ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«АНИМАЦИЯ ТОЧКИ»

ПО ДИСЦИПЛИНЕ «ТЕОРЕТИЧЕСКАЯ МЕХАНИКА И ОСНОВЫ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ»

ВАРИАНТ ЗАДАНИЯ №9

Выполнил студент группы М8О-209Б-22

Концебалов О.С.______________________________

подпись, дата

Проверил и принял

Авдюшкин А.Н.______________________________

подпись, дата

Москва, 2023

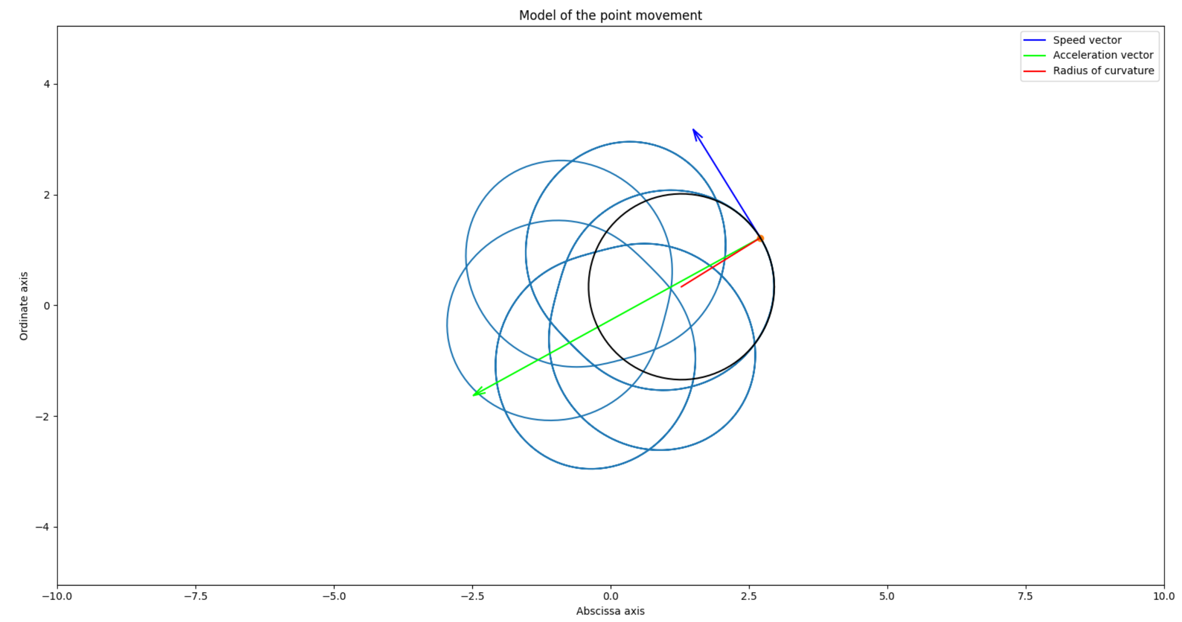
Задание: построить заданную траекторию, запустить анимацию движения точки, построить стрелки радиус-вектора, вектора скорости, вектора ускорения и радиуса кривизны.

