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 _____ Вердин К.К

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 местности, 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

Была импортирована библиотека *numpy*.

Была реализована функция *def check_collision(bot1, bot2)*, принимающая на вход два *ndarray* – коэффициенты *bot1*, *bot2* уравнений прямых *bot1* = (*a1, b1, c1*), bot2 = (*a2, b2, c2*). В функции создаются 2 ndarray a и b, в которых хранятся коэффициенты a1, b1, a2, b2 и -c1, -c2 (являются решениями уравнения ax+by) соответственно. В условном выражении *if* с помощью функции *np.linalg.matrix_rank(a)* было проверено, что ранг матрицы, состоящей из этих уравнений, имеет допустимое для нахождения решения значение. Если условие *if np.linalg.matrix_rank([bot1, bot2]) >= 2* выполнялось, то функция возвращала кортеж состоящий из двух чисел, которые были получены путём применения функции numpy.linalg.solve(a,b), предварительно округлённые до 2 знаков после запятой при помощи np.round(). Если условие не выполнилось, то функция возращала “None”.

В функции *def check_surface(point1, point2, point*3), принимающей на вход три *ndarray* – координаты дакиботов, было реализовано нахождение коэффициентов *a, b, c* в виде *ndarray* для уравнения плоскости вида *ax+by+c=z*. Были созданы массивы вида *ndarray- matrix* состоящий из x, y и 1 и *vector* из z. При помощи функции *np.linalg.matrix_rank(matrix)* было проверено, что ранг матрицы коэффициентов имеет допустимое для нахождения решения значение. Если условие *if np.linalg.matrix_rank(matrix) >= 3* не выполнялось, то функция возвращала *None*, иначе с помощью функции *np.linalg.solve(matrix,vector)* были получены искомые коэффициенты записанные в переменную *surface*. С помощью функции *np.round()* коэффициенты в этой переменной были округлены до 2 знаков после запятой.

Была реализована функция *def check_rotation(vec, rad)*, принимающая на вход *ndarray* 3-х координат дакибота *vec* и угол поворота *rad*. Была составлена матрица поворота в трёхмерном пространстве по оси *z*: *rotation= np.array([[np.cos(rad), -1 * np.sin(rad)], [np.sin(rad), np.cos(rad)]])* для которой, с помощью функций *np.sin()* и *np.cos()* были вычислены значения синуса и косинуса угла. Для получения повернутых координат необходимо умножить матрицу поворота на вектор координат дакибота. Была использованная функция

np.dot(), которая возвращает массив *ndarray* из произведений двух других массивов. Результат работы этой функции был записан в переменную ans. Повернутые координаты бы были округлены до 2 знаков после запятой с помощью *np.round()* и возвращены.

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. Были получены базовые навыки работы с библиотекой *numpy*. Были освоены функции округления, решения линейных систем уравнения, создания и умножения матриц, получения значений синуса и косинуса, получения ранга матрицы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Verdin_Kirill_lb1.py

```
import numpy
import numpy as np


def check_collision(bot1, bot2):
    a = np.array([[bot1[0], bot1[1]], [bot2[0], bot2[1]]])
    b = np.array([-bot1[2], -bot2[2]])
    if np.linalg.matrix_rank(a) >= 2:
        return tuple(np.round(numpy.linalg.solve(a, b), 2))
    return "None"


def check_surface(point1, point2, point3):
    matrix = np.array([[point1[0], point1[1], 1], [point2[0], point2[1],
1], [point3[0], point3[1], 1]])
    vector = np.array([point1[2], point2[2], point3[2]])
    if np.linalg.matrix_rank(matrix) >= 3:
        surface = np.linalg.solve(matrix, vector)
        return np.round(surface, 2)
    return "None"


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