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

Студентка гр. 3341 Чинаева М.Р.

Преподаватель Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Цель лабораторной работы состоит в решении задач, включающих использование модуля numpy и пакета numpy.linalg для работы с линейной алгеброй. В результате лабораторной работы необходимо разработать и протестировать 3 функции, каждая из которых решает свою задачу.

Задание

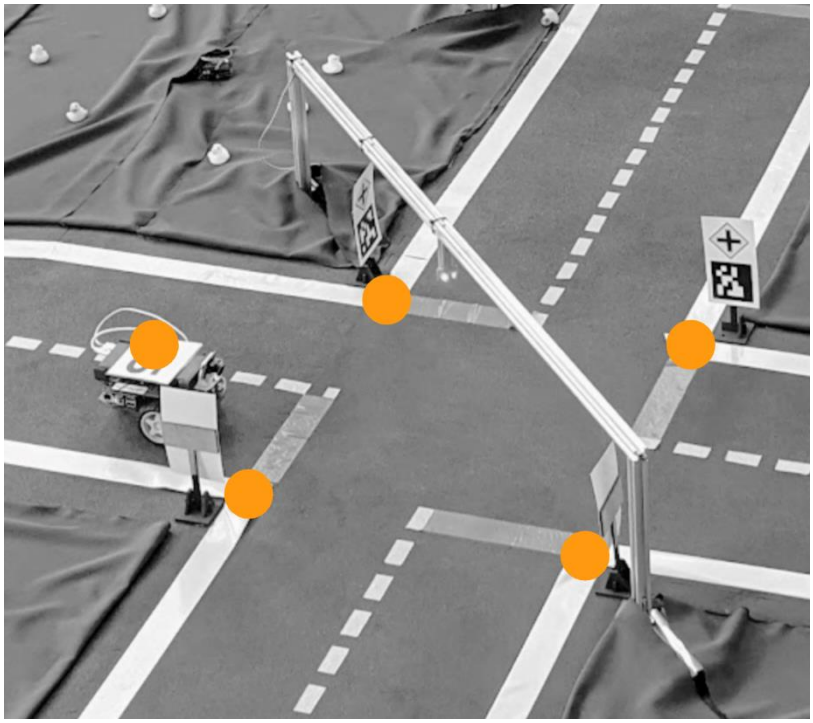
Вариант 2

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.

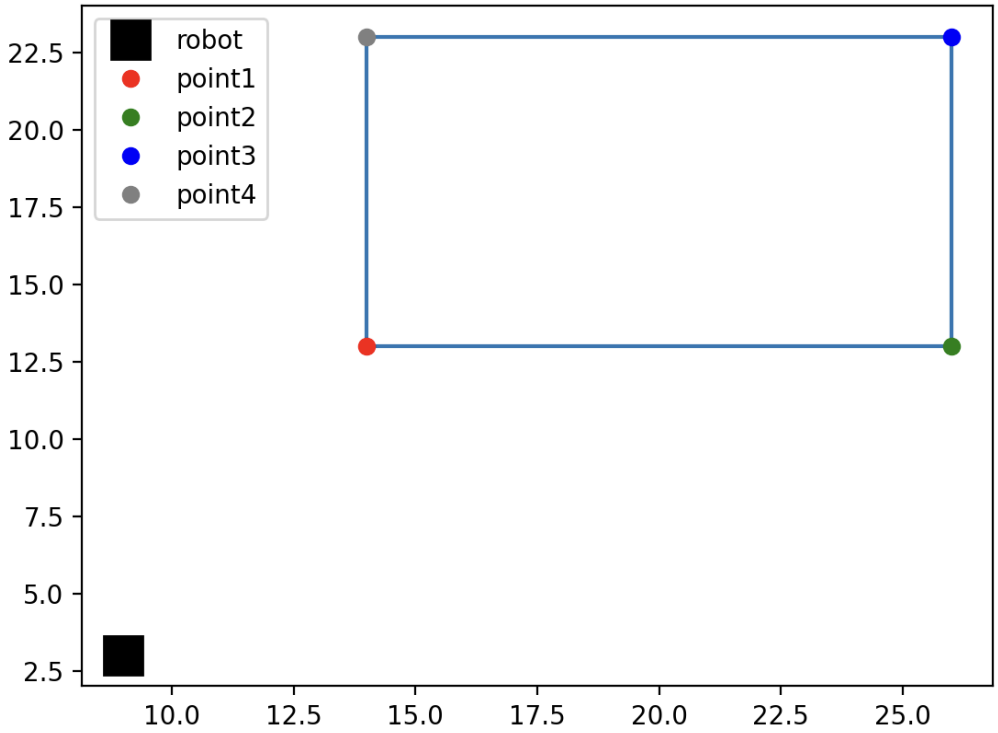
Задача 1. Содержательная постановка задачи

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).

