**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «информатика»**

## Тема: Основные управляющие конструкции языка Python

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3341 |  | БАЙРАМ Э. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2023

## Цель работы

## Код написан для решения математических задач в рамках лабораторной работы. Он включает три функции, которые решают задачи, связанные с вычислениями и геометрией. Каждая функция выполняет свою задачу, используя библиотеку NumPy для математических операций. Этот код предназначен для автоматизации математических расчетов и может быть полезен в робототехнике и научных исследованиях.

## Задание

**Вариант 1**

**Задача 1**: Нахождение возможной точки столкновения для двух ботов: В этой задаче два робота движутся к пересечению двух траекторий. Для предотвращения столкновения им необходимо знать точку пересечения траекторий. Задача состоит в нахождении этой точки. Функция check\_collision принимает на вход коэффициенты уравнений прямых (bot1 и bot2) и возвращает точку пересечения этих прямых в виде кортежа с округленными значениями.

**Задача 2**: Нахождение уравнения плоскости, через которую проходят боты: Здесь требуется найти уравнение плоскости, через которую проходят три заданные точки. Функция check\_surface принимает координаты этих трех точек и возвращает коэффициенты уравнения плоскости (a, b, c) в виде массива с округленными значениями.

**Задача 3**: Вычисление вращения бота: Эта задача связана с вращением робота вокруг своей оси (вокруг оси z) на заданный угол. Функция check\_rotation принимает текущие координаты бота и угол вращения (в радианах) и возвращает новые координаты бота после вращения с округленными значениями.

## Выполнение работы

1. **check\_collision(bot1, bot2):**

-Сначала извлекаются коэффициенты прямых из аргументов ‘bot1’ и ‘bot2’. Каждый ‘bot’ представляет собой кортеж из трех чисел ‘(a, b, c)’, где ‘a’ и ‘b’ - коэффициенты при переменных ‘x’ и ‘y’ в уравнении прямой, а ‘c’ - свободный член.

-Затем создается матрица коэффициентов ‘coefficient\_matrix’, состоящая из коэффициентов ‘a’ и ‘b’ обоих прямых.

-Также создается вектор свободных членов ‘constants’, состоящий из ‘-c’ каждой из прямых.

-Затем код пытается решить систему линейных уравнений с использованием ‘np.linalg.solve’ и находит точку пересечения прямых.

## --Если решение возможно, результат округляется до двух десятичных знаков и возвращается в виде кортежа. Если решение невозможно (например, прямые параллельны), функция возвращает ‘None’.

## check\_surface(point1, point2, point3):

## -Задаются три точки в трехмерном пространстве с координатами ‘point1’, ‘point2’ и ‘point3’.

## -Создается матрица коэффициентов ‘coefficients\_matrix’, в которой первый столбец - x-координаты точек, второй столбец - y-координаты точек, и третий столбец - единицы (для свободных членов).

## -Вычисляется ранг ‘rank\_coefficients’ матрицы коэффициентов.

## -Создается вектор свободных членов ‘free\_terms’ из z-координат точек.

## -Если ранг матрицы равен 3 (то есть точки не коллинеарны), функция пытается решить систему уравнений и находит коэффициенты уравнения плоскости ‘abc’.

## -Результат округляется до двух десятичных знаков и возвращается в виде массива. Если уравнение плоскости невозможно найти (например, точки коллинеарны), функция возвращает ‘None’.

## check\_rotation(coordinates, angle):

## -Функция принимает координаты точек в виде массива ‘coordinates’ и угол вращения ‘angle’ (в радианах).

## -Создается матрица вращения ‘rotation\_matrix’, которая выполняет вращение вокруг z-оси на заданный угол.

## -Выполняется умножение матрицы вращения на координаты точек с помощью ‘np.dot’.

## -Результат округляется до двух десятичных знаков и возвращается в виде массива координат после вращения.

## Тестирование

Результаты тестирования представлены в табл. 1

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | |
| *check\_collision* | bot1 = (2, 1, -3)  bot2 = (1, 2, 4) | (2.0, -1.0) | |
| *check\_surface* | point1 = (1, 2, 3)  point2 = (4, 5, 6)  point3 = (7, 8, 9) | [0. 0. 3.] | |
| *check\_rotation* | coordinates = np.array([3, 4, 1])  angle = np.pi / 4 | | [4.95 0.05 1.] |

## Выводы

**1.check\_collision(bot1, bot2):**

-Эта функция решает задачу нахождения точки пересечения двух прямых (представленных уравнениями ax + by + c = 0).

-Если прямые пересекаются, функция возвращает координаты точки пересечения в виде кортежа, округленные до двух десятичных знаков.

-Если пересечение отсутствует (например, прямые параллельны), функция возвращает ‘None’.

**2.check\_surface(point1, point2, point3):**

-Эта функция решает задачу нахождения уравнения плоскости, проходящей через три заданные точки в трехмерном пространстве.

-Если заданные точки не коллинеарны, функция возвращает коэффициенты уравнения плоскости в виде массива, округленные до двух десятичных знаков.

-Если уравнение плоскости невозможно найти (например, точки коллинеарны), функция возвращает ‘None’.

**3.check\_rotation(coordinates, angle):**

-Эта функция решает задачу вращения точек в 2D-пространстве вокруг начала координат на заданный угол (в радианах).

-Функция возвращает координаты точек после вращения, округленные до двух десятичных знаков.

# **Приложение А Исходный код программы**

import numpy as np

def check\_collision(bot1, bot2):

    a1, b1, c1 = bot1

    a2, b2, c2 = bot2

    coefficient\_matrix = np.array([[a1, b1], [a2, b2]])

    constants = np.array([-c1, -c2])

    try:

        intersection = np.linalg.solve(coefficient\_matrix, constants)

        intersection = np.round(intersection, 2)

        return tuple(intersection)

    except np.linalg.LinAlgError:

        return None

def check\_surface(point1, point2, point3):

    points\_matrix = np.array([point1, point2, point3])

    coefficients\_matrix = np.column\_stack(

        (points\_matrix[:, 0], points\_matrix[:, 1], np.ones(3))

    )

    rank\_coefficients = np.linalg.matrix\_rank(coefficients\_matrix)

    free\_terms = points\_matrix[:, 2]

    if rank\_coefficients == 3:

        abc = np.linalg.solve(coefficients\_matrix, free\_terms)

        return np.round(abc, 2)

    else:

        return None

def check\_rotation(coordinates, angle):

    rotation\_matrix = np.array(

        [

            [np.cos(angle), -np.sin(angle), 0],

            [np.sin(angle), np.cos(angle), 0],

            [0, 0, 1],

        ]

    )

    rotated\_coordinates = np.dot(rotation\_matrix, coordinates)

    rotated\_coordinates = np.round(rotated\_coordinates, 2)

    return rotated\_coordinates