МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Информатике»

Тема: Основные управляющие конструкции языка Python

Студент гр. 3384 _____ Рудаков А.Л.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы.

Изучить основные управляющие конструкции языка Python. Применить их на практике реализовав программный код.

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.

Пример ситуации:

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)*.

Пример входных данных:

array([-3, -6, 9]), array([8, -7, 0])

Пример возвращаемого результата:

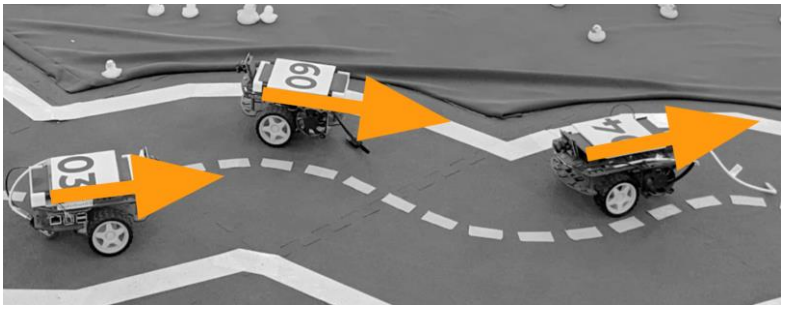
(0.91, 1.04)

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.

Пример ситуации:



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)*.

Например, даны точки: A(1, -6, 1); B(0, -3, 2); C(-3, 0, -1). Подставим их в уравнение плоскости:

$$a \cdot 1 + b(-6) + c = 1$$
$$a \cdot 0 + b(-3) + c = 2$$
$$a(-3) + b \cdot 0 + c = -1$$

Составим матрицу коэффициентов:

$$\begin{pmatrix} 1 & -6 & 1 \\ 0 & -3 & 1 \\ -3 & 0 & 1 \end{pmatrix}$$

Вектор свободных членов:

$$\begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$$

Для такой системы уравнение плоскости имеет вид: *z = 2x + 1y + 5*

Пример входных данных:

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

Возвращаемый результат:

[2. 1. 5.]

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
<Ссылка на википедию с матрицами поворота>

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)*..

Пример входных аргументов:

array([1, -2, 3]), 1.57

Пример возвращаемого результата:

[2. 1. 3.]

Выполнение работы.

В начале программы была подключена библиотека *numpy*, требующаяся для математических вычислений. Она была подключена как *np* — удобное сокращение.

Далее была написана функция *def check_collision(bot1, bot2)*, решающая первую подзадачу основной задачи. Она принимает 2 аргумента типа *ndarray*. В следующих двух строках были инициализированы 2 листа *nbot1* и *nbot2*, имеющие значения *bot1* и *bot2* соответственно. Далее была создана матрица *matr=np.array([[nbot1[0],nbot1[1]],[nbot2[0],nbot2[1]]])* размером 2x2, хранящая в первой строке значения 0 и 1 элементов листа *nbot1* и 0 и 1 элементов листа *nbot2* во второй строке. Данная матрица была создана благодаря методу *np.array()*. Далее был создан вектор *ans=np.array([-nbot1[2],-nbot2[2]])*, хранящий в себе противоположные по знаку значения 2-ых элементов листов *nbot1* и *nbot2* соответственно. Следующим в теле функции идет условный оператор *if np.linalg.matrix_rank(matr)<2*, проверяющий ранг матрицы matr. Ранг матрицы узнается через метод *np.linalg.matrix_rank()*. Если ранг матрицы *matr* размером 2x2 меньше, чем 2, то выполняется действие внутри условного оператора, а именно выполняется возврат *None*, организованный благодаря оператору *return*. Если же условие неверно, то выполняется ветка *else*. Внутри оператора выполняется инициализация кортежа *answer=tuple(np.round(np.linalg.solve(matr,ans),2))*. Метод *np.linalg.solve(matr,ans)* (1) находит решение системы уравнения, построенного через матрицы и создает матрицу с решениями. Метод *np.round(*(1), *2)* округляет значения в матрице до 2 знаков после запятой. Метод *tuple()* преобразует полученную матрицу в кортеж. В следующей строке выполняется возврат кортежа *answer*, выполненный благодаря оператору *return*.

