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 № 22**

Выполнил(а) студент группы М8О-201Б-23

Тутаев В.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

Зав. каф. 802, Волков Е.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

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

Москва, 2024

*Задание:* построить анимацию движения системы с помощью Python.

*Вариант 22:* r(t) = 5 – 0.5t

φ(t) = 2t

*Код:*

*Файл lab1.py:*

import numpy as np  
import sympy as sp  
import matplotlib.pyplot as plt  
import matplotlib.animation as animation  
import math  
  
# Время  
T = np.linspace(0, 10, 1000)  
  
# Символьная переменная  
t = sp.Symbol('t')  
  
# Функции r(t) и phi(t)  
r = 5 - 0.5 \* t  
phi = 2 \* t  
  
# Координаты в декартовой системе  
x = r \* sp.cos(phi)  
y = r \* sp.sin(phi)  
  
# Производные  
x\_diff = sp.diff(x, t)  
y\_diff = sp.diff(y, t)  
  
x\_diff2 = sp.diff(x\_diff, t)  
y\_diff2 = sp.diff(y\_diff, t)  
  
# Массивы для хранения значений  
X = np.zeros\_like(T)  
Y = np.zeros\_like(T)  
VX = np.zeros\_like(T)  
VY = np.zeros\_like(T)  
AX = np.zeros\_like(T)  
AY = np.zeros\_like(T)  
  
# Вычисление значений  
for i in range(len(T)):  
 X[i] = float(x.subs(t, T[i]))  
 Y[i] = float(y.subs(t, T[i]))  
 VX[i] = float(x\_diff.subs(t, T[i]))  
 VY[i] = float(y\_diff.subs(t, T[i]))  
 AX[i] = float(x\_diff2.subs(t, T[i]))  
 AY[i] = float(y\_diff2.subs(t, T[i]))  
  
# Модули скорости и ускорения  
v\_magnitudes = np.sqrt(VX\*\*2 + VY\*\*2)  
a\_magnitudes = np.sqrt(AX\*\*2 + AY\*\*2)  
  
max\_v = np.max(v\_magnitudes)  
max\_a = np.max(a\_magnitudes)  
  
# Создание графика  
fig, ax = plt.subplots()  
ax.axis('equal')  
a\_lim = 0.8  
ax.set\_xlim([min(X) - a\_lim, max(X) + a\_lim])  
ax.set\_ylim([min(Y) - a\_lim, max(Y) + a\_lim])  
  
point, = ax.plot([], [], 'go', markersize=10)  
ax.plot(X, Y, 'r-', lw=1)  
velocity\_line, = ax.plot([], [], 'b-', lw=1)  
velocity\_arrow\_head, = ax.plot([], [], 'b-')  
acceleration\_line, = ax.plot([], [], 'g-', lw=1)  
acceleration\_arrow\_head, = ax.plot([], [], 'g-')  
radius\_vector\_line, = ax.plot([], [], 'y-', lw=1)  
radius\_vector\_arrow\_head, = ax.plot([], [], 'y-')  
curvature\_radius\_line, = ax.plot([], [], 'm--', lw=1)  
curvature\_radius\_arrow\_head, = ax.plot([], [], 'm--')  
  
def rotate\_2d(x\_arr, y\_arr, angle):  
 x\_new = x\_arr \* np.cos(angle) - y\_arr \* np.sin(angle)  
 y\_new = x\_arr \* np.sin(angle) + y\_arr \* np.cos(angle)  
 return x\_new, y\_new  
  
def update(frame):  
 x0 = X[frame]  
 y0 = Y[frame]  
 vx = VX[frame]  
 vy = VY[frame]  
 ax0 = AX[frame]  
 ay0 = AY[frame]  
  
 point.set\_data([x0], [y0])  
  
 v\_mag = math.sqrt(vx\*\*2 + vy\*\*2)  
 a\_mag = math.sqrt(ax0\*\*2 + ay0\*\*2)  
  
 v\_scale = v\_mag / max\_v if max\_v != 0 else 0  
 a\_scale = a\_mag / max\_a if max\_a != 0 else 0  
  
 if v\_mag != 0:  
 vx\_norm = vx / v\_mag  
 vy\_norm = vy / v\_mag  
 else:  
 vx\_norm, vy\_norm = 0, 0  
  
 if a\_mag != 0:  
 ax\_norm = ax0 / a\_mag  
 ay\_norm = ay0 / a\_mag  
 else:  
 ax\_norm, ay\_norm = 0, 0  
  
 vx\_draw = vx\_norm \* v\_scale  
 vy\_draw = vy\_norm \* v\_scale  
 ax\_draw = ax\_norm \* a\_scale  
 ay\_draw = ay\_norm \* a\_scale  
  
 velocity\_line.set\_data([x0, x0 + vx\_draw], [y0, y0 + vy\_draw])  
 angle\_v = math.atan2(vy\_draw, vx\_draw)  
 arrow\_x = np.array([-0.08, 0, -0.08])  
 arrow\_y = np.array([0.04, 0, -0.04])  
 VArrowX, VArrowY = rotate\_2d(arrow\_x, arrow\_y, angle\_v)  
 velocity\_arrow\_head.set\_data(  
 VArrowX + x0 + vx\_draw, VArrowY + y0 + vy\_draw)  
  
 acceleration\_line.set\_data([x0, x0 + ax\_draw], [y0, y0 + ay\_draw])  
 angle\_a = math.atan2(ay\_draw, ax\_draw)  
 AArrowX, AArrowY = rotate\_2d(arrow\_x, arrow\_y, angle\_a)  
 acceleration\_arrow\_head.set\_data(  
 AArrowX + x0 + ax\_draw, AArrowY + y0 + ay\_draw)  
  
