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 №15**

Выполнил студент группы М8О-209Б-22

Недосекин А.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

Авдюшкин А.Н.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Москва, 2023

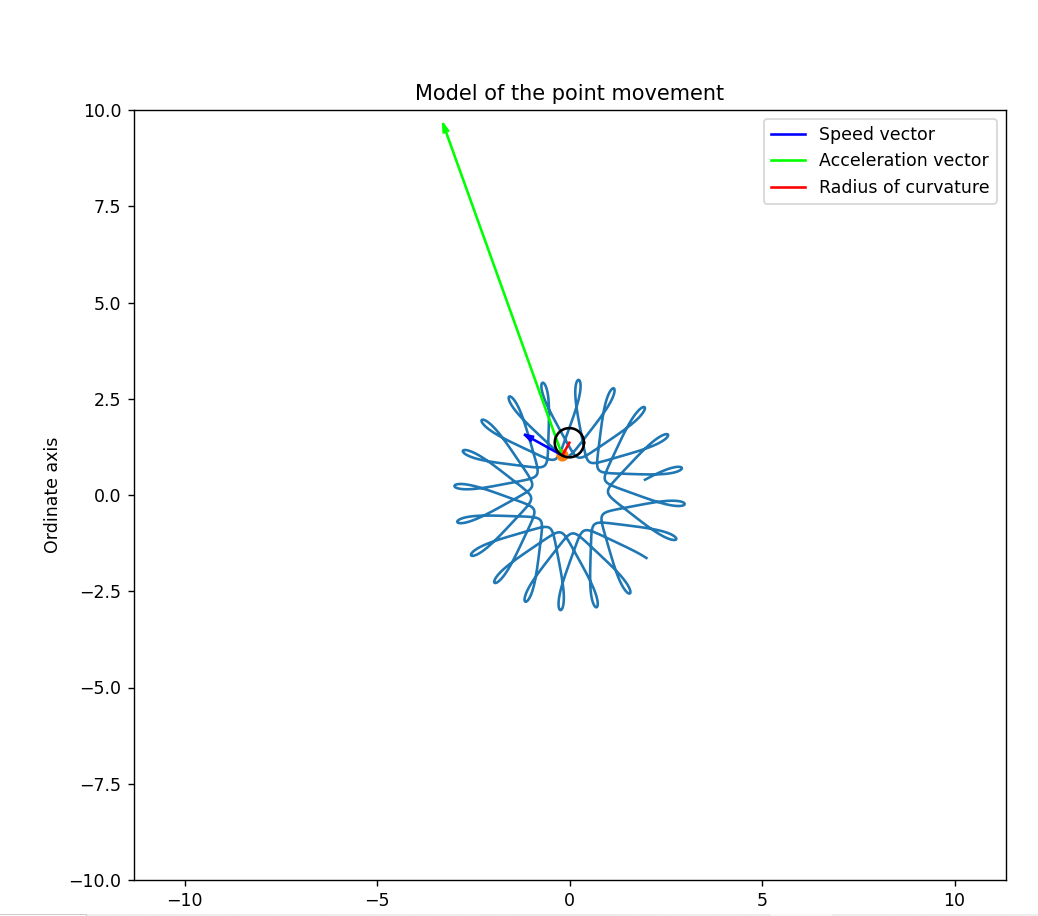
**Задание:** построить заданную траекторию, запустить анимацию движения точки, построить стрелки радиус-вектора, вектора скорости, вектора ускорения и радиуса кривизны.

**Условия задачи 15 варианта:**

r(t) = 2 + sin(12t)

φ(t) = 1.8t + 0.2cos(12t)

**Рисунок получившейся физической модели:**



Красным цветом изображен радиус-вектор  
Синим цветомизображен 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')

kv = 0.2

kw = 0.075

a = 0.2

b = 0.06

polar\_r = 2 + sp.sin(12 \* t)

polar\_phi = 1.8 \* t + 0.2 \* sp.cos(12 \* t)

x = polar\_r \* sp.cos(polar\_phi)

y = polar\_r \* sp.sin(polar\_phi)

def rotate2D(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 calculateSpeed(x\_cord, y\_cord) -> tuple:

Vx = sp.diff(x\_cord, t)

Vy = sp.diff(y\_cord, t)

return Vx, Vy

def calculateAcceleration(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 intoTangentialAndNormal(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 = calculateSpeed(x, y)

v = sp.sqrt(v\_x \* v\_x + v\_y \* v\_y)

w\_x, w\_y = calculateAcceleration(x, y)

w = sp.sqrt(w\_x \* w\_x + w\_y \* w\_y)

w\_tan, w\_norm = intoTangentialAndNormal(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] + kv \* v\_x[0]],

[y[0], y[0] + kv \* v\_y[0]],

color=[0, 0, 1],

label="Speed vector")[0]

w\_vec = graph.plot([x[0], x[0] + kw \* w\_x[0]],

[y[0], y[0] + kw \* 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 = rotate2D(x\_arr, y\_arr, v\_alpha[0])

w\_rot\_x, w\_rot\_y = rotate2D(x\_arr, y\_arr, w\_alpha[0])

v\_arrow = graph.plot(x[0] + kv \* v\_x[0] + v\_rot\_x,

y[0] + kv \* v\_y[0] + v\_rot\_y,

color=[0, 0, 1])[0]

w\_arrow = graph.plot(x[0] + kw \* w\_x[0] + w\_rot\_x,

y[0] + kw \* 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] + kv \* v\_x[i]], [y[i], y[i] + kv \* v\_y[i]])

w\_vec.set\_data([x[i], x[i] + kw \* w\_x[i]], [y[i], y[i] + kw \* 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 = rotate2D(x\_arr, y\_arr, v\_alpha[i])

w\_rot\_x, w\_rot\_y = rotate2D(x\_arr, y\_arr, w\_alpha[i])

v\_arrow.set\_data(x[i] + kv \* v\_x[i] + v\_rot\_x, y[i] + kv \* v\_y[i] + v\_rot\_y)

w\_arrow.set\_data(x[i] + kw \* w\_x[i] + w\_rot\_x, y[i] + kw \* 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**

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

где – нормальное ускорение. можно узнать из полного и тангенциального. Тангенциальное ускорение получается, как производная модуля скорости по времени. Таким образом, получаем 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.