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:** Построить заданную траекторию и анимацию движения точки, а также отобразить стрелки радиус-вектора, скорости и ускорения. Построить радиус кривизны траектории.

**Закон движения точки:**

r(t) = 1 + sin(8t)

φ(t) = t + 0.5 \* sin(8t)

**Текст программы:**

import numpy as np

import sympy as sym

import math

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

# Определение символической переменной времени

time = sym.Symbol('t')

# Исходные урвнения из условия

radius = 1 + sym.sin(8 \* time)

angle\_phi = time + 0.5 \* sym.sin(8 \* time)

# Выражения координат

coord\_x = radius \* sym.cos(angle\_phi)

coord\_y = radius \* sym.sin(angle\_phi)

# Вычисление производных для скорости

velocity\_x = sym.diff(coord\_x, time)

velocity\_y = sym.diff(coord\_y, time)

speed = sym.sqrt(velocity\_x\*\*2 + velocity\_y\*\*2)

# Вычисление производных для ускорения

accel\_x = sym.diff(velocity\_x, time)

accel\_y = sym.diff(velocity\_y, time)

accel\_magnitude = sym.sqrt(accel\_x\*\*2 + accel\_y\*\*2)

# Тангенциальное и нормальное ускорение

tangential\_accel = sym.diff(speed, time)

normal\_accel = sym.sqrt(accel\_magnitude\*\*2 - tangential\_accel\*\*2)

# Радиус кривизны

radius\_of\_curvature = speed\*\*2 / normal\_accel

# Компоненты тангенциального ускорения

tangential\_accel\_x = (velocity\_x / speed) \* tangential\_accel

tangential\_accel\_y = (velocity\_y / speed) \* tangential\_accel

# Нормальный единичный вектор

normal\_accel\_x = accel\_x - tangential\_accel\_x

normal\_accel\_y = accel\_y - tangential\_accel\_y

normal\_magnitude = sym.sqrt(normal\_accel\_x\*\*2 + normal\_accel\_y\*\*2)

unit\_normal\_x = normal\_accel\_x / normal\_magnitude

unit\_normal\_y = normal\_accel\_y / normal\_magnitude

# Радиус кривизны по осям

curvature\_radius\_x = unit\_normal\_x \* radius\_of\_curvature

curvature\_radius\_y = unit\_normal\_y \* radius\_of\_curvature

# Временной массив

time\_values = np.linspace(0, 2, 300) # 2 секунды

# Инициализация массивов для хранения данных

pos\_x = np.zeros\_like(time\_values)

pos\_y = np.zeros\_like(time\_values)

vel\_x = np.zeros\_like(time\_values)

vel\_y = np.zeros\_like(time\_values)

accel\_x\_vals = np.zeros\_like(time\_values)

accel\_y\_vals = np.zeros\_like(time\_values)

radius\_vec\_x = np.zeros\_like(time\_values)

radius\_vec\_y = np.zeros\_like(time\_values)

curv\_radius\_x = np.zeros\_like(time\_values)

curv\_radius\_y = np.zeros\_like(time\_values)

# Вычисление значений для каждого момента времени

for idx, t\_val in enumerate(time\_values):

# Позиция

pos\_x[idx] = coord\_x.subs(time, t\_val).evalf()

pos\_y[idx] = coord\_y.subs(time, t\_val).evalf()

# Скорость

vel\_x[idx] = velocity\_x.subs(time, t\_val).evalf()

vel\_y[idx] = velocity\_y.subs(time, t\_val).evalf()

# Ускорение

accel\_x\_vals[idx] = accel\_x.subs(time, t\_val).evalf()

accel\_y\_vals[idx] = accel\_y.subs(time, t\_val).evalf()

# Радиус-вектор

radius\_vec\_x[idx] = coord\_x.subs(time, t\_val).evalf()

radius\_vec\_y[idx] = coord\_y.subs(time, t\_val).evalf()

# Радиус кривизны

curv\_radius\_x[idx] = curvature\_radius\_x.subs(time, t\_val).evalf()

curv\_radius\_y[idx] = curvature\_radius\_y.subs(time, t\_val).evalf()

# Настройка графика

fig, ax = plt.subplots()

ax.set\_aspect('equal')

ax.set\_xlim(np.min(pos\_x) - 1, np.max(pos\_x) + 1)

ax.set\_ylim(np.min(pos\_y) - 1, np.max(pos\_y) + 1)

