ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**«АНИМАЦИЯ ТОЧКИ»**

**ПО ДИСЦИПЛИНЕ «ОСНОВЫ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ ДИНАМИЧЕСКИХ СИСТЕМ»**

**ВАРИАНТ ЗАДАНИЯ №23**

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

Фокин Л.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

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

подпись, дата

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

Москва, 2024

**Вариант №23 «Фантастическая кривая»**

**Задание:**

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

**Закон движения точки:**

**Текст программы**

import numpy as np

import sympy as sp

import matplotlib.pyplot as plt

import matplotlib.animation as animation

import math

# Определяем временные точки и символическую переменную времени

time\_points = np.linspace(0, 10, 1000)

radius = 1

t = sp.symbols('t')

# Параметрические уравнения

r\_expr = 1 - sp.sin(t)

fi\_expr = 2 \* t

# Преобразование в декартовы координаты

x\_expr = radius \* r\_expr \* sp.cos(fi\_expr)

y\_expr = radius \* r\_expr \* sp.sin(fi\_expr)

# Вычисление скорости (первая производная)

vx\_expr = sp.diff(x\_expr, t)

vy\_expr = sp.diff(y\_expr, t)

# Вычисление ускорения (вторая производная)

ax\_expr = sp.diff(vx\_expr, t)

ay\_expr = sp.diff(vy\_expr, t)

# Преобразование в функции numpy для численной оценки

x\_func = sp.lambdify(t, x\_expr, 'numpy')

y\_func = sp.lambdify(t, y\_expr, 'numpy')

vx\_func = sp.lambdify(t, vx\_expr, 'numpy')

vy\_func = sp.lambdify(t, vy\_expr, 'numpy')

ax\_func = sp.lambdify(t, ax\_expr, 'numpy')

ay\_func = sp.lambdify(t, ay\_expr, 'numpy')

# Оценка функций

x\_vals = x\_func(time\_points)

y\_vals = y\_func(time\_points)

vx\_vals = vx\_func(time\_points)

vy\_vals = vy\_func(time\_points)

ax\_vals = ax\_func(time\_points)

ay\_vals = ay\_func(time\_points)

# Настройка графика для анимации траектории

fig, ax = plt.subplots(*figsize*=(8, 6))

ax.axis('equal')

ax.set\_xlim([np.min(x\_vals) - 0.8, np.max(x\_vals) + 0.8])

ax.set\_ylim([np.min(y\_vals) - 0.8, np.max(y\_vals) + 0.8])

# Инициализация элементов графика для анимации траектории

point, = ax.plot([], [], 'go', *markersize*=10)

ax.plot(x\_vals, y\_vals, 'r-', *lw*=1)

velocity\_line, = ax.plot([], [], 'b-', *lw*=1)

velocity\_arrow\_head, = ax.plot([], [], 'b-', *lw*=1)

acceleration\_line, = ax.plot([], [], 'g-', *lw*=1)

acceleration\_arrow\_head, = ax.plot([], [], 'g-', *lw*=1)

radius\_vector\_line, = ax.plot([], [], 'y-', *lw*=1)

radius\_vector\_arrow\_head, = ax.plot([], [], 'y-', *lw*=1)

curvature\_radius\_line, = ax.plot([], [], 'm--', *lw*=1)

curvature\_radius\_arrow\_head, = ax.plot([], [], 'm--', *lw*=1)

# Настройка отдельного графика для 3 функций

fig2, (ax3, ax4, ax5) = plt.subplots(3, 1, *figsize*=(10, 12))

# Построение временных функций для позиции, скорости и ускорения

x\_line, = ax3.plot([], [], *label*="x(t)", *color*='r')

y\_line, = ax3.plot([], [], *label*="y(t)", *color*='b')

vx\_line, = ax4.plot([], [], *label*="vx(t)", *color*='g')

vy\_line, = ax4.plot([], [], *label*="vy(t)", *color*='purple')

ax\_line, = ax5.plot([], [], *label*="ax(t)", *color*='orange')

ay\_line, = ax5.plot([], [], *label*="ay(t)", *color*='pink')

