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. 3344 Клюкин А.В.

Преподаватель Иванов Д.В.

Санкт-Петербург

2023

Цель работы.

Изучение и освоение основных управляющих конструкций языка Python и модуля numpy с помощью практического применения в индивидуальном задании.

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.

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)

Примечание: помните про ранг матрицы и как от него зависит наличие решения системы уравнений. В случае, если решение найти невозможно (например, из-за линейно зависимых векторов), функция должна вернуть None.

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.

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:

Составим матрицу коэффициентов:

Вектор свободных членов:

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.]

Примечание: помните про ранг матрицы и как от него зависит существование решения системы уравнений. В случае, если решение найти невозможно (невозможно найти коэффициенты плоскости из-за, например, линейно зависимых векторов), функция должна вернуть None.

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.

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.

Было реализовано три функции.

check_collision(bot1, bot2), принимающая на вход два массива чисел, содержащие коэффициенты уравнений прямых. В ней берутся свободные коэффициенты и записываются в отдельный массив. Так же оставшиеся коэффициенты записываются в иной массив. После идет проверка на существование решения системы уравнений с использованием функции linalg.matrix_rank для поиска ранга матрицы. В случае удовлетворения условия, решаем систему уравнений функцией numpy - linalg.solve. А после возвращаются полученные результаты в виде кортежа.

check_surface, которая принимает координаты трёх точек в виде массива ndarray. Аналогичным образом разделяются свободные члены и остальные коэффициенты. После проходит проверка на существование решений через ранг матрицы и решение, если оно возможно. Затем полученный ответ округляется через round() и возвращается функцией

check_rotation, принимающая массив ndarray и переменную. Далее идет попытка расчета с использованием тригонометрических функций из библиотеки numpy - numpy.cos и .sin. В случае успешного расчета - возвращается результат

итоговый код см. в приложении А

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	array([-3, -6, 9]), array([8, -7, 0])	(0.91, 1.04)	Верно
2.	array([1, -6, 1]), array([0, -3, 2]), array([-3, 0, -1])	[2. 1. 5.]	Верно
3.	array([1, -2, 3]), 1.57	[2. 1. 3.]	Верно

Выводы.

На практике была изучена работа основных управляющих конструкций языка Python, некоторые функции модуля numpy, с написанием программы для индивидуального задания.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Klyukin_Aleksandr_lb1

```
import numpy as np


def check_collision(bot1, bot2):
    res1 = bot1[-1]
    res2 = bot2[-1]
    second = np.array([-res1, -res2])
    first = np.array([[*bot1[0:2]], [*bot2[0:2]]])
    if np.linalg.matrix_rank(first) == 2:
        ans = np.linalg.solve(first, second)
        ans[0] = round(ans[0], 2)
        ans[1] = round(ans[1], 2)
        return tuple(ans)
    else:
        return None


def check_surface(point1, point2, point3):
        a = np.array([[point1[0], point1[1], 1], [point2[0],
point2[1], 1], [point3[0], point3[1], 1]])
    b = np.array([point1[2], point2[2], point3[2]])
    if np.linalg.matrix_rank(a) == 3:
        ans = np.linalg.solve(a, b)
        for i in range(len(ans)):
            ans[i] = round(float(ans[i]), 2)
        return ans
    else:
        return None


def check_rotation(vec, rad):
    try:
            ans = np.array([round(vec[0] * np.cos(rad) - vec[1] *
np.sin(rad), 2),
                            round(vec[0] * np.sin(rad) + vec[1] *
np.cos(rad), 2), round(vec[2], 2)])
        return ans
    except ValueError:
        return None
```