# Траектория движения

ax.plot(pos\_x, pos\_y, label='Траектория')

# Оси координат

ax.axhline(0, color='black')

ax.axvline(0, color='black')

# Инициализация элементов анимации

point, = ax.plot(pos\_x[0], pos\_y[0], 'o', label='Точка')

velocity\_line, = ax.plot(

[pos\_x[0], pos\_x[0] + vel\_x[0]],

[pos\_y[0], pos\_y[0] + vel\_y[0]],

'r-', label='Скорость'

)

accel\_line, = ax.plot(

[pos\_x[0], pos\_x[0] + accel\_x\_vals[0]],

[pos\_y[0], pos\_y[0] + accel\_y\_vals[0]],

'g-', label='Ускорение'

)

radius\_vector, = ax.plot(

[0, pos\_x[0]],

[0, pos\_y[0]],

'c-', label='Радиус-вектор'

)

curvature\_vector, = ax.plot(

[pos\_x[0], pos\_x[0] + curv\_radius\_x[0]],

[pos\_y[0], pos\_y[0] + curv\_radius\_y[0]],

'b-', label='Радиус кривизны'

)

# Настройка стрелок

arrow\_scale = 0.3

arrow\_shape\_x = np.array([-0.2 \* arrow\_scale, 0, -0.2 \* arrow\_scale])

arrow\_shape\_y = np.array([0.1 \* arrow\_scale, 0, -0.1 \* arrow\_scale])

def add\_rotated\_arrow(x, y, dx, dy, color):

angle = math.atan2(dy, dx)

rotated\_x = arrow\_shape\_x \* np.cos(angle) - arrow\_shape\_y \* np.sin(angle)

rotated\_y = arrow\_shape\_x \* np.sin(angle) + arrow\_shape\_y \* np.cos(angle)

arrow, = ax.plot(

rotated\_x + x + dx,

rotated\_y + y + dy,

color + '-'

)

return arrow

# Добавление стрелок

velocity\_arrow = add\_rotated\_arrow(pos\_x[0], pos\_y[0], vel\_x[0], vel\_y[0], 'r')

accel\_arrow = add\_rotated\_arrow(pos\_x[0], pos\_y[0], accel\_x\_vals[0], accel\_y\_vals[0], 'g')

radius\_arrow = add\_rotated\_arrow(pos\_x[0], pos\_y[0], 0, 0, 'c')

curvature\_arrow = add\_rotated\_arrow(

pos\_x[0], pos\_y[0], curv\_radius\_x[0], curv\_radius\_y[0], 'b'

)

def update\_animation(frame):

# Обновление позиции точки

point.set\_data(pos\_x[frame], pos\_y[frame])

# Обновление линии скорости

velocity\_line.set\_data(

[pos\_x[frame], pos\_x[frame] + vel\_x[frame]],

[pos\_y[frame], pos\_y[frame] + vel\_y[frame]]

)

# Обновление линии ускорения

accel\_line.set\_data(

[pos\_x[frame], pos\_x[frame] + accel\_x\_vals[frame]],

[pos\_y[frame], pos\_y[frame] + accel\_y\_vals[frame]]

)

# Обновление радиус-вектора

radius\_vector.set\_data(

[0, pos\_x[frame]],

[0, pos\_y[frame]]

)

# Обновление радиуса кривизны

curvature\_vector.set\_data(

[pos\_x[frame], pos\_x[frame] + curv\_radius\_x[frame]],

[pos\_y[frame], pos\_y[frame] + curv\_radius\_y[frame]]

)

# Обновление стрелки скорости

velocity\_arrow.set\_data(

arrow\_shape\_x \* np.cos(math.atan2(vel\_y[frame], vel\_x[frame])) - arrow\_shape\_y \* np.sin(math.atan2(vel\_y[frame], vel\_x[frame])) + pos\_x[frame] + vel\_x[frame],

arrow\_shape\_x \* np.sin(math.atan2(vel\_y[frame], vel\_x[frame])) + arrow\_shape\_y \* np.cos(math.atan2(vel\_y[frame], vel\_x[frame])) + pos\_y[frame] + vel\_y[frame]

)

