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. 3342 _____ Антипина В.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Изучить основные управляющие конструкции языка Python, использование модуля NumPy.

Задание

Вариант 2

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

Дакибот приближается к перекрестку. Он знает 4 координаты, соответствующие координатам углов перекрестка (координаты образуют прямоугольник), и свои координаты. По правилам движения дакибот должен остановиться сразу, как только оказывается на перекрестке. Ваша задача -- помочь дакиботу понять, находится ли он на перекрестке (внутри прямоугольника).

Формальная постановка задачи

Оформите задачу как отдельную функцию: def check_crossroad(robot, point1, point2, point3, point4). На вход функции подаются: координаты дакибота robot и координаты точек, описывающих перекресток: point1, point2, point3, point4. Точка -- это кортеж из двух целых чисел (x, y). Функция должна возвращать True, если дакибот на перекрестке, и False, если дакибот вне перекрестка.

Примеры входных аргументов и результатов работы функции:

1. Входные аргументы: (9, 3) (14, 13) (26, 13) (26, 23) (14, 23)

Результат: False

2. Входные аргументы: (5, 8) (0, 3) (12, 3) (12, 16) (0, 16)

Результат: True

Задача 2. Содержательная часть задачи

Несколько дакиботов прибыли на базу, но их корпуса оказались поврежденными. В логах ботов программисты нашли сведения про их траектории движения, которые задаются линейными уравнениями вида: ax+by+c=0. В логах хранятся коэффициенты этих уравнений a, b, c.

Ваша задача -- вывести список номеров ботов (кортежи), которые столкнулись с друг другом (боты нумеруются с нуля, порядок следования коэффициентов уравнений соответствует порядку ботов).

Формальная постановка задачи

Оформите решение в виде отдельной функции check_collision(). На вход функции подается матрица ndarray Nx3 (N -- количество ботов, может быть разным в разных тестах) коэффициентов уравнений траекторий coefficients. Функция возвращает список пар -- номера столкнувшихся ботов (если никто из ботов не столкнулся, возвращается пустой список).

Пример входного аргумента ndarray 4x3 :

```
[[-1 -4  0]
 [-7 -5  5]
 [ 1  4  2]
 [-5  2  2]]
```

Пример выходных данных:

[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]

Первая пара в этом списке (0, 1) означает, что столкнулись 0-й и 1-й боты (то есть их траектории имеют общую точку).

В списке отсутствует пара (0, 2), можно сделать вывод, это боты 0-й и 2-й не сталкивались (их траектории НЕ имеют общей точки).

Примечание: помните про ранг матрицы и как от него зависит существование решения системы уравнений. В случае, если ни одного решение не было найдено (например, из-за линейно зависимых векторов), функция должна вернуть пустой список [].

Задача 3. Содержательная часть задачи

При перемещении по дакитауну дакибот должен регулярно отправлять на базу сведения, среди которых есть длина пройденного пути. Дакиботу известна последовательность своих координат (x, y), по которым он проехал. Ваша задача -- помочь дакиботу посчитать длину пути.

Формальная постановка задачи

Оформите задачу как отдельную функцию check_path, на вход которой передается последовательность (список) двумерных точек (пар) points_list. Функция должна возвращать число -- длину пройденного дакиботом пути (выполните округление до 2 знака с помощью round(value, 2)).

Пример входных данных:

[(1.0, 2.0), (2.0, 3.0)]

Пример выходных данных:

1.41

Пример входных данных:

[(2.0, 3.0), (4.0, 5.0)]

Пример выходных данных:

2.83

Основные теоретические положения

Функция округления вещественных чисел round(value[, signs]), которая принимает на вход 2 аргумента: вещественное число value, которое нужно округлить и номер знака после запятой signs, до которого нужно округлить (если этот аргумент отсутствует, то округление происходит до ближайшего целого).

Методы модуля numpy:

1) numpy.radians(array)

Данных метод позволяет перевести последовательность значений углов (в геометрическом смысле) array (типа ndarray) из градусов в радианы. Возвращает новый ndarray со значениями тех же углов в радианах.

2) numpy.ones(shape))

Данный метод позволяет создать матрицу из единиц заданного размера. Размер задается с помощью кортежа shape, где через запятую передаются размеры матрицы. Например, (2, 2) размерность говорит про 2-мерную матрицу, которая состоит из двух строк и 2-х столбцов.

Данный метод в общем случае может принимать больше аргументов.

3) numpy.vstack(arr1, arr2[, arrN])

Данный метод позволяет дописать матрицы последовательного друг к другу. При вертикальном "дописывании" матриц получится матрица с большим количество строк в результате.

Есть похожий метод numpy.hstack(arr1, arr2[, arrN]), который делает то же самое, но в горизонтальной ориентации (в результате увеличивается количество столбцов матрицы)

4) numpy.linalg.norm(vector)

Данный метод из пакета linalg модуля numpy позволяет вычислить норму (модуль, длину) вектора vector(в общем случае — матрицы), переданного на вход.

Длина вектора — это число, равное корню из суммы квадратов координат вектора.