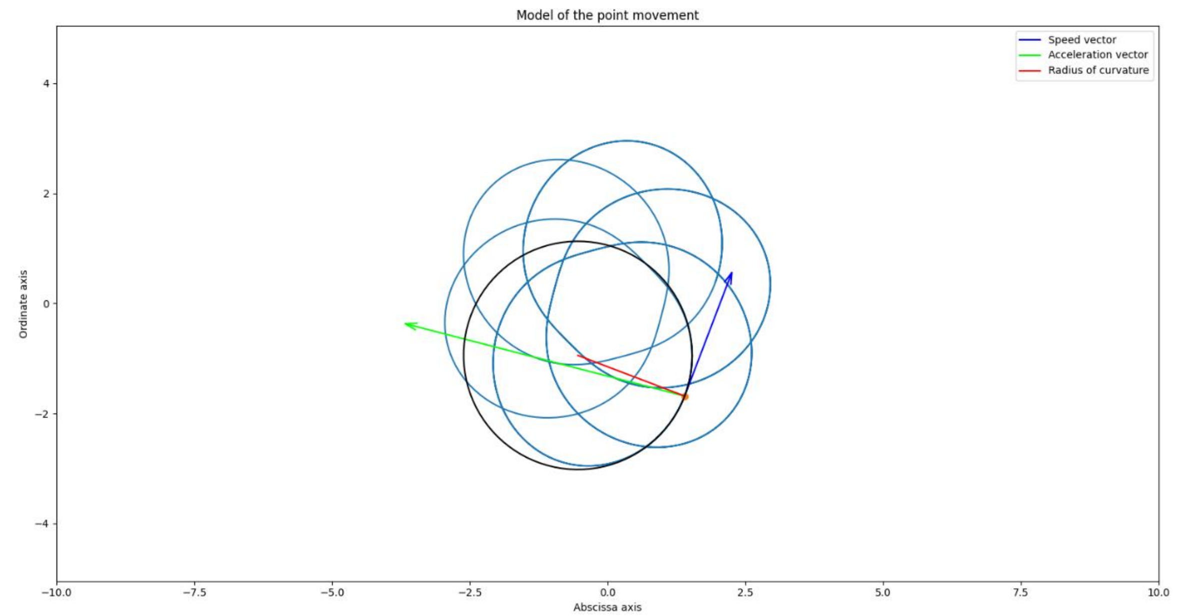
Условия задачи 9 варианта:

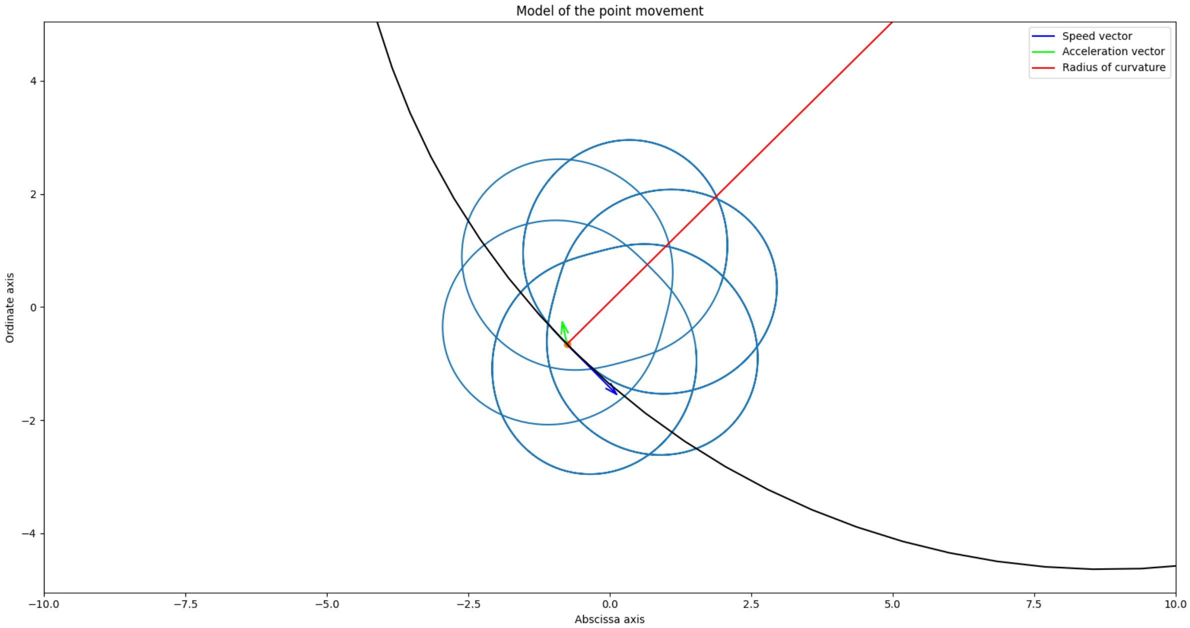
r(t) = 2 + sin(6t)

φ(t) = 5t + 0.2cos(6t)

Рисунок получившейся физической модели:







Красным цветом изображен радиус-вектор
Синим цветом изображен вектор скорости
Зеленым цветом изображен вектор ускорения

