 Волков Е.В.______________________________
подпись, дата
с оценкой ______________________________

Москва, 2024

Вариант №25

Задание:

Реализовать анимацию движения механической системы.

Механическая система:

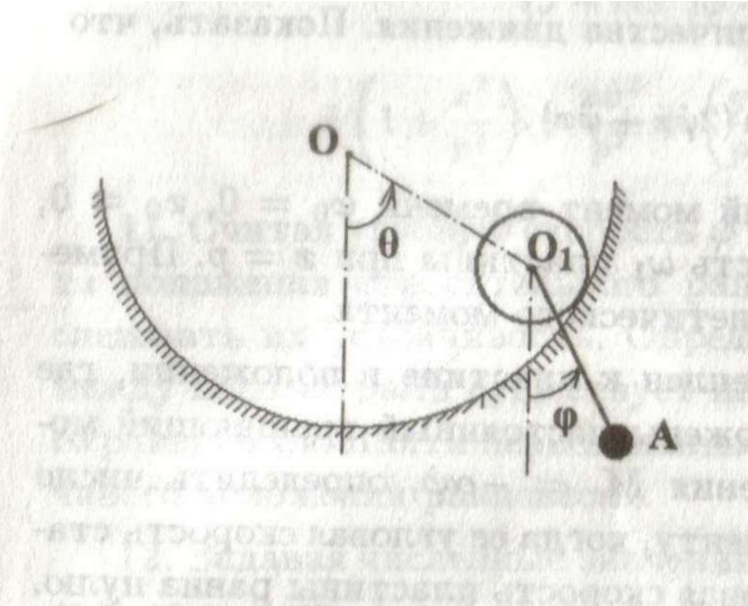
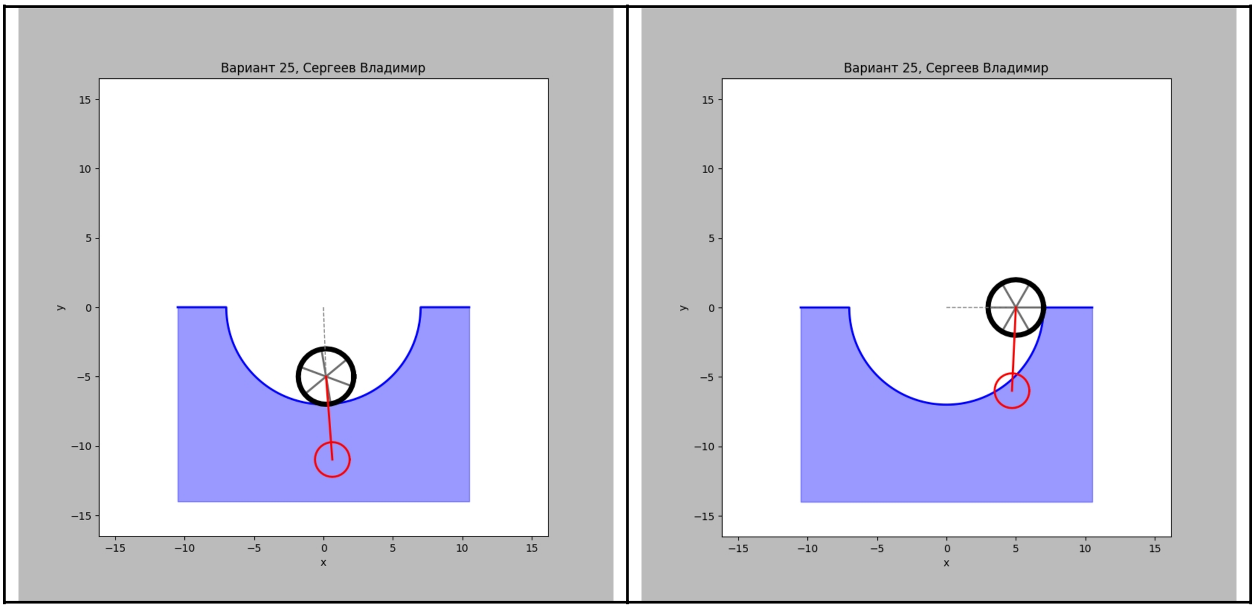