Следующей была создана функция *def check_surface(point1, point2, point3)*, решающая вторую подзадачу основной задачи. Она принимает на вход 3 аргумента типа *ndarray*. В следующих трех строках были инициализированы 3 листа *npoint1*, *npoint2* и *npoint3*, имеющие преобразованные значения *point1*, *point2* и *point3* соответственно. Далее была создана матрица *matr=np.array([[npoint1[0],npoint1[1],1],[npoint2[0],npoint2[1],1],[npoint3[0],np oint3[1],1]])* размером 3x2, хранящая в первой строке значения 0 и 1 элемента листа npoin1, во второй строке значения 0 и 1 элемента листа npoint2 и значения

0 и 1 элемента листа *npoint3* в третьей строке. Далее был инициализирован вектор *ans=np.array([npoint1[2],npoint2[2],npoint3[2]])*, хранящий в себе значения 2-ых элементов листов *npoin1*, *npoin2* и *npoint3* соответственно. Следующим идет условный оператор if *np.linalg.matrix_rank(matr)<3*, проверяющий ранг матрицы. Ранг матрицы узнается через метод *np.linalg.matrix_rank()*. Если ранг матрицы *matr* размером 3x2 меньше, чем 3, то выполняется действие внутри условного оператора, а именно выполняется возврат *None*, организованный благодаря оператору *return*. Если же условие неверно, то выполняется ветка *else*. Внутри оператора выполняется инициализация кортежа *answer=np.round(np.linalg.solve(matr,ans),2)*. Метод *np.linalg.solve(matr,ans)* (1) находит решение системы уравнения, построенного через матрицы и создает матрицу с решениями. Метод *np.round(*(1), *2)* округляет значения в матрице до 2 знаков после запятой. В следующей строке выполняется возврат *answer*, выполненный благодаря оператору *return*.

Последней бал написана функция *def check_rotation(vec, rad)*, решающая третью подзадачу основной задачи. Она принимает на вход 2 аргумента. Первый типа *ndarray*, второй типа *float*. В следующих двух строках были инициализированы переменные *c* и *s*, типа *float*, хранящие в себе значения тригонометрических функций *cos* и *sin*, полученных благодаря методам *np.cos()* и *np.sin()* соответственно. Далее была создана матрица поворота по оси z *round_z=np.array([[c,-s,0],[s,c,0],[0,0,1]])* типа *ndarray*, размером 3x3. В первой строке хранятся c, -s, 0, во второй s, c, 0, в третьей 0, 0, 1. Следующей была создана повернутая матрица *answer=np.round(round_z.dot(vec),2)*. Метод *round_z.dot(vec)* (1) отвечает за умножение полученной в функции матрицы на соданную матрицу поворота. Метод *np.round(*(1),*2)* отвечает за округление всех значений в матрице до двух знаков после запятой. В следующей строке происходит возврат матрицы *answer* благодаря оператору *return*.

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

Выводы.

Были изучены основные управляющие конструкции языка Python. Был создан программный код решающий поставленную задачу.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main_lb1.py

```
import numpy as np


def check_collision(bot1, bot2):
    nbot1=list(bot1)
    nbot2=list(bot2)
    matr=np.array([[nbot1[0],nbot1[1]],[nbot2[0],nbot2[1]]])
    ans=np.array([-nbot1[2],-nbot2[2]])
    if np.linalg.matrix_rank(matr)<2:
        return None
    else:
        answer=tuple(np.round(np.linalg.solve(matr,ans),2))
        return answer


def check_surface(point1, point2, point3):
    npoint1=list(point1)
    npoint2=list(point2)
    npoint3=list(point3)

matr=np.array([[npoint1[0],npoint1[1],1],[npoint2[0],npoint2[1],1],[npoin
t3[0],npoint3[1],1]])
    ans=np.array([npoint1[2],npoint2[2],npoint3[2]])
    if np.linalg.matrix_rank(matr)<3:
        return None
    else:
        answer=np.round(np.linalg.solve(matr,ans),2)
        return answer


def check_rotation(vec, rad):
    c=np.cos(rad)
    s=np.sin(rad)
    round_z=np.array([[c,-s,0],[s,c,0],[0,0,1]])
    answer=np.round(round_z.dot(vec),2)
    return answer
```