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(ка) гр. 3343 _____ Гельман П.Е.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Изучить основные управляющие конструкции языка Python, библиотеку NumPy, а также научиться их применять для разработки программ.

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.

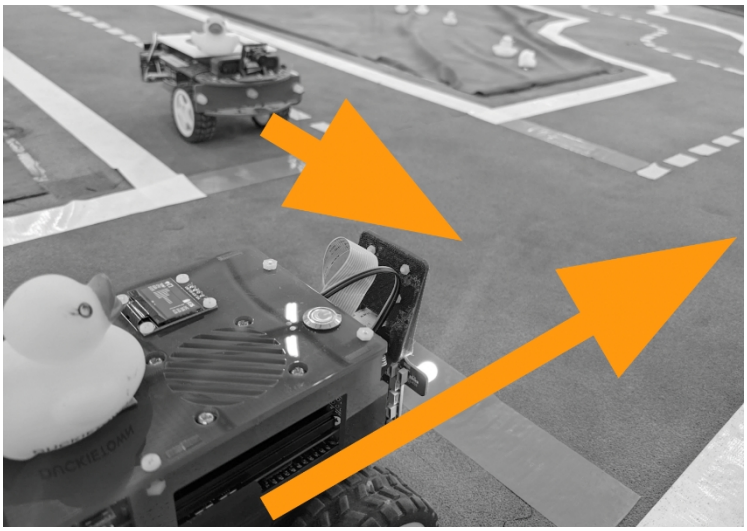

Рисунок 1 – Пример задачи 1

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.

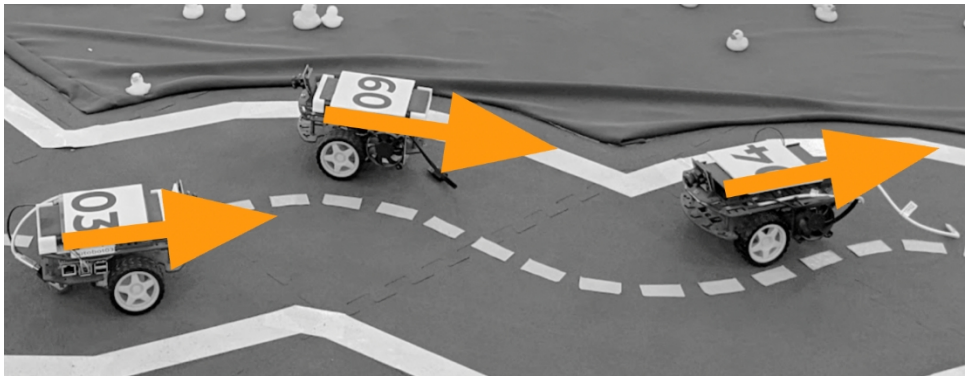

Рисунок 2 – Пример задачи 2

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.

Рисунок 3 – Пример задачи 3

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

Для решения поставленных задач была написана программа на языке Python, в которой реализуются три функции.

Решение первой задачи представлено в функции check_collision. На вход подаются коэффициенты траекторий двух дакиботов в виде ndarray, функция должна возвращать точку пересечения траекторий дакиботов (кортеж). Если у системы уравнений нет решения, то функция возвращает ***None***.

Вторая задача реализована в функции check_surface, на вход которой подаются ndarray с числами – координаты 3 точек. На выходе пользователь получает коэффициенты уравнения плоскости, в которой двигались дакиботы. Если у системы уравнений нет решения, то функция возвращает ***None***.

Для решения третьей задачи представлена функция check_rotation, на вход подаются ndarray с 3 координатами дакибота и угол поворота в радианах. Функция возвращает повернутые ndarray координаты дакибота.

В программе представлены следующие функции модуля NumPy:

1. *np.array()* - функция, которая создает объект типа nparray (массив/матрицу) из поданных на вход данных.
2. *np.linalg.det()* – функция, которая находит определитель заданной матрицы. В коде она необходима, чтобы установить, существуют ли решения у системы уравнений или нет.
3. *np.linalg.solve()* – функция, которая решает систему линейных уравнений, представленных в матричной форме. На вход подается матрица коэффициентов и вектор свободных членов.
4. *np.round()* – функция, округляющая все элементы объекта nparray.
5. *np.matmul()* – функция, которая перемножает два объекта ndarray. Эта функция позволяет умножать массивы разных размеров и форм, если их размерности совместимы.

Переменные, используемые в коде :

1. *bot1, bot2* – переменных, хранящие массивы с коэффициентами прямых в функции *check_collision.*

2. *A* – матрица коэффициентов при переменных (объект ndarray) в функции *check_collision.*
3. *B* - матрица-столбец свободных членов (объект ndarray) в функции *check_collision.*
4. *intersection_point* – переменная, хранящая координаты точки пересечения траекторий двух дакиботов в функции *check_collision.*

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	check_collision input: [-5 -10 0] [-10 -5 2]	(0.27, -0.13)	Проверка работы функции для задачи 1
2.	check_surface input: [1 -6 1] [0 -3 2] [-3 0 -1]	[2. 1. 5.]	Проверка работы функции для задачи 2
3. П	check_rotation input: [2 -2 -1] 1.4	None	Проверка работы функции для задачи 3
4.	check_collision input: [7 -9 7] [-9 5 5] check_surface input: [1 -3 -1] [-2 7 2] [3 2 -4] check_rotation input: [5 -6 1] 1.13	(1.74, 2.13) [-1.29 -0.09 0.03] [7.56 1.96 1.]	Проверка всех функций для трех задач

Выводы

В ходе лабораторной работы были изучены основные управляющие конструкции языка Python, а также модуль NumPy со всеми его функциями и методами.

Была разработана программа, включающая три функции, предназначенные для решения трех различных задач. Первая функция позволяет найти точку пересечения траекторий двух дакиботов. Вторая функция вычисляет коэффициенты уравнения плоскости, в которой двигались дакиботы. А третья функция помогает дакиботу вернуться в необходимое направление для продолжения движения.

Для успешного решения математических задач, связанных с матрицами, мы воспользовались библиотекой NumPy, что позволило эффективно работать с линейной алгеброй в контексте наших задач.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np
from math import *
def check_collision(bot1, bot2):
    # формируем систему линейных уравнений в виде Ax = B,
     # где A - матрица коэфф. при переменных, x - матрица-столбец переменных,
    # B - матрица-столбец свободных членов
    A = np.array([[bot1[0], bot1[1]], [bot2[0], bot2[1]]])
    B = np.array([-bot1[2], -bot2[2]])
     if np.linalg.det(A) == 0.0: # если определитель матрицы равен нулю, то есть линейная зависимость => у системы нет решений
        return None
    else:
                                            intersection_point     = tuple(np.round(np.linalg.solve(A, B),2))
        return intersection_point



def check_surface(point1, point2, point3):
    x1, y1, z1 = point1
    x2, y2, z2 = point2
    x3, y3, z3 = point3
     matrix_c = np.array([[x1, y1, 1], [x2, y2, 1], [x3, y3, 1]]) # матрица коэффициентов
     free_vector = np.array([z1, z2, z3]) # вектор свободных членов

    if np.linalg.det(matrix_c) == 0.0:
        return None
    else:
                  answer  =  np.round(np.linalg.solve(matrix_c, free_vector),2)

        return answer


def check_rotation(vec, rad):
      matrix_rotation = np.array([[cos(rad), sin(rad), 0], [-sin(rad), cos(rad), 0], [0, 0, 1]]) # создаем матрицу поворота вокруг оси z
      result = np.round(np.matmul(vec, matrix_rotation),2) # матричное умножение
    return result
```