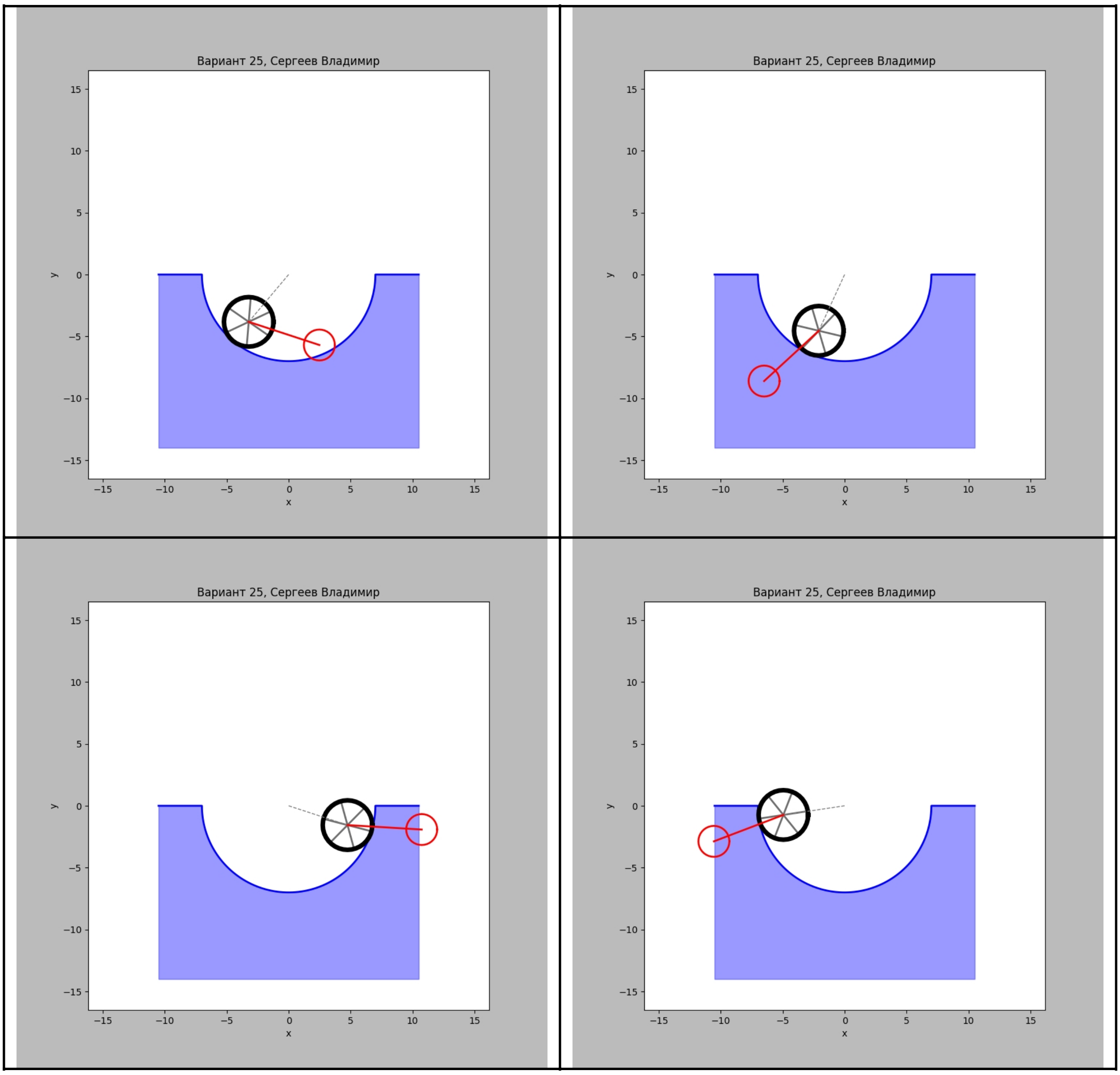
Рис. 25

Текст программы

Текст программы в приложении №1.

