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: Управляющие конструкции языка Python

Студент гр. 3344 _____ Жаворонок Д.Н.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Освоение работы с управляющими конструкциями на языке Python с помощью модуля 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)*.

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).*

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)*.

Выполнение работы

Библиотека numpy была импортирована как np.

def check_collision(pos1, pos2) — функция, принимающая на вход *pos1, pos2* — ndarray'и, в которых хранятся – коэффициенты уравнений прямых *pos1 = (a1, b1, c1), pos2 = (a2, b2, c2)*. Создаются 2 ndarray — *a* и *b,* в которых хранятся коэффициенты *a1, b1, a2, b2* и *-c1, -c2* соответственно. Используя функцию *np.linalg.matrix_rank()* осуществлялся возврат *None*, в случае, если ранг матрицы оказывался меньше 2. Иначе возвращается кортеж состоящий из двух округленных с помощью *np.round()* чисел, полученных через *np.linalg.solve()*.

def check_surface(A, B, C) — функция, принимающая на вход A, B, C — ndarray'и, в которых хранятся координаты точек. Реализовано нахождение коэффициентов a, b, c для уравнения плоскости вида ax+by+c=z. Используя функцию *np.linalg.matrix_rank()* осуществлялся возврат None, в случае, если ранг матрицы оказывался меньше 3. Иначе возвращается кортеж состоящий из двух округленных с помощью *np.round()* чисел, полученных через *np.linalg.solve()*.

def check_rotation(pos, alfa) — функция, принимающая на вход *pos* — *ndarray*, в котором хранятся координаты дакибота, *alfa* — угол поворота. Используя *np.sin()* и *np.cos()* составляется матрица поворота по оси *z*. Используя *np.dot()* было выполнено перемножение координат дакибота и матрицы поворота, позже новые координаты возвращаются из функции округленными используя *np.round()*

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

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

Выводы

Была освоена работа с управляющими конструкциями на языке Python с помощью модуля numpy. Были использованы функции подключаемой внешней библиотеки.

Приложение А

Исходный код программы

Название файла: main.py

```
import numpy as np

def check_collision(pos1, pos2):
    a = np.array([pos1[:2], pos2[:2]])
    b = np.array([-pos1[2], -pos2[2]])

    if np.linalg.matrix_rank(a) < 2:
        return None

    return tuple(np.round(np.linalg.solve(a, b), 2))

def check_surface(A, B, C):
    m = np.hstack((np.array([A[:2], B[:2], C[:2]]), np.array([[1], [1],
[1]])))
    v = np.array([A[2], B[2], C[2]])

    if np.linalg.matrix_rank(m) < 3:
        return None

    return np.round(np.linalg.solve(m, v), 2)

def check_rotation(pos, alfa):
        rotation = np.array([[np.cos(alfa), -1 * np.sin(alfa)],
[np.sin(alfa), np.cos(alfa)]])
    return np.round(np.append(np.dot(rotation, pos[:2]), pos[2]), 2)
```