# Подписи и легенды

ax3.set\_ylabel('Позиция')

ax4.set\_ylabel('Скорость')

ax5.set\_ylabel('Ускорение')

ax5.set\_xlabel('Время (t)')

ax3.legend()

ax4.legend()

ax5.legend()

# Установка масштаба для каждого графика

ax3.set\_xlim([0, 10])

ax3.set\_ylim([np.min(x\_vals) - 0.8, np.max(x\_vals) + 0.8])

ax4.set\_xlim([0, 10])

ax4.set\_ylim([np.min(vx\_vals) - 0.8, np.max(vx\_vals) + 0.8])

ax5.set\_xlim([0, 10])

ax5.set\_ylim([np.min(ax\_vals) - 0.8, np.max(ax\_vals) + 0.8])

def rotate\_vectors(*x\_arr*, *y\_arr*, *angle*):

    # Поворот векторов на заданный угол

    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*):

    # Получение текущей позиции, скорости, ускорения и других векторов

    x0 = x\_vals[*frame*]

    y0 = y\_vals[*frame*]

    vx = vx\_vals[*frame*]

    vy = vy\_vals[*frame*]

    ax0 = ax\_vals[*frame*]

    ay0 = ay\_vals[*frame*]

    # Обновление позиции движущейся точки

    point.set\_data([x0], [y0])

    # Обновление вектора скорости

    velocity\_line.set\_data([x0, x0 + vx], [y0, y0 + vy])

    velocity\_angle = math.atan2(vy, vx)

    arrow\_x = np.array([-0.08, 0, -0.08])

    arrow\_y = np.array([0.04, 0, -0.04])

    VArrowX, VArrowY = rotate\_vectors(arrow\_x, arrow\_y, velocity\_angle)

    velocity\_arrow\_head.set\_data(VArrowX + x0 + vx, VArrowY + y0 + vy)

    # Обновление вектора ускорения

    acceleration\_line.set\_data([x0, x0 + ax0], [y0, y0 + ay0])

    acceleration\_angle = math.atan2(ay0, ax0)

    AArrowX, AArrowY = rotate\_vectors(arrow\_x, arrow\_y, acceleration\_angle)

    acceleration\_arrow\_head.set\_data(AArrowX + x0 + ax0, AArrowY + y0 + ay0)

    # Обновление радиус-вектора

    radius\_vector\_line.set\_data([0, x0], [0, y0])

    radius\_angle = math.atan2(y0, x0)

    RArrowX, RArrowY = rotate\_vectors(arrow\_x, arrow\_y, radius\_angle)

    radius\_vector\_arrow\_head.set\_data(RArrowX + x0, RArrowY + y0)

    # Вычисление радиуса кривизны

    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

    # Вычисление центра кривизны

    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

    # Обновление вектора радиуса кривизны

    curvature\_radius\_line.set\_data([x0, center\_x], [y0, center\_y])

    curvature\_angle = math.atan2(center\_y - y0, center\_x - x0)

    CArrowX, CArrowY = rotate\_vectors(arrow\_x, arrow\_y, curvature\_angle)

    curvature\_radius\_arrow\_head.set\_data(CArrowX + center\_x, CArrowY + center\_y)

    # Обновление временных графиков

    x\_line.set\_data(time\_points[:*frame*], x\_vals[:*frame*])

    y\_line.set\_data(time\_points[:*frame*], y\_vals[:*frame*])

    vx\_line.set\_data(time\_points[:*frame*], vx\_vals[:*frame*])

    vy\_line.set\_data(time\_points[:*frame*], vy\_vals[:*frame*])

    ax\_line.set\_data(time\_points[:*frame*], ax\_vals[:*frame*])

    ay\_line.set\_data(time\_points[:*frame*], ay\_vals[:*frame*])

    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,

            x\_line, y\_line, vx\_line, vy\_line, ax\_line, ay\_line)

# Создание анимации

ani = animation.FuncAnimation(fig, update, *frames*=len(time\_points), *interval*=20, *blit*=True)