Результат работы программы:





Вывод:

В ходе выполнения лабораторной работы была написана программа на языка Python, для симуляции и визуализации движения системы тел в соответствии с заданными законами движения.

Приложение №1

```
import numpy as np
from matplotlib import use, pyplot as plt
from matplotlib.animation import FuncAnimation


class Wheel:
    """Class for displaying a wheel."""

    def __init__(self, radius, position, phase=0, spokes_count=2, precision=360,
                 tire_color='black', spokes_color='gray', tire_width=5, spokes_width=2):

        self.spokes_count = spokes_count
        self.radius = radius

        self.spoke_functions = (
            (lambda x, phase, number:
             np.array([
                 x + self.radius * np.cos(phase + np.pi / self.spokes_count * number),
                 x - self.radius * np.cos(phase + np.pi / self.spokes_count * number)
             ])
             ),
            (lambda y, phase, number:
             np.array([
                 y + self.radius * np.sin(phase + np.pi / self.spokes_count * number),
                 y - self.radius * np.sin(phase + np.pi / self.spokes_count * number)
             ])
             )
        )

        self.spokes = []
        for i in range(self.spokes_count):
            self.spokes.append(region.plot(
                self.spoke_functions[0](position[0], phase, i),
                self.spoke_functions[1](position[1], phase, i),
                color=spokes_color, lw=spokes_width
            )[0])

        angles = np.linspace(0, 2 * np.pi, precision + 1)
        self.tire_values = self.radius * np.cos(angles), self.radius * np.sin(angles)

        self.tire, = region.plot(self.tire_values[0] + position[0], self.tire_values[1] + position[1],
                                 color=tire_color, lw=tire_width)

    def update(self, position, phase):
        for i in range(self.spokes_count):
            self.spokes[i].set_data(
                self.spoke_functions[0](position[0], phase, i),
                self.spoke_functions[1](position[1], phase, i)
            )

        self.tire.set_data(self.tire_values[0] + position[0], self.tire_values[1] + position[1])

    def return_plot(self):
        return *self.spokes, self.tire,


class Pendulum:
    """Class for displaying a pendulum."""

    def __init__(self, weight_radius, suspension_position, weight_position, precision=360,
                 color='red', line_width=2):
        angles = np.linspace(0, 2 * np.pi, precision + 1)
        self.weight_values = weight_radius * np.cos(angles), weight_radius * np.sin(angles)
        self.weight, = region.plot(
            self.weight_values[0] + weight_position[0],
            self.weight_values[1] + weight_position[1],
            color=color, lw=line_width
        )
        self.rod, = region.plot(
            np.array([suspension_position[0], weight_position[0]]),
            np.array([suspension_position[1], weight_position[1]]),
            color=color, lw=line_width
        )

    def update(self, weight_position, suspension_position):
        self.weight.set_data(
            self.weight_values[0] + weight_position[0],
```

```
            self.weight_values[1] + weight_position[1]
        )
        self.rod.set_data(
            np.array([suspension_position[0], weight_position[0]]),
            np.array([suspension_position[1], weight_position[1]]),
        )

    def return_plot(self):
        return self.weight, self.rod,


class Cord:
    """Class for displaying a cord."""

    def __init__(self, start, end, color='gray', line_style='--', line_width=1):
        self.plot, = region.plot(
            np.array([start[0], end[0]]),
            np.array([start[1], end[1]]),
            color=color, linestyle=line_style, lw=line_width