Код программы

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation


t = sp.Symbol('t')

k_v = 0.2
k_w = 0.075

a = 0.2
b = 0.06

polar_r = 2 + sp.sin(6 * t)
polar_phi = 5 * t + 0.2 * sp.cos(6 * t)

x = polar_r * sp.cos(polar_phi)
y = polar_r * sp.sin(polar_phi)


def rotate_2D(X, Y, angle) -> tuple:
```

```
    rotated_X = X * np.cos(angle) - Y * np.sin(angle)
    rotated_Y = X * np.sin(angle) + Y * np.cos(angle)

    return rotated_X, rotated_Y


def calculate_speed_projections(x_cord, y_cord) -> tuple:
    Vx = sp.diff(x_cord, t)
    Vy = sp.diff(y_cord, t)

    return Vx, Vy


def calculate_acceleration_projections(x_cord, y_cord) -> tuple:
    Vx = sp.diff(x_cord, t)
    Vy = sp.diff(y_cord, t)

    Wx = sp.diff(Vx, t)
    Wy = sp.diff(Vy, t)

    return Wx, Wy


def decompose_acceleration_into_tangential_and_normal(V, W) -> tuple:
    W_tan = sp.diff(V, t)
    W_norm = sp.sqrt(W * W - W_tan * W_tan)

    return W_tan, W_norm


v_x, v_y = calculate_speed_projections(x, y)
v = sp.sqrt(v_x * v_x + v_y * v_y)

w_x, w_y = calculate_acceleration_projections(x, y)
w = sp.sqrt(w_x * w_x + w_y * w_y)

w_tan, w_norm = decompose_acceleration_into_tangential_and_normal(v, w)
radius_of_curvature = v * v / w_norm

norm_x = (w_x - ((v_x / v) * w_tan)) / w_norm
norm_y = (w_y - ((v_y / v) * w_tan)) / w_norm

F_x = sp.lambdify(t, x)
F_y = sp.lambdify(t, y)
F_Vx = sp.lambdify(t, v_x)
```

```
F_Vy = sp.lambdify(t, v_y)
F_Wx = sp.lambdify(t, w_x)
F_Wy = sp.lambdify(t, w_y)
F_R = sp.lambdify(t, radius_of_curvature)
F_NORMx = sp.lambdify(t, norm_x)
F_NORMy = sp.lambdify(t, norm_y)

t = np.linspace(0, 10, 1000)

x = F_x(t)
y = F_y(t)
v_x = F_Vx(t)
v_y = F_Vy(t)
w_x = F_Wx(t)
w_y = F_Wy(t)
radius_of_curvature = F_R(t)
norm_x = F_NORMx(t)
norm_y = F_NORMy(t)

fig = plt.figure(figsize=(9, 8))

graph = fig.add_subplot(1, 1, 1)
graph.axis('equal')
graph.set_title("Model of the point movement")
graph.set_xlabel("Abscissa axis")
graph.set_ylabel("Ordinate axis")
graph.set(xlim=[-10, 10], ylim=[-10, 10])

graph.plot(x, y)
point = graph.plot(x[0], y[0], marker='o')[0]

v_vec = graph.plot([x[0], x[0] + k_v * v_x[0]],
                   [y[0], y[0] + k_v * v_y[0]],
                   color=[0, 0, 1],
                   label="Speed vector")[0]

w_vec = graph.plot([x[0], x[0] + k_w * w_x[0]],
                   [y[0], y[0] + k_w * w_y[0]],
                   color=[0, 1, 0],
                   label="Acceleration vector")[0]

radius_of_curvature_line = graph.plot([x[0], x[0] + radius_of_curvature[0] * norm_x[0]],
                                      [y[0], y[0] + radius_of_curvature[0] * norm_y[0]],
                                      color=[1, 0, 0],
                                      label="Radius of curvature")[0]
```

