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

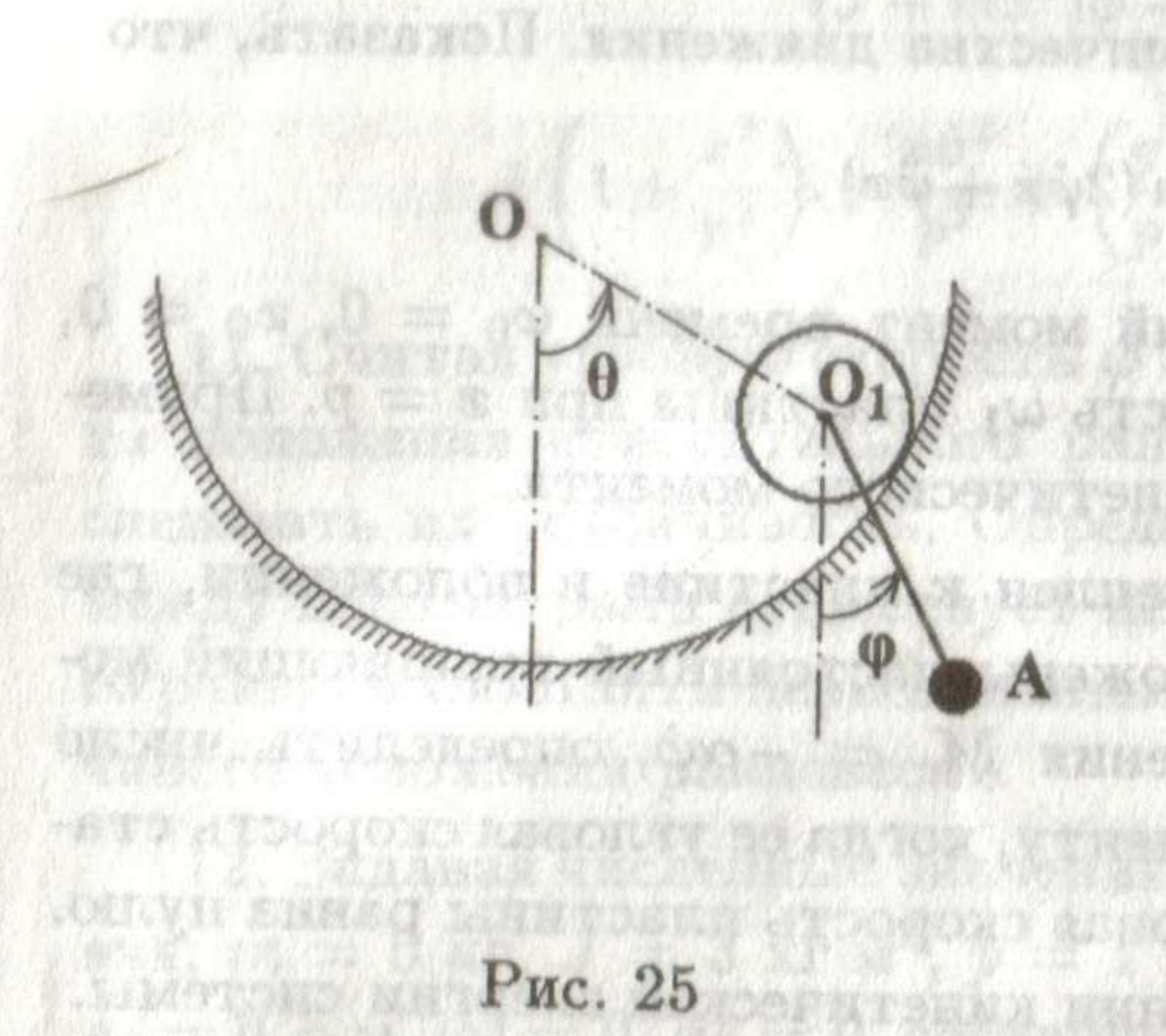
с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024**Вариант №25**

**Задание:**

Реализовать анимацию движения механической системы.

**Механическая система:**



**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

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.")