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. 2381 _____ Рыжиков И.С.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2022

Цель работы

Изучение основных управляющих конструкций языка Python, методов и классов библиотеки *numpy* для работы с элементами линейной алгебры, в частности пакет *numpy.linalg*.

Применить полученные знания для решения практической задачи.

Задание

Вариант №1

Задача 1.

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)*.

Пример входных данных:

array([-3, -6, 9]), array([8, -7, 0])

Пример возвращаемого результата:

(0.91, 1.04)

Задача 2.

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)*.

Например, даны точки: A(1, -6, 1); B(0, -3, 2); C(-3, 0, -1). Подставим их в уравнение плоскости:

$$a \cdot 1 + b(-6) + c = 1$$
$$a \cdot 0 + b(-3) + c = 2$$
$$a(-3) + b \cdot 0 + c = -1$$

Составим матрицу коэффициентов:

$$\begin{pmatrix} 1 & -6 & 1 \\ 0 & -3 & 1 \\ -3 & 0 & 1 \end{pmatrix}$$

Вектор свободных членов:

$$\begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$$

Для такой системы уравнение плоскости имеет вид: *z = 2x + 1y + 5*

Пример входных данных:

array([1, -6, 1]), array([0, -3, 2]), array([-3, 0, -1])

Возвращаемый результат:

[2. 1. 5.]

Задача 2.

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)*.

Пример входных аргументов:

array([1, -2, 3]), 1.57

Пример возвращаемого результата:

[2. 1. 3.]

Выполнение работы

Импортируем модуль *numpy* и его пакет *linalg*, переименовывая их в *np* и *LA* соответственно.

Функция check_collision

Принимает на вход два ndarray — коэффициенты bot1, bot2 уравнений прямых bot1 = (a1, b1, c1), bot2 = (a2, b2, c2) (уравнение прямой имеет вид ax+by+c=0).

Возвращает точку пересечения траекторий (кортеж из 2 значений), предварительно округлив координаты до 2 знаков после запятой.

Подготавливаем введенные данные для решения задачи с помощью функции функция *solve* из пакета *numpy.linalg,* а именно: создаем матрицу системы *mx* и столбец свободных членов *arr*.

Находим решение системы $mx \cdot \begin{pmatrix} x \\ y \end{pmatrix} = arr$ с помощью *numpy.linalg.solve*, которое на самом деле является исходной точкой пересечения траекторий.

Возвращаем результат, округлив числа до требуемого количества знаков после запятой и преобразовав ответ к нужному типу.

Функция check_surface

Принимает на вход координаты 3 точек: *point1*, *point2*, *point3* (точка — ndarray 1 на 3).

Возвращает округленные до 2 знаков после запятой коэффициенты a, b, c в виде ndarray для уравнения плоскости вида *ax+by+c=z*.

Подготавливаем введенные данные для решения задачи:

- создаем матрицу системы *mx;*
- последний столбец матрицы заполняем единицами;

- создаем столбец свободных членов *arr*.

Пробуем решить систему:

- если получается, то тогда возвращаем решение, округлив числа до требуемого количества знаков после запятой
- если нет — перехватываем ошибку *LinAlgError*, обозначающею, что матрица системы вырожденная и решение найти невозможно, в таком случаи возвращаем *None*.

Функция check_rotation

Принимает на вход *ndarray* 3-х координат дакибота и угол поворота в радианах.

Возвращает повернутые *ndarray* координаты, округленные до 2 знаков после запятой.

Создаем матрицу поворота на угол φ в плоскости xy: $\begin{pmatrix} cos\varphi & -sin\varphi & 0 \\ sin\varphi & cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

Возвращаем результат умножение вектора на матрицу поворота, округлив числа до требуемого количества знаков после запятой.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
	Функция check_collision		
1.	np.array([-3, -6, 9]), np.array([8, -7, 0])	(0.91, 1.04)	ОК
2.	np.array([3, 1, 2]), np.array([-1, 6, 3])	(-0.47, -0.58)	ОК
	Функция check_surface		
3.	np.array([1, -6, 1]), np.array([0, -3, 2]), np.array([-3, 0, -1])	[2. 1. 5.]	ОК
	Функция check_rotation		
4.	np.array([1, -2, 3]), 1.57	[2. 1. 3.]	ОК
5.	np.array([1, -2, 3]), 1.13	[2.24 0.05 3.]	ОК

Выводы

Были изучены основные управляющие конструкция языка Python, методы и классы библиотеки numpy.

Разработаны функции нахождения точки пересечения траекторий, уравнения плоскости, в которой находятся заданные 3 точки, и повернутых на некоторый угол координат радиус вектора.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: *index_first_negative.c*

```
import numpy as np
from numpy import linalg as LA


def check_collision(bot1: np.ndarray, bot2: np.ndarray) -> tuple:
    mx = np.matrix((bot1[0:2],
                    bot2[0:2]))
    arr = -np.array((bot1[2], bot2[2]))
    solve = LA.solve(mx, arr)
    return tuple(np.round(solve, 2))


def check_surface(point1: np.ndarray, point2: np.ndarray, point3:
np.ndarray) -> np.ndarray:
    mx = np.matrix((point1[0:2],
                    point2[0:2],
                    point3[0:2]))
    mx = np.hstack((mx, np.ones((3, 1))))
    arr = np.array((point1[2], point2[2], point3[2]))
    try:
        return np.round(LA.solve(mx, arr), 2)
    except LA.LinAlgError: # matrix is Singular
        return None


def check_rotation(vec: np.ndarray, rad: float) -> np.ndarray:
    # Create rotation_matrix
    cos, sin = np.cos(rad), np.sin(rad)
    rotation_matrix = np.matrix([[cos, -sin, 0],
                                 [sin, cos, 0],
                                 [0, 0, 1]])

    return np.round(np.dot(rotation_matrix, vec), 2)[0]  # Just np.dot
return matrix
```