```
graph.legend()

v_alpha = np.arctan2(v_y, v_x)
w_alpha = np.arctan2(w_y, w_x)

x_arr = np.array([-a, 0, -a])
y_arr = np.array([b, 0, -b])

v_rot_x, v_rot_y = rotate_2D(x_arr, y_arr, v_alpha[0])
w_rot_x, w_rot_y = rotate_2D(x_arr, y_arr, w_alpha[0])

v_arrow = graph.plot(x[0] + k_v * v_x[0] + v_rot_x,
                     y[0] + k_v * v_y[0] + v_rot_y,
                     color=[0, 0, 1])[0]

w_arrow = graph.plot(x[0] + k_w * w_x[0] + w_rot_x,
                     y[0] + k_w * w_y[0] + w_rot_y,
                     color=[0, 1, 0])[0]

phi = np.linspace(0, 6.28, 100)
circle = graph.plot(x[0] + radius_of_curvature[0] * norm_x[0] * np.cos(phi),
                    y[0] + radius_of_curvature[0] * norm_y[0] * np.sin(phi),
                    color=[0, 0, 0])[0]



def animation(i) -> list:
    point.set_data(x[i], y[i])
    v_vec.set_data([x[i], x[i] + k_v * v_x[i]], [y[i], y[i] + k_v * v_y[i]])
    w_vec.set_data([x[i], x[i] + k_w * w_x[i]], [y[i], y[i] + k_w * w_y[i]])
    radius_of_curvature_line.set_data([x[i], x[i] + radius_of_curvature[i] * norm_x[i]],
                                      [y[i], y[i] + radius_of_curvature[i] * norm_y[i]])
    circle.set_data(x[i] + radius_of_curvature[i] * norm_x[i] + radius_of_curvature[i] *
np.cos(phi),
                    y[i] + radius_of_curvature[i] * norm_y[i] + radius_of_curvature[i] * np.sin(phi))

    v_rot_x, v_rot_y = rotate_2D(x_arr, y_arr, v_alpha[i])
    w_rot_x, w_rot_y = rotate_2D(x_arr, y_arr, w_alpha[i])

    v_arrow.set_data(x[i] + k_v * v_x[i] + v_rot_x, y[i] + k_v * v_y[i] + v_rot_y)
    w_arrow.set_data(x[i] + k_w * w_x[i] + w_rot_x, y[i] + k_w * w_y[i] + w_rot_y)

    return [point, v_vec, w_vec, radius_of_curvature_line, v_arrow, w_arrow]


show_movement = FuncAnimation(fig, animation, frames=len(t), interval=20)
```

```
animation_running = True


def animation_pause(event) -> None:
    global animation_running

    if animation_running:
        show_movement.event_source.stop()
        animation_running = False
    else:
        show_movement.event_source.start()
        animation_running = True


if __name__ == "__main__":
    fig.canvas.mpl_connect('button_press_event', animation_pause)
    plt.show()
```

Пояснения

В процессе выполнения работы мне нужно было изобразить несколько векторов. Вектор скорости достаточно просто рисовался, благодаря имеющейся функции координат, требовалось всего лишь взять производную по каждой из координат. Таким же образом рисовался вектор ускорения – вторая производная по времени от координат. Радиус-вектор – отрезок, соединяющий начало координат и движущуюся точку. А вот построить радиус кривизны уже не было такой простой задачей.

Чтобы построить радиус кривизны, нам требовалось узнать направление и модуль. Модуль можно узнать по формуле:

$$\rho = \frac{V^2}{w_n},$$

где w_n – нормальное ускорение. w_n можно узнать из полного и тангенциального. Тангенциальное ускорение получается, как производная модуля скорости по времени. Таким образом, получаем ρ. Следующим шагом требуется найти направление. Радиус кривизны сонаправлен с нормальным ускорением, следовательно, требуется лишь найти вектор нормального ускорения и нормировать его. Получить координаты нормального ускорения можно через полное и тангенциальное, и поделив координаты на длины получим единичный вектор, сонаправленный с радиусом кривизны. Умножая длину радиуса кривизны на единичный вектор получаем вектор радиуса кривизны.

Вывод

Я успешно выполнил лабораторную работу по теоретической механике. С помощью языка программирования Python и библиотек matplotlib, numpy и sympy я построил заданную траекторию, а также запустил анимацию движения точки по этой траектории. Для каждого момента времени я изобразил векторы скорости, ускорения, радиус-вектора, вектора радиуса кривизны.

Эта лабораторная работа позволила мне лучше разобраться в теме движения точки, понять как связаны между собой разные характеристики движения точки – скорость и ускорения.