# Обновление стрелки ускорения

accel\_arrow.set\_data(

arrow\_shape\_x \* np.cos(math.atan2(accel\_y\_vals[frame], accel\_x\_vals[frame])) - arrow\_shape\_y \* np.sin(math.atan2(accel\_y\_vals[frame], accel\_x\_vals[frame])) + pos\_x[frame] + accel\_x\_vals[frame],

arrow\_shape\_x \* np.sin(math.atan2(accel\_y\_vals[frame], accel\_x\_vals[frame])) + arrow\_shape\_y \* np.cos(math.atan2(accel\_y\_vals[frame], accel\_x\_vals[frame])) + pos\_y[frame] + accel\_y\_vals[frame]

)

# Обновление стрелки радиус-вектора

radius\_arrow.set\_data(

arrow\_shape\_x \* np.cos(math.atan2(pos\_y[frame], pos\_x[frame])) - arrow\_shape\_y \* np.sin(math.atan2(pos\_y[frame], pos\_x[frame])) + pos\_x[frame],

arrow\_shape\_x \* np.sin(math.atan2(pos\_y[frame], pos\_x[frame])) + arrow\_shape\_y \* np.cos(math.atan2(pos\_y[frame], pos\_x[frame])) + pos\_y[frame]

)

# Обновление стрелки радиуса кривизны

curvature\_arrow.set\_data(

arrow\_shape\_x \* np.cos(math.atan2(curv\_radius\_y[frame], curv\_radius\_x[frame])) - arrow\_shape\_y \* np.sin(math.atan2(curv\_radius\_y[frame], curv\_radius\_x[frame])) + pos\_x[frame] + curv\_radius\_x[frame],

arrow\_shape\_x \* np.sin(math.atan2(curv\_radius\_y[frame], curv\_radius\_x[frame])) + arrow\_shape\_y \* np.cos(math.atan2(curv\_radius\_y[frame], curv\_radius\_x[frame])) + pos\_y[frame] + curv\_radius\_y[frame]

)

# Создание анимации

animation = FuncAnimation(

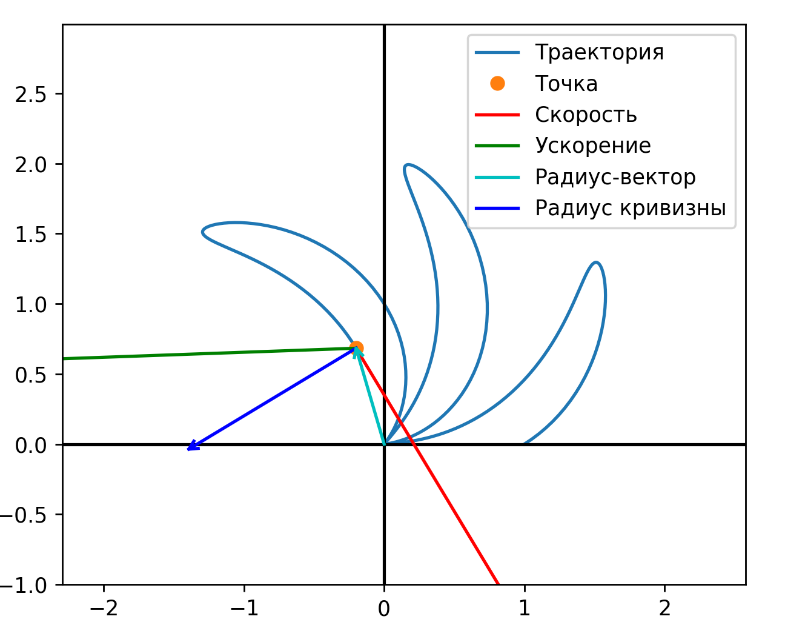
fig, update\_animation, frames=300, interval=50, repeat=False

)

# Добавление подписей

ax.legend()

plt.show()

**Результат работы программы:**

**Вывод:**

В рамках этой лабораторной работы я успешно реализовал решение задачи с использованием языка программирования Python и библиотек matplotlib, numpy и sympy. Я смоделировал заданную траекторию движения точки и создал анимацию, иллюстрирующую это движение. Кроме того, визуализировал векторы скорости, ускорения и радиус-вектора для каждого момента времени, что позволило наглядно понять динамику процесса.

Эта работа помогла понять тему движения точки. В ходе её выполнения я не только справился с поставленными задачами, но и получил ценный опыт применения программирования для анализа и визуализации задач.