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 _____ Мурдасов М.К.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Изучение и работа с управляющими конструкциями языка Python.

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. 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. 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. 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. 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. 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. 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 *check_collision(),* принимающая на вход два ndarray - коэффициенты *bot1*, *bot2* уравнений прямых *bot1* = (*a1, b1, c1*), *bot2 = (a2, b2, c2)* (уравнение прямой имеет вид *ax+by+c=0*). В переменную *equation*, с помощью функции *np.array([[bot1[0], bot1[1]],[bot2[0], bot2[1]]]),* была записана матрица коэффициентов. В переменную *solution*, с помощью функции *np.array([-bot1[2], -bot2[2]])*, была записана матрица свободных членов. Далее, чтобы удостовериться, что ранг матрицы позволяет получить решение системы уравнений, с помощью условного оператора *if* проверяется условие *np.linalg.matrix_rank(equation) >= 2*. Если ранг матрицы удовлетворяет условию, то функция возвращает точку пересечения траекторий в виде кортежа из 2 значений, полученных с помощью функции *np.linalg.solve()* и заранее округленных до 2 знаков после запятой с помощью функции *round()* (*return tuple([round(x,2) for x in np.linalg.solve(equation, solution)])*), иначе возвращает *None*.

Была реализована функция *check_surface()*, принимающая на вход координаты 3 точек (3 *ndarray* 1 на 3): *point1*, *point2*, *point3*. В переменную *equation*, с помощью функции *np.array([[point1[0], point1[1], 1], [point2[0], point2[1], 1], [point3[0], point3[1], 1]]),* была записана матрица коэффициентов. В переменную *solution*, с помощью функции *np.array([point1[2], point2[2], point3[2]])*, была записана матрица свободных членов. Далее, чтобы удостовериться, что ранг матрицы позволяет получить решение системы уравнений, с помощью условного оператора *if* проверяется условие *np.linalg.matrix_rank(equation) >= 3*. Если ранг матрицы удовлетворяет условию, то функция возвращает коэффициенты a, b, c в виде *ndarray*, полученного с помощью функции *np.linalg.solve()*, для уравнения плоскости вида *ax+by+c=z*, предварительно округлив каждый элемент в *ndarray* с помощью функции *np.round()* (*return np.round(np.linalg.solve(equation, solution),2)*), иначе возвращает *None*.

Была реализована функция *check_rotation()*, принимающая на вход *ndarray* 3-х координат дакибота и угол поворота. В переменную *turn*, с помощью функции *np.array([[np.cos(rad), -(np.sin(rad)), 0], [np.sin(rad), np.cos(rad), 0], [0, 0, 1]])*, была записана матрица поворота в трехмерном пространстве. В переменную *map_point*, с помощью функции *np.array([vec[0], vec[1], vec[2]])*, была записана матрица, содержащая координаты дакибота. Далее, с помощью функции *turn.dot(map_point)*, матрица поворота была умножена на матрицу координат и с помощью функции *np.array([round(x,2) for x in turn.dot(map_point)])* была получена и возвращена функцией матрица с повернутыми *ndarray* координатами, каждая из которых округлена до 2 знаков после запятой с помощью функции *round()*.

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)	-
2.	array([1, -6, 1]), array([0, -3, 2]), array([-3, 0, -1])	[2. 1. 5.]	-
3.	array([1, -2, 3]), 1.57	[2. 1. 3.]	-

Выводы

Были освоены управляющие конструкции языка Python. Была изучена небольшая часть библиотеки numpy и таких ее функций, как умножение матриц, создание матриц, округление, решение линейных систем уравнений, вычисление ранга матрицы и поиск sin и cos углов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Murdasov_Mikhail_lb1.py

```
import numpy as np


def check_collision(bot1, bot2):
    equation = np.array([[bot1[0], bot1[1]],[bot2[0], bot2[1]]])
    solution = np.array([-bot1[2], -bot2[2]])
    if np.linalg.matrix_rank(equation) >= 2:
        return tuple([round(x,2) for x in np.linalg.solve(equation,
solution)])
    else:
        return None


def check_surface(point1, point2, point3):
      equation = np.array([[point1[0], point1[1], 1], [point2[0],
point2[1], 1], [point3[0], point3[1], 1]])
    solution = np.array([point1[2], point2[2], point3[2]])
    if np.linalg.matrix_rank(equation) >= 3:
        return np.round(np.linalg.solve(equation, solution),2)
    else:
        return None


def check_rotation(vec, rad):
          turn = np.array([[np.cos(rad), -(np.sin(rad)), 0],
[np.sin(rad), np.cos(rad), 0], [0, 0, 1]])
    map_point = np.array([vec[0], vec[1], vec[2]])
    return np.array([round(x,2) for x in turn.dot(map_point)])
```