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 _____ Пачев Д.К.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы.

Написать программу на языке Python, выделив каждую задачу в отдельную функцию, а также освоить на практических задачах модуль numpy.

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 часть 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 часть 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)*.

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 часть 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)*.

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 часть 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.

- Функция `check_collision(bot1, bot2)` принимает на вход два аргумента типа ndarray - коэффициенты *bot1*, *bot2* уравнений прямых *bot1* = (*a1, b1, c1*), bot2 = (*a2, b2, c2*). Переменная `koef_matrix` представляет собой матрицу коэффициентов уравнений, а переменная `free_member_matrix` представляет собой матрицу свободных членов уравнений. Далее делается проверка но то, имеет ли система уравнений решения, эта проверка производится при помощи метода `numpy.linalg.matrix_rank()` Если ранг

матрицы меньше 2, то система не будет иметь решений и функция возвращает `None.` Иначе система уравнений будет иметь решение, и функция возвращает кортеж округленных до 2 знаков после запятой значений, который формируется с помощью генератора:

```
return tuple(round(value,2)for value in
list(np.linalg.solve(koef_matrix,free_member_matrix)))
```

- Функция `check_surface(point1, point2, point3)` принимает на вход координаты трех точек в виде *ndarray*. Так же, как в предыдущей функции в переменных `koef_matrix` и `free_member_matrix` хранятся значения, соответственно, коэффициентов уравнений плоскостей и свободных членов. Производится проверка на наличие решений системы уравнений с помощью `numpy.linalg.matrix_rank()` Если ранг матрицы меньше 3, то система не имеет решений, и функция возвращает `None.` Иначе функция возвращает массив *ndarray* следующим образом:

```
return np.array(list(round(value,2)for value in
list(np.linalg.solve(koef_matrix,free_member_matrix))))
```

- Функция `check_rotation(vec, rad)` принимает на вход *ndarray* 3-х координат дакибота и угол поворота. В переменной `rotate_matrix` хранится матрица, которая понадобится для вычисления скалярного произведения. В переменной `multi_matrix` хранится значение скалярного произведения `rotate_matrix` и `vec`. Возвращает функция *ndarray* с округленными до 2 знаков после запятой значениями, который формируется с помощью генератора:

```
return np.array(list(round(value,2) for value in
multi_matrix))
```

Тестирование.

Результаты тестирования представлены в Таблице 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)	Верно
2.	check_surface(np.array([1, -6, 1]), np.array([0, -3, 2]), np.array([-3, 0, -1]))	[2. 1. 5.]	Верно
3.	check_rotation(np.array([1, -2, 3]), 1.57)	[2. 1. 3.]	Верно

Выводы.

Была разработана программа на языке Python, где были использованы основные управляющие конструкции языка, а также развиты навыки по работе с модулем *numpy*, были изучены методы нахождения ранга матрицы, умножения матриц, использования функции округления и др.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np
from math import cos, sin


def check_collision(bot1, bot2):
    koef_matrix = np.array([[bot1[0], bot1[1]], [bot2[0], bot2[1]]])
    free_member_matrix = np.array([-bot1[2], -bot2[2]])
    if np.linalg.matrix_rank(koef_matrix) < 2:
        return None
    return tuple(round(value, 2) for value in
list(np.linalg.solve(koef_matrix, free_member_matrix)))


def check_surface(point1, point2, point3):
    koef_matrix = np.array([[point1[0], point1[1], 1],[point2[0],
point2[1], 1],[point3[0], point3[1], 1],])
    free_member_matrix = np.array([point1[2], point2[2], point3[2]])
    if np.linalg.matrix_rank(koef_matrix) < 3:
        return None
    return np.array(list(round(value, 2) for value in
list(np.linalg.solve(koef_matrix, free_member_matrix))))


def check_rotation(vec, rad):
    rotate_matrix = np.array([[cos(rad), -sin(rad), 0], [sin(rad),
cos(rad), 0], [0, 0, 1]])
    multi_matrix = np.dot(rotate_matrix, vec)
    return np.array(list(round(value, 2) for value in multi_matrix))
```