# Отображение графиков

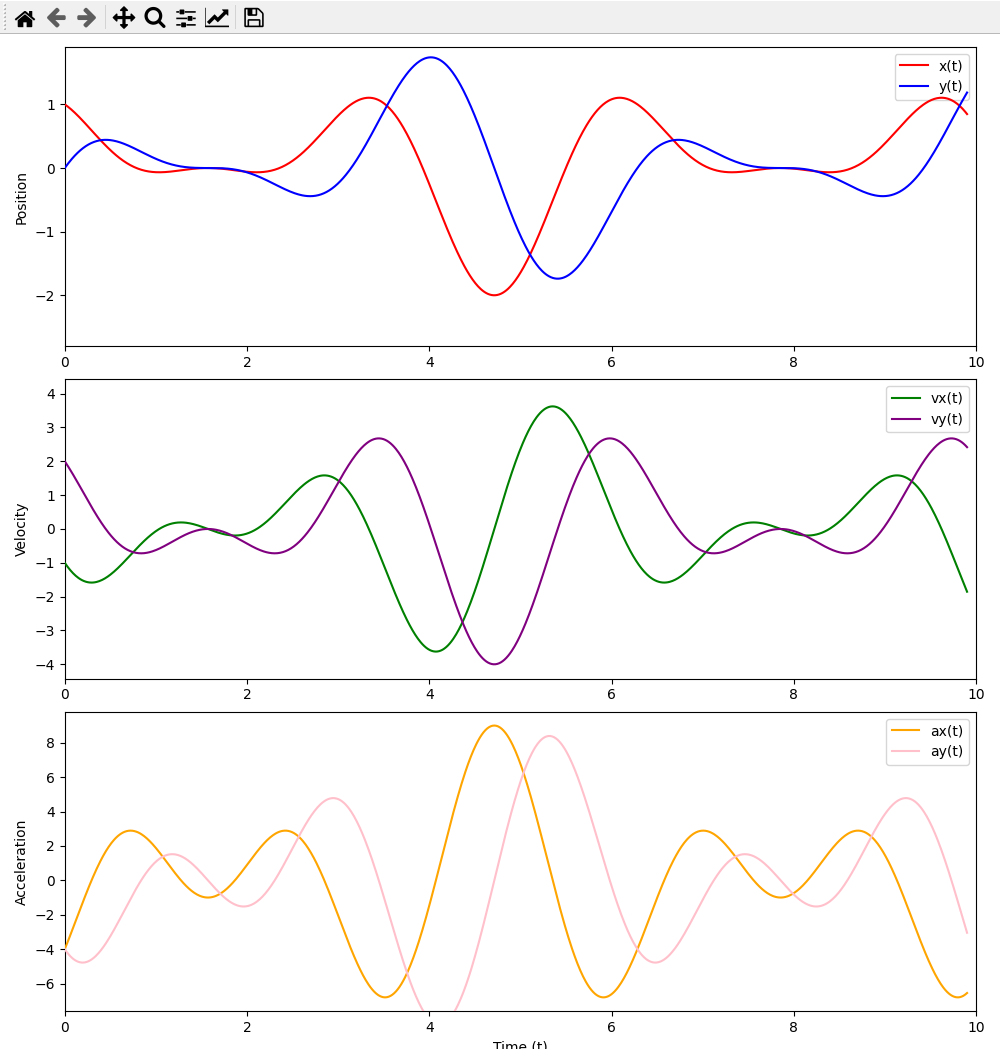
plt.tight\_layout()

plt.show()

**Результат работы программы:**

|  |  |
| --- | --- |
|  |  |

**Результаты работы программы:**



**Вывод:** в ходе выполнения лабораторной работы была разработана программа для моделирования движения точки по заданной траектории с использованием параметрических уравнений. Для реализации задачи использовались библиотеки Python, такие как numpy для работы с массивами и численных вычислений, sympy для символьных вычислений (нахождения производных и преобразования выражений), а также matplotlib для визуализации результатов.

Основная цель работы заключалась в построении траектории движения точки, отображении векторов скорости, ускорения и радиус-вектора, а также визуализации радиуса кривизны траектории. Для этого были вычислены первая и вторая производные координат точки, что позволило определить скорость и ускорение. Анимация движения точки была реализована с помощью библиотеки matplotlib.animation, что позволило динамически отображать изменение положения точки и соответствующих векторов.

Также были построены графики зависимости координат, скорости и ускорения от времени, что дало полное представление о динамике движения точки. В результате работы программы была получена наглядная визуализация движения точки по "фантастической кривой".