ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ОТЧЕТ**

**О ВЫПОЛОНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**«АНИМАЦИЯ ТОЧКИ»**

**ПО ДИСЦИПЛИНЕ «Основы Компьютерного Моделирования Математических Систем»**

**ВАРИАНТ ЗАДАНИЯ № 2**

Выполнил(а) студент группы М8О-209Б-23

Борисов Д.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

Ст. преп. каф. 802 Волков Е.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

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

Москва, 2024

**Вариант № 2**

**Задание:**

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

**Закон движения точки:**

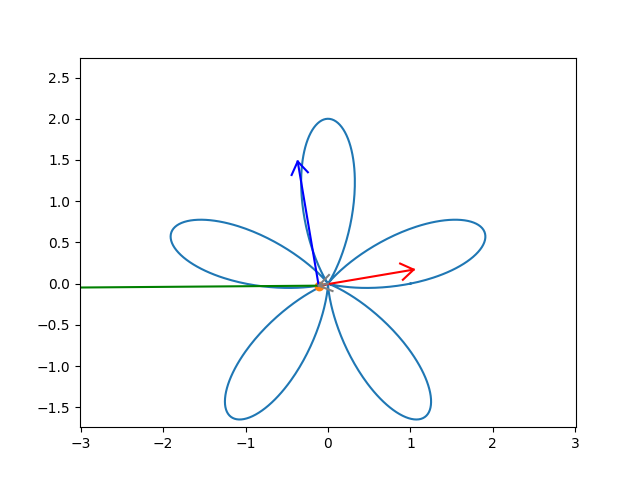
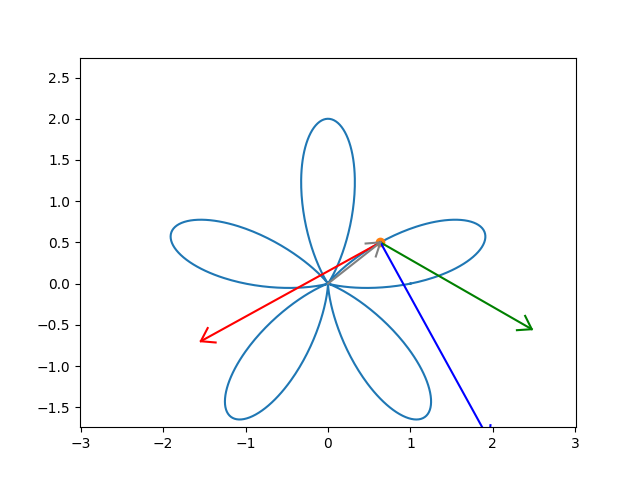
𝑟 = 1 + 𝑠𝑖𝑛(5𝑡)

𝜑 = 𝑡

**Текст программы**

*# Импорт необходимых библиотек*import math  
import sympy as sp  
import matplotlib.pyplot as plot  
import numpy as np  
from matplotlib.animation import FuncAnimation  
  
*# Указание параметров моделирования*STEPS = 1000  
TIP\_LENGTH = 0.15  
TIP\_WIDTH = 0.1  
START\_VALUE = 0  
END\_VALUE = 2 \* math.pi  
  
  
*# Определение параметрических функций для движения точки*def r(t: float) -> float:  
 *# Функция радиус-вектора материальной точки от времени* return 1 + sp.sin(5 \* t)  
  
  
def phi(t: float) -> float:  
 *# Функция угла материальной точки от времени* return t  
  
  
*# Функция поворота двумерной ДСК*def rot2D(X: np.ndarray, Y: np.ndarray, phi: float):  
 *# Поворот двумерной ДСК с помощью матрицы поворота* X\_r = X \* np.cos(phi) - Y \* np.sin(phi)  
 Y\_r = X \* np.sin(phi) + Y \* np.cos(phi)  
 return X\_r, Y\_r  
  
  
*# Создание символьной переменной времени*t = sp.Symbol('t')  
  
*# Переход от полярных координат к декартовым*x = r(t) \* sp.cos(phi(t)) *# x-координата*y = r(t) \* sp.sin(phi(t)) *# y-координата  
  
# Вычисление кинематических характеристик  
# Скорость - первая производная по времени*Vx = sp.diff(x, t) *# Проекция скорости на ось x*Vy = sp.diff(y, t) *# Проекция скорости на ось y  
  
# Ускорение - вторая производная по времени*ax = sp.diff(Vx, t) *# Проекция ускорения на ось x*ay = sp.diff(Vy, t) *# Проекция ускорения на ось y  
  
# Вычисление радиуса кривизны*V = sp.sqrt(Vx \* Vx + Vy \* Vy) *# Модуль скорости*R = (V \*\* 3) / abs(Vx \* ay - Vy \* ax) *# Радиус кривизны траектории  
# Пояснение к R: (V\_x^2 + V\_y^3)^{3/2} / (V\_x \* a\_y - V\_y \* a\_x)  
  
# Вычисление нормального вектора*nx = -Vy / V  
ny = Vx / V  
  
*# Вектор к центру кривизны (нормальный вектор на радиус)*rx = R \* nx  
ry = R \* ny  
  
*# Создание массивов для числовых расчетов*T = np.linspace(START\_VALUE, END\_VALUE, STEPS) *# Массив значений времени  
# Инициализация массивов для хранения значений*X = np.zeros\_like(T) *# x-координаты*Y = np.zeros\_like(T) *# y-координаты*VX = np.zeros\_like(T) *# x-компоненты скорости*VY = np.zeros\_like(T) *# y-компоненты скорости*AX = np.zeros\_like(T) *# x-компоненты ускорения*AY = np.zeros\_like(T) *# y-компоненты ускорения*RX = np.zeros\_like(T) *# x-компоненты радиуса кривизны*RY = np.zeros\_like(T) *# y-компоненты радиуса кривизны  
  
