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

Студентка гр. 2381 _____ Фомина А.К.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2022

Цель работы.

Изучить основные управляющие конструкции языка Python, а также модуль numpy, в частности пакет numpy.linalg.

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.

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.

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.

Выполнение работы.

Подключение модуля *numpy* с псевдонимом *np*. Также была создана вспомогательная функция **result_round**, в которой происходит округление результатов с помощью метода *np.around()* до двух знаков после запятой.

Функция check_collision

На вход объявляются два аргумента (*bot1*, *bot2*) типа *ndarray*, по условию принимающие коэффициенты уравнений прямых. Можно заметить, что линейные уравнения можно преобразовать в систему линейных уравнений, создав при этом матрицу и вектор свободных коэффициентов. Поэтому с помощью *np.array* создаются *matrix_a,* коэффициенты которого равны bot1[0], bot1[1], bot2[0], bot2[1], и *vector_b*, коэффициенты которого равны bot1[2], bot2[2] соответственно. Проверяется ранг матрицы и, если он будет не равен 2, то функция вернет *None*. Если ранг будет равен 2, то в созданной переменой *array* с помощью метода *np.linalg.solve()* находится решение линейной системы уравнений. В переменную result, которую возвращает функция, с помощью встроенной функции tuple(), которая меняет тип ndarray на кортеж, записывается полученный результат.

Функция check_surface

На вход объявляются три аргумента (*point1, point2, point3*), по условию принимающие координаты точек. Создаётся *matrix_a* с помощью *np.array()*. Получается матрица вида [[point1[0], point1[1], 1],[point2[0], point2[1], 1],

[point3[0], point3[1], 1]]. Также создается *vector_b*. Заполняется он также с помощью *np.array()*. Проходит проверка на ранг матрицы, которая в случае, если ранг не будет равен трём, возвращает *None*. В противном случае высчитывается результат с помощью метода *np.linalg.solve()* и записывается в переменную *result*, которая округляется с помощью метода *np.around* до двух знаков после запятой. Функция возвращается полученный результат.

Функция check_rotation

На вход объявляются две переменные (*vec, rad*), которым присваиваются значения координат и угол поворота. Согласно с матрицами поворота в трёхмерном пространстве, создается *matrix*, которая заполняется с помощью метода *np.array()*. С помощью метода *dot()* матрица умножается на переменную типа *ndarray*. Результат записывается в переменную result и округляется с помощью метода *np.arround()*. Функция возвращает полученный результат.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в тбл.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)	Верный ответ
2.	array([1, -6, 1]), array([0, -3, 2]), array([-3, 0, -1])	[2. 1. 5.]	Верный ответ
3.	array([-3 5 9]), array([-6 -6 6])	(1.75, -0.75)	Верный ответ

4.	array([1 -2 3]), array([2 -3 4]), array([3 -4 5])	None	Верный ответ, так как ранг матрицы равен 2
5.	array([1, -2, 3]), 1.57	[2. 1. 3.]	Верный ответ

Вывод.

Был изучены основные конструкции языка, модуль *numpy*, метод *numpy.linalg*. Была написана программа, состоящая из трёх функций, для каждой задачи, и одной вспомогательной функции, для округления полученного результата. С помощью основной конструкции *if else* были рассмотрены случаи, когда невозможно найти решения, из-за чего функция возвращала *None*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: index_first_even.h

```
import numpy as np

def result_round(arr):
    result_rounded = np.around(arr, decimals = 2)
    return result_rounded

def check_collision(bot1, bot2):
    matrix_a = np.array([[bot1[0], bot1[1]], [bot2[0], bot2[1]]])
    vector_b = np.array([-bot1[2], -bot2[2]])
    if np.linalg.matrix_rank(matrix_a) != 2:
        return None
    else:
        array = np.linalg.solve(matrix_a, vector_b)
        result = tuple(result_round(array))
        return result

def check_surface(point1, point2, point3):
    matrix_a  =  np.array([[point1[0],  point1[1],  1],[point2[0],
point2[1], 1], [point3[0], point3[1], 1]])
    vector_b = np.array([point1[2], point2[2], point3[2]])
    if np.linalg.matrix_rank(matrix_a) != 3:
        return None
    else:
        result = np.linalg.solve(matrix_a, vector_b)
        result = result_round(result)
        return result

def check_rotation(vec, rad):
    matrix    =    np.array([[np.cos(rad),    -np.sin(rad),    0],
[np.sin(rad), np.cos(rad), 0], [0, 0, 1]])
    result = matrix.dot(vec)
    result = result_round(result)
    return result
```