 # Рисуем радиус-вектор  
 radius\_vector\_line.set\_data([0, x0], [0, y0])  
 angle\_r = math.atan2(y0, x0)  
 RArrowX, RArrowY = rotate\_2d(arrow\_x, arrow\_y, angle\_r)  
 radius\_vector\_arrow\_head.set\_data(RArrowX + x0, RArrowY + y0)  
  
 numerator = (vx\*\*2 + vy\*\*2)\*\*1.5  
 denominator = abs(vx \* ay0 - vy \* ax0)  
 if denominator != 0:  
 R\_curv = numerator / denominator  
 else:  
 R\_curv = np.inf  
  
 norm\_vx = -vy  
 norm\_vy = vx  
 norm = np.hypot(norm\_vx, norm\_vy)  
 if norm != 0:  
 norm\_vx /= norm  
 norm\_vy /= norm  
  
 center\_x = x0 + R\_curv \* norm\_vx  
 center\_y = y0 + R\_curv \* norm\_vy  
  
 curvature\_radius\_line.set\_data([x0, center\_x], [y0, center\_y])  
 angle\_c = math.atan2(center\_y - y0, center\_x - x0)  
 CArrowX, CArrowY = rotate\_2d(arrow\_x, arrow\_y, angle\_c)  
 curvature\_radius\_arrow\_head.set\_data(  
 CArrowX + center\_x, CArrowY + center\_y)  
  
 return (point,  
 velocity\_line, velocity\_arrow\_head,  
 acceleration\_line, acceleration\_arrow\_head,  
 radius\_vector\_line, radius\_vector\_arrow\_head,  
 curvature\_radius\_line, curvature\_radius\_arrow\_head)  
  
ani = animation.FuncAnimation(  
 fig, update, frames=len(T), interval=20, blit=True)  
plt.show()

*Процесс выполнения работы:*

В данном коде вычисления скоростей и ускорений точек выполняются через нахождение производных функций r⋅cos(ϕ) и r⋅sin(ϕ) где r и ϕ — заданные функции от времени. Это позволяет получить проекции скоростей и ускорений на оси x и y.

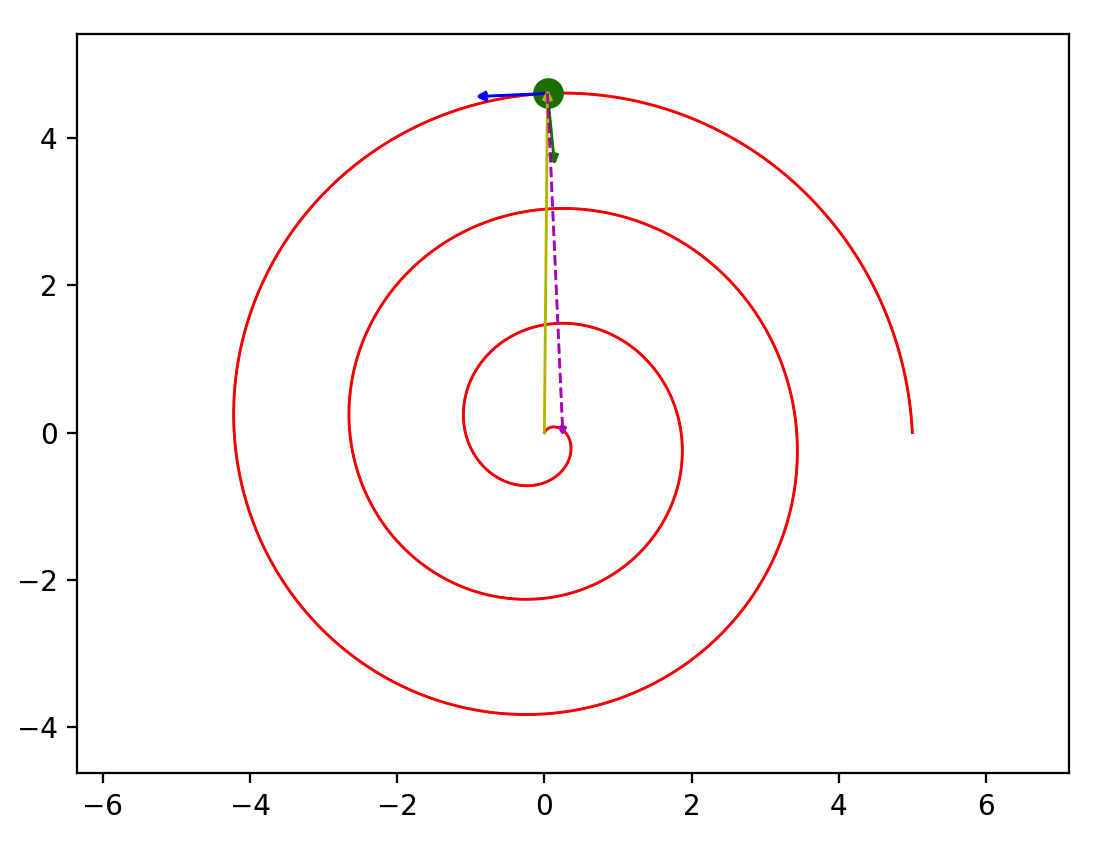
Векторы скорости, ускорения и нормали визуализируются с помощью библиотеки `matplotlib`. На каждом шаге анимации строятся линии, соединяющие текущее положение точки с её конечной точкой вектора.