        )

    def update(self, start, end):
        self.plot.set_data([start[0], end[0]], [start[1], end[1]])

    def return_plot(self):
        return self.plot,


def make_concave_platform(radius, position, precision=180, color='blue'):
    """Function for displaying a concave platform."""

    angles = np.linspace(-np.pi, 0, precision)
    plot_values = (
        np.concatenate((
            [-1.5 * radius, -radius], radius * np.cos(angles), [radius, 1.5 * radius]
        )) + position[0],
        np.concatenate((
            [0, 0], radius * np.sin(angles), [0, 0]
        )) + position[1]
    )

    region.plot(
        plot_values[0], plot_values[1],
        color=color, lw=2
    )

    region.fill_between(
        plot_values[0],
        plot_values[1],
        np.full(precision + 4, - 2 * radius),
        alpha=.4, color=color
    )


if __name__ == '__main__':
    # --- Simulation settings ---

    platform_position = (0, 0)  # Position of concave platform
    platform_radius = 5  # Radius of concave platform

    wheel_radius = 2  # Radius of wheel
    wheel_frequency = .8  # Oscillation frequency of wheel

    pendulum_length = 6  # Length of pendulum rod
    pendulum_frequency = 1.6  # Oscillation frequency of pendulum

    MAX_TIME = 30  # Maximum simulation time
    STEPS = 1000  # Count of time points

    FPS = 60  # Frames per second

    # --- Calculations ---

    time_points = np.linspace(0, MAX_TIME, STEPS)

    # Angles of the wheel's deviation from the vertical
    wheel_angles = np.pi * (np.sin(wheel_frequency * time_points) - 1) / 2

    # Wheel locations at each time point
```

```python
wheel_positions = (
    platform_position[0] + platform_radius * np.cos(wheel_angles),
    platform_position[1] + platform_radius * np.sin(wheel_angles)
)

gear_ratio = -(platform_radius / wheel_radius)

# Phases of rotation of the wheel
wheel_phases = wheel_angles * gear_ratio

# Angles of the pendulum's deviation from the vertical
pendulum_angles = np.pi * (np.sin(pendulum_frequency * time_points) - 1) / 2

# Pendulum locations at each time point
pendulum_positions = (
    wheel_positions[0] + pendulum_length * np.cos(pendulum_angles),
    wheel_positions[1] + pendulum_length * np.sin(pendulum_angles)
)

# --- Rendering ---

use('QtAgg')

window = plt.figure(facecolor='#bbbbbb')
region = window.add_subplot(1, 1, 1)
region.set_title("Вариант 25, Сергеев Владимир")

x_axis_limit = 1.5 * np.max(np.array([np.max(np.abs(pendulum_positions[0])), np.max(np.abs(wheel_positions[0]))]))
y_axis_limit = 1.5 * np.max(np.array([np.max(np.abs(pendulum_positions[1])), np.max(np.abs(wheel_positions[1]))]))
region.set_xlim(-x_axis_limit, x_axis_limit)
region.set_ylim(-y_axis_limit, y_axis_limit)
# We will set the value to "image" in order to comply with the established limits and avoid distortions.
region.axis('image')

make_concave_platform(platform_radius + wheel_radius, platform_position, precision=180)
wheel = Wheel(wheel_radius, (0, 0), 0, spokes_count=3, precision=36, spokes_color='#6e6e6e')
cord = Cord(platform_position, (0, 0))  # Cord from the platform position to a wheel position
pendulum = Pendulum(1.25, (0, 0), (0, 0), precision=36)

plt.xlabel('x')
plt.ylabel('y')


def animate(i):
    wheel.update((wheel_positions[0][i], wheel_positions[1][i]), wheel_phases[i])
    cord.update(platform_position, (wheel_positions[0][i], wheel_positions[1][i]))
    pendulum.update((pendulum_positions[0][i], pendulum_positions[1][i]),
                    (wheel_positions[0][i], wheel_positions[1][i]))
    return *wheel.return_plot(), *cord.return_plot(), *pendulum.return_plot()


ani = FuncAnimation(window, animate, frames=STEPS, interval=round(1000 / FPS), repeat=False, blit=True)
plt.show()
print("Симуляция успешно завершена.")
```