5) numpy.linalg.matrix_rank(matrix)

Данный метод из пакета linalg позволяет посчитать ранг квадратной матрицы matrix.

Ранг — это количество линейно-независимых строк или столбцов квадратной матрицы.

Два вектора r1 и r2 считаются линейно-независимыми, если ни один из них нельзя представить в виде линейной комбинации другого.

Это значит, что нет таких чисел, которые можно прибавить, отнять или на которые можно умножить, разделить вектор r1, чтобы получился r2. Линейная комбинация как раз и означает сложение, вычитание с числом, умножение и деление на число.

6) numpy.linalg.solve(A, v)

Данный метод из пакета linalg позволяет найти решение линейной системы уравнений, которая представлена матрицей коэффициентов A и вектором свободных членов v.

7) ndarray.dot(ndarray)

Данный метод объекта ndarray позволяет выполнить умножение двух матриц друг на друга. Для того чтобы умножение было возможно, количество столбцов первой матрицы (та, у которой вызывается метод) должно быть равным количеству строк второй матрицы (та, которая передается как аргумент).

Выполнение работы

Был импортирован модуль NumPy и библиотека math (чтобы вычислить модуль). Была объявлена функция check_crossroad с аргументами robot, point1, point2, point3, point4. Далее происходило сравнение координат: если координата по оси абсцисс робота попадает в интервал, определённый точками (для каждой точки есть другая точка, с которой у них одни и те же координаты по оси абсцисс или ординат, поэтому все переменные point? можно не использовать). Аналогично, если по оси Y точка попадает в интервал, определённый координатами по этой оси, то при соблюдении двух этих условий, дакибот попадает на перекрёсток. Реализовано это с помощью условного оператора if и обращения по индексу к элементам кортежа.

Далее была написана функция check_collision(coefficient), которая получает на вход матрицу типа nparray. В теле функции были объявлены пустые массивы add и result и записано количество строк в массиве в переменную num_bots. В цикле for добавлялись нули в массиве add. Этот массив нужен для того, чтобы добавить к массиву с коэффициентами столбец с нулями (это, в свою очередь, нужно для того, чтобы потом разделить матрицу на две, в каждой из которых по два столбца (метод np.hsplit работает только с матрицами, которые можно разделить на равное число столбцов)), что и было реализовано далее. С помощью np.append был добавлен столбец, с помощью упомянутого ранее метода была разделена основная матрица, в first и second были записаны полученные матрицы, столбец с нулями был удалён из массива second. Далее с помощью вложенных циклов были составлены возможные пары ботов, а их коэффициенты записаны в массивы A и B. Если ранг матрицы A был равен 2, в массив с результатом добавлялся кортеж из индексов переменных. Так как сталкивался не только первый бот со вторым, например, но и второй с первым, нужно было настроить вывод и «перевернутых» кортежей. Поэтому в результат добавлялись кортежи и с перевернутыми индексами, на выходе сортировался массив.

Затем была написана функция check_path, которой на вход поступает список. Длина этого списка была записана в переменную num, сумматор приравнен к нулю. В цикле for до num-1 (так как мы используем i+1 элемент) к сумматору добавляется корень из квадратов разностей координат x и y для соседних элементов. Корень — функция abs из библиотеки math. Полученный результат округляется методом round().

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.	(5, 8) (0, 3) (12, 3) (12, 16) (0, 16)	True	Для первой функции
2.	[[-3,-7,1],[-3,-6,7],[1,4,2],[0,5,5]]	[(0, 1), (0, 2), (0, 3), (1, 0), (1, 2), (1, 3), (2, 0), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]	Для второй
3.	[(3.0,6.0),(9.1,5.5)]	6.12	Для третьей
...			

Выводы

Я изучила основные управляющие конструкции языка Python, использование модуля NumPy.

Были написаны функции, определяющие, попадают ли координаты в определённую область, столкнутся ли боты с заданными траекториями и какова длина пройденного ботом пути. Я использовала методы модуля numpy, операции с матрицами.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Antipina_Veronika_lb1.py

```
import numpy as np
import math


def check_crossroad(robot, point1, point2, point3, point4):
    if (point1[0] <= robot[0] <= point2[0]) and (point2[1] <=
robot[1] <= point3[1]):
        return True
    else:
        return False


def check_collision(coefficients):
    add = []
    result = []
    num_bots = len(coefficients)
    for i in range(num_bots):
        add.append([0])
    coefficients = np.append(coefficients, [*add], axis= 1 )
    ready = np.hsplit(coefficients,2)
    first = ready[0]
    second = ready[1]
    second = np.delete(second,1,1)

    for i in range(num_bots):
        for j in range(i+1,num_bots):
            A = np.array([first[i],first[j]])
            B = np.array([second[i],second[j]])
            if np.linalg.matrix_rank(A)==2:
                result.append((i,j))
                result.append((j,i))

    return sorted(result)


def check_path(points_list):
    num = len(points_list)
    summ = 0
    for i in range(num-1):
        summ+=math.sqrt((points_list[i+1][0]-
points_list[i][0])**2 + (points_list[i+1][1]-points_list[i][1])**2)
    summ = round(summ,2)
    return summ
```