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 _____ Ханнанов А.Ф.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы.

Ознакомиться с основными управляющими конструкциями Python и библиотеки Numpy.

Задание.

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)*.

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.

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)*.

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

a+b(-6)+c=1

a*0+b(-3)+c=2

a(-3)+b*0+c=-1

Составим матрицу коэффициентов:

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

Вектор свободных членов:

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

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.

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)*..
Пример входных аргументов:
array([1, -2, 3]), 1.57
Пример возвращаемого результата:
[2. 1. 3.]

Выполнение работы.

Функции:

1) check_collision(bot1, bot2) — Получает на вход уравнения движений двух ботов. Возвращает координаты точки пересечения траекторий.

2) check_surface(point1, point2, point3) — Получает на вход координаты трёх ботов. Возвращает коэффициенты уравнения плоскости.

3) check_rotation(vec, rad) — Получает на вход координаты бота и угол, на который он поворачивается. Возвращает изменённые координаты бота.

4) is_one_solution(system_matrix, extended_matrix) — Вспомогательная функция. Определяет количество решений системы уравнений. Если есть решения, то возвращает ранг функции, иначе возвращает None.

Основные переменные:

В решении применяются локальные переменные.

1) system_matrix — numpy массив, в нём хранится матрица системы уравнений.

2) free_members_matrix — numpy массив, в нём хранится матрица свободных членов системы уравнений.

3) extended_matrix — numpy массив, в нём хранится расширенная матрица системы, она нужна для нахождения количества решений.

4) coefficient_matrix — numpy массив, хранятся коэффициенты уравнения плоскости.

5) rotation_matrix — numpy массив, хранится транспонированная поворотная мартица.

6) matrix_for_changing_coordinates — numpy массив, хранятся изменяемые при повороте координаты.

7) rotated_matrix — numpy массив, хранятся координаты бота после поворота.

Использованные методы numpy:

- linalg.matrix_rank — определяет ранг матрицы
- array — создание массива numpy
- linalg.solve — решение системы линейных уравнений
- hstack — прибавление к матрице столбца

- round — округление значений матрицы
- transpose — получение транспонированной матрицы
- cos/sin — получение значений косинуса и синуса
- append — прибавление значения в конец массива

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
1.	array([-3, -6, 9]), array([8, -7, 0])	(0.91, 1.04)	При вызове check_collision
2.	array([1, -6, 1]), array([0, -3, 2]), array([-3, 0, -1])	[2. 1. 5.]	При вызове check_surface
3.	array([1, -2, 3]), 1.57	[2. 1. 3.]	При вызове check_rotation

Выводы.

В ходе выполнения лабораторной работы, были получены навыки работы с языком Pyhton, получены базовые умения для работы с матрицами при помощи библиотеки Numpy.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np


def is_one_solution(system, extended):
    if np.linalg.matrix_rank(system) == np.linalg.matrix_rank(extended):
        return np.linalg.matrix_rank(system)
    return False


def check_collision(bot1, bot2):
    # матрица системы:
    system_matrix = np.array([[bot1[0], bot1[1]], [bot2[0], bot2[1]]])
    # матрица свободных членов:
    free_members_matrix = np.array([bot1[2] * -1, bot2[2] * -1])
    # расширенная матрица системы:
    extended_matrix = np.array([[bot1[0], bot1[1], bot1[2] * -1], [bot2[0], bot2[1], bot2[2] * -1]])

    # проверка на наличие решения и возвращение результата
    if is_one_solution(system_matrix, extended_matrix) >= 2:
        x, y = np.linalg.solve(system_matrix, free_members_matrix)
        return round(x, 2), round(y, 2)
    return None


def check_surface(point1, point2, point3):
    # матрица системы
    system_matrix = np.array([[point1[0], point1[1], 1], [point2[0], point2[1], 1], [point3[0], point3[1], 1]])
    # матрица свободных членов
    free_members_matrix = np.array([[point1[2]], [point2[2]], [point3[2]]])
    # расширенная матрица
    extended_matrix = np.hstack((system_matrix, free_members_matrix))

    # проверка на наличие решения и возвращение результата
```

```
    if is_one_solution(system_matrix, extended_matrix) >= 3:
        coefficient_matrix = np.linalg.solve(system_matrix, free_members_matrix)
        return np.round(np.transpose(coefficient_matrix)[0], 2)
    return None


def check_rotation(vec, rad):
    # матрица поворота
    rotation_matrix = np.transpose(np.array([[np.cos(rad), np.sin(rad) * -1], [np.sin(rad), np.cos(rad)]]))
    # изменение координат
    x, y, z = vec
    matrix_for_changing_coordinates = np.array([[x], [y]])
    rotated_matrix = np.linalg.solve(rotation_matrix, matrix_for_changing_coordinates)
    rotated_matrix = np.append(np.transpose(rotated_matrix), z)
    return np.round(np.transpose(rotated_matrix), 2)
```