# Вычисление значений для каждого момента времени*for i in range(len(T)):  
 X[i] = x.subs(t, T[i])  
 Y[i] = y.subs(t, T[i])  
 VX[i] = 0.5 \* Vx.subs(t, T[i]) *# Масштабирование для визуализации* VY[i] = 0.5 \* Vy.subs(t, T[i])  
 AX[i] = 0.2 \* ax.subs(t, T[i])  
 AY[i] = 0.2 \* ay.subs(t, T[i])  
 RX[i] = rx.subs(t, T[i])  
 RY[i] = ry.subs(t, T[i])  
  
*# Настройка графика*fgr = plot.figure()  
grf = fgr.add\_subplot(1, 1, 1)  
grf.axis('equal') *# Одинаковый масштаб по осям*grf.set(xlim=[-3, 3], ylim=[-2, 3])  
grf.plot(X, Y) *# Построение траектории  
  
# Создание начальных элементов анимации*Pnt = grf.plot(X[0], Y[0], marker='o')[0] *# Точка*Vpl = grf.plot([X[0], X[0] + VX[0]], [Y[0], Y[0] + VY[0]], 'r')[0] *# Вектор скорости*Apl = grf.plot([X[0], X[0] + AX[0]], [Y[0], Y[0] + AY[0]], 'g')[0] *# Вектор ускорения*Rpl = grf.plot([X[0], X[0] + RX[0]], [Y[0], Y[0] + RY[0]], 'b')[0] *# Вектор радиуса кривизны*Radpl = grf.plot([0, X[0]], [0, Y[0]], color='gray')[0] *# Радиус-вектор  
  
  
# Функция для создания стрелок векторов*def vect\_arrow(vec\_x, vec\_y, \_x, \_y):  
 *# Создание наконечника стрелки* arr\_x = np.array([-TIP\_LENGTH, 0, -TIP\_LENGTH])  
 arr\_y = np.array([TIP\_WIDTH, 0, -TIP\_WIDTH])  
  
 *# Поворот наконечника* phi = math.atan2(vec\_y, vec\_x)  
 rot\_x, rot\_y = rot2D(arr\_x, arr\_y, phi)  
  
 *# Перемещение наконечника в нужную позицию* arr\_x = rot\_x + \_x + vec\_x  
 arr\_y = rot\_y + \_y + vec\_y  
  
 return arr\_x, arr\_y  
  
  
*# Создание начальных стрелок для векторов*ArVX, ArVY = vect\_arrow(VX[0], VY[0], X[0], Y[0])  
V\_arr = grf.plot(ArVX, ArVY, 'r')[0]  
  
ArAX, ArAY = vect\_arrow(AX[0], AY[0], X[0], Y[0])  
A\_arr = grf.plot(ArAX, ArAY, 'g')[0]  
  
ArRX, ArRY = vect\_arrow(RX[0], RY[0], X[0], Y[0])  
R\_arr = grf.plot(ArRX, ArRY, 'b')[0]  
  
ArRadX, ArRadY = vect\_arrow(X[0], Y[0], 0, 0)  
Rad\_arr = grf.plot(ArRadX, ArRadY, color='gray')[0]  
  
  
*# Функция анимации*def animate(j):  
 global ArVX, ArVY, ArAX, ArAY, ArRX, ArRY, ArRadX, ArRadY  
 *# Обновление положения точки* Pnt.set\_data([X[j]], [Y[j]])  
  
 *# Обновление вектора скорости* Vpl.set\_data([X[j], X[j] + VX[j]], [Y[j], Y[j] + VY[j]])  
 ArVX, ArVY = vect\_arrow(VX[j], VY[j], X[j], Y[j])  
 V\_arr.set\_data(ArVX, ArVY)  
  
 *# Обновление вектора ускорения* Apl.set\_data([X[j], X[j] + AX[j]], [Y[j], Y[j] + AY[j]])  
 ArAX, ArAY = vect\_arrow(AX[j], AY[j], X[j], Y[j])  
 A\_arr.set\_data(ArAX, ArAY)  
  
 *# Обновление вектора радиуса кривизны* Rpl.set\_data([X[j], X[j] + RX[j]], [Y[j], Y[j] + RY[j]])  
 ArRX, ArRY = vect\_arrow(RX[j], RY[j], X[j], Y[j])  
 R\_arr.set\_data(ArRX, ArRY)  
  
 *# Обновление радиус-вектора* Radpl.set\_data([0, X[j]], [0, Y[j]])  
 ArRadX, ArRadY = vect\_arrow(X[j], Y[j], 0, 0)  
 Rad\_arr.set\_data(ArRadX, ArRadY)  
  
 return [Pnt, Vpl, V\_arr, Apl, A\_arr, Rpl, R\_arr, Radpl, Rad\_arr]  
  
  
*# Создание и запуск анимации*an = FuncAnimation(fgr, animate, frames=STEPS, interval=1)  
plot.show() *# Отображение графика*

**Результат работы программы:**



**Вывод:**

В процессе первой лабораторной работы я научился задавать движение точки по заданной траектории в полярных координатах и строить её анимацию. Мне удалось получить координаты x(t) и y(t) в каждый момент времени, а также вычислить скорость и ускорение точки. Дополнительно я добавил визуализацию радиус-вектора, что позволило наглядно отображать положение точки относительно начала координат. После этого я создал анимацию с векторами скорости, ускорения и радиус-вектора. Это помогло мне лучше понять кинематику точки и увидеть преимущества использования символьных и численных методов при анализе движения.