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

Студент гр. 3341 _____ Перевалов П.И.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Изучение основных управляющих конструкций в языке Python, модуля *numpy.*

Задание

Вариант 1

Вариант лабораторной работы состоит из 3 задач, оформите каждую задачу в виде отдельной функции согласно условиям задач. Приветствуется использование модуля *numpy*, в частности пакета *numpy.linalg*. Вы можете реализовывать вспомогательные функции, главное - использовать те же названия основных функций, что требуются в задании. Сами функции вызывать не надо, это делает за вас проверяющая система.

Задача 1. Содержательная постановка задачи

Два дакибота приближаются к перекрестку. Чтобы избежать столкновения, им необходимо знать точку пересечения их траекторий движения. Траектории - линейные, и дакиботы уже вычислили коэффициенты этих уравнений. Ваша задача - помочь ботам вычислить точку потенциального столкновения.

Формальная постановка задачи

Оформите решение в виде отдельной функции *check_collision*. На вход функции подаются два *ndarray* - коэффициенты *bot1*, *bot2* уравнений прямых *bot1 = (a1, b1, c1), bot2 = (a2, b2, c2)* (уравнение прямой имеет вид *ax+by+c=0*).

Функция должна возвращать точку пересечения траекторий (кортеж из 2 значений), предварительно округлив координаты до 2 знаков после запятой с помощью *round(value, 2)*.

Задача 2. Содержательная часть задачи

Три дакибота начали движение, отъехали от условной точки старта и через некоторое время остановились. Каждый дакибот уже вычислил свою координату относительно точки старта. Дакиботам нужно передать на базу карту местностности, по которой они двигались. Для построения карты местности необходимо знать уравнение плоскости. Ваша задача - помочь дакиботам найти уравнение плоскости, в которой они двигались.

Формальная постановка задачи

Оформите задачу как отдельную функцию *check_surface*, на вход которой передаются координаты 3 точек (3 *ndarray* 1 на 3): *point1, point2, point3*. Функция

должна возвращать коэффициенты *a, b, c* в виде *ndarray* для уравнения плоскости вида *ax+by+c=z*. Перед возвращением результата выполнение округление каждого коэффициента до 2 знаков после запятой с помощью *round(value, 2)*.

Задача 3. Содержательная часть задачи

Дакибот выехал на перекресток и готовится к выполнению поворота вокруг своей оси (вокруг оси *z*), чтобы продолжить движение в другом направлении. Он знает свои координаты и знает угол поворота (в радианах). Помогите дакиботу повернуться в нужное направление для продолжения движения.

Формальная постановка задачи

Оформите решение в виде отдельной функции *check_rotation*. На вход функции подаются *ndarray* 3-х координат дакибота и угол поворота. Функция возвращает повернутые ndarray координаты, каждая из которых округлена до 2 знаков после запятой с помощью *round(value, 2)*.

Выполнение работы

Функции:

- *check_collision* принимает на вход два *ndarray* коэффициенты *bot1*, *bot2* уравнений прямых *bot1 = (a1, b1, c1), bot2 = (a2, b2, c2)*, возвращает точку пересечения траекторий (кортеж из 2 значений) или *None* если решение невозможно.
- *check_surface* принимает на вход координаты 3 точек (3 *ndarray* 1 на 3): *point1, point2, point3*, возвращает коэффициенты *a, b, c* в виде *ndarray* для уравнения плоскости вида *ax+by+c=z* или *None* если решение невозможно.
- *check_rotation* принимает на вход *ndarray* 3-х координат *vec* дакибота и угол поворота *rad*, возвращает повернутые *ndarray* координаты.

В функции *check_collision:*

Проверяется, не являются ли линии параллельными, сравнивая отношения **bot1[0] / bot2[0]** и **bot1[1] / bot2[1]**. Если они равны, это означает, что линии параллельны, и функция возвращает **None.**

В другом случае, находятся координаты точки пересечения **(x, y)** с использованием метода Крамера, и затем результат округляется до двух цифр после запятой.

В функции *check_surface:*

С помощью функции **np.vstack** и **np.concatenate**. Создаем матрицу ‘M’, состоящую из координат точек **point1**, **point2** и **point3.**

Преобразуем матрицу ‘M’ и вектор ‘v’ к типу float.

С помощью функции **np.linalg.matrix_rank(M)** вычисляем ранг матрицы.

Если ранг матрицы **M** равен количеству строк матрицы **M** и также равен количеству столбцов матрицы **M, то** точки **point1**, **point2** и **point3** лежат на одной плоскости, и функция продолжает выполнение.

Находим параметры уравнений плоскости, на которой лежат точки и округляем результат до цифр знаков после запятой.

В функции *check_rotation:*

Создается матрица **turn**, которая представляет собой матрицу поворота вокруг оси Z на угол **rad.** Эта матрица 3x3 использует тригонометрические функции **cos** и **sin** для выполнения поворота.

С помощью функции **np.dot** умножаем вектор **vec** на матрицу **turn**, из-за чего происходит переворот вектора. Затем **result** округляется до двух цифр после запятой.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	**bot1= np.array([2, 1, 3])** **bot2 = np.array([4, 2, 6])**	**(0.75, 2.0)**	
2.	**bot1 = np.array([1, 1, 2])**	**None**	

	bot2 = np.array([3, 3, 6])		
3.	**point1=np.array([1, 2, 3])** **point2=np.array([3, 5, 2])** **point3=np.array([2, 1, 5])**	**[0.29, 0.29, 2.14]**	
4.	**point1=np.array([1, 2, 3])** **point2=np.array([4, 5, 6])** **point3=np.array([7, 8, 9])**	**None**	
5.	**vec = np.array([0, 1, 0])** **rad = 0.785**	**[0.0, 1.0, 0.0]**	

Выводы

В результате выполнения лабораторной работы были изучены и применены некоторые функции модуля NumPy в Python при решении практической задачи, а так же реализовано 3 функции для решения поставленных задач.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np


def check_collision(bot1, bot2):
    if (bot1[0] / bot2[0] == bot1[1] / bot2[1]):
        return None
    x = (bot1[2]*bot2[1] - bot2[2]*bot1[1]) / (bot2[0]*bot1[1] -
bot1[0]*bot2[1])
    y = (-bot1[2] - bot1[0]*x) / bot1[1]
    return np.round(x, 2)+0, np.round(y, 2)+0


def check_surface(point1, point2, point3):
    M = np.vstack( (np.concatenate((point1[:-1], [1])),
                    np.concatenate((point2[:-1], [1])),
                    np.concatenate((point3[:-1], [1]))) )
    M = np.array( M, float )
    v = np.array( [point1[2], point2[2], point3[2]], float )

    rank = np.linalg.matrix_rank(M)
    if (not(rank == M.shape[0] and rank == M.shape[1])):
        return None

    result = np.linalg.solve(M, v)
    result[0] = np.round(result[0], 2)
    result[1] = np.round(result[1], 2)
    result[2] = np.round(result[2], 2)
    return result


def check_rotation(vec, rad):
    turn = np.array([[np.cos(rad), np.sin(rad), 0],
                     [-np.sin(rad), np.cos(rad), 0],
                     [0, 0, 1]])

    result = np.dot(vec, turn)
    result[0] = np.round(result[0], 2)
    result[1] = np.round(result[1], 2)
    result[2] = np.round(result[2], 2)
    return result
```