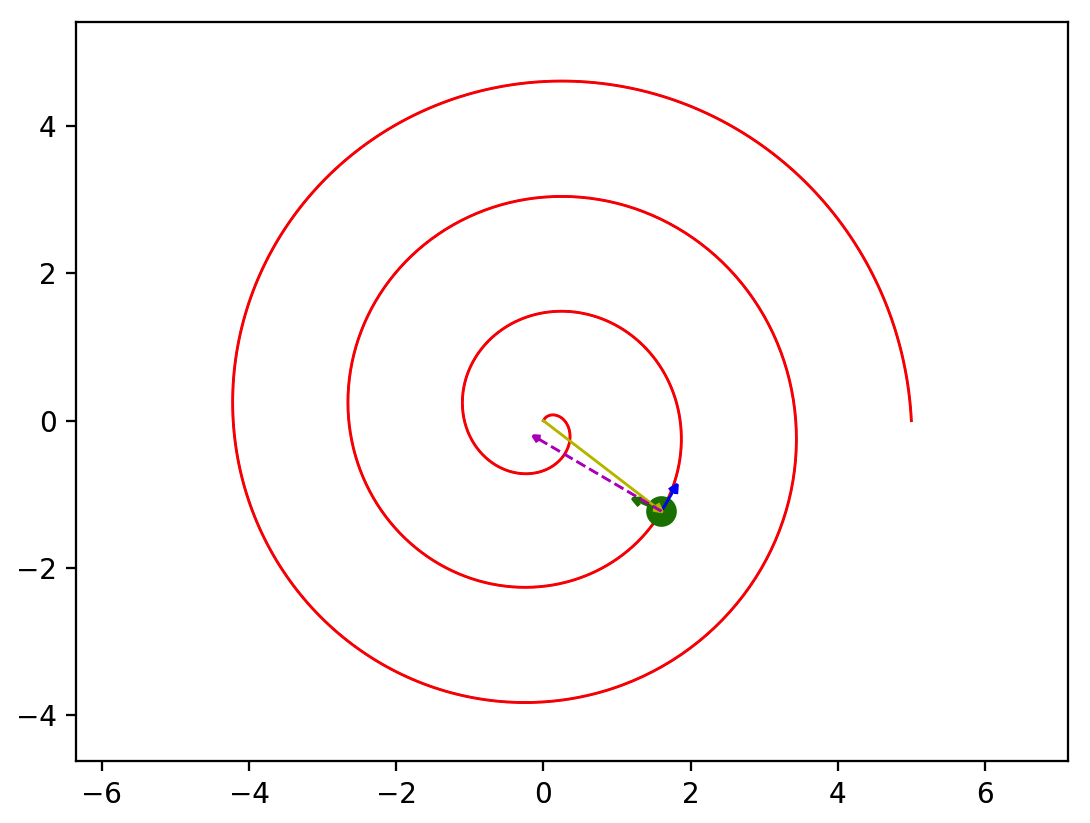
Для отображения стрелок используется функция `vect\_arrow`, которая формирует координаты стрелки через матрицу поворота, где угол определяется как арктангенс отношения координат вектора. Это позволяет корректно ориентировать стрелки относительно направлений векторов скорости, ускорения и радиуса нормали.

Вектор нормали рассчитывается как поворот вектора скорости на 90° против часовой стрелки, а его координаты нормализуются делением на длину скорости. Визуализация вектора нормали добавляет информацию о кривизне траектории точки.

Анимация создаётся с помощью `FuncAnimation`, где на каждом кадре обновляются положения точки, её векторов скорости, ускорения и нормали. График обновляется с учётом времени, обеспечивая плавное движение точки по траектории.

*Результат работы программы:*





На данных изображениях можно увидеть траекторию точки, радиус-вектор к ней из центра фигуры, векторы скорости и ускорения точки, а также вектор кривизны траектории точки в текущий момент времени.

*Вывод:* Научился использованию средств языка Python для моделирования движения точки по некоторой траектории в течение промежутка времени. Применил данные знания на практике при выполнении лабораторной работы.