Пример ситуации:



Геометрическое представление (вид сверху со схематичным обозначением объектов; перекресток ограничен прямыми линиями; обратите внимание, как пронумерованы точки):



Формальная постановка задачи

Оформите задачу как отдельную функцию: def check_rectangle(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)

Резлультат: False

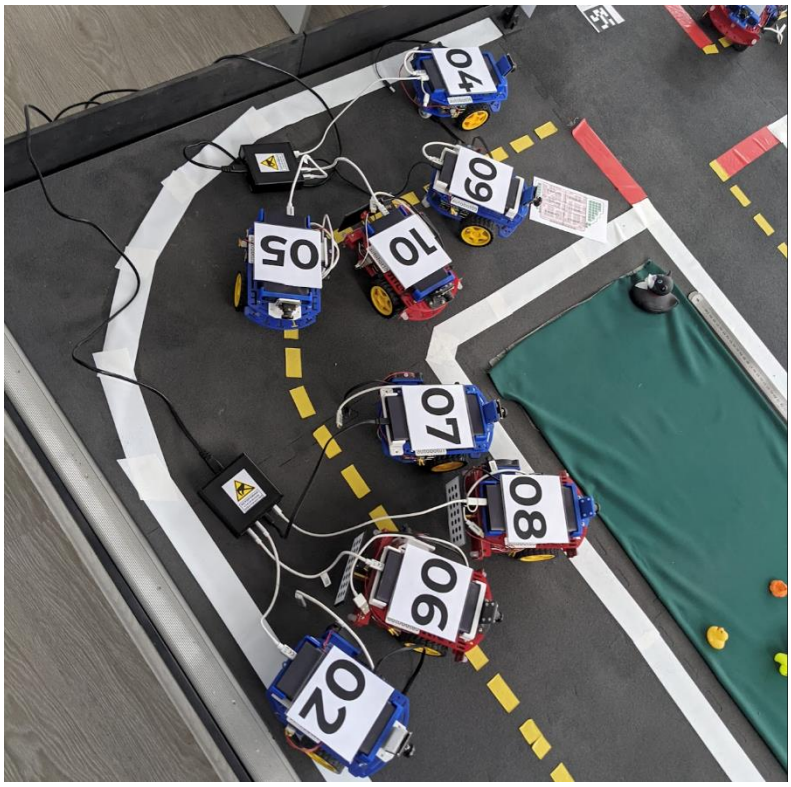
2. Входные аргументы: (5, 8) (0, 3) (12, 3) (12, 16) (0, 16)

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)]

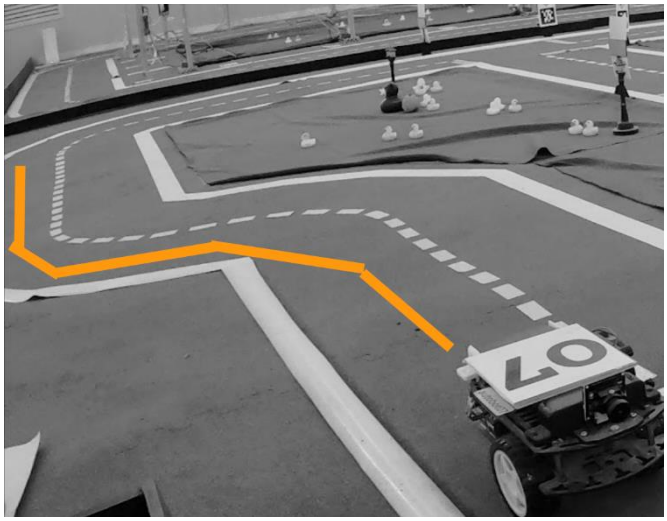
Первая пара в этом списке (0, 1) означает, что столкнулись 0-й и 1-й боты (то есть их траектории имеют общую точку).

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

Для решения поставленных задач была использована библиотека *NumPy* — это библиотека Python, которую применяют для математических вычислений и работы с массивами.

Использованные методы:

1. *np.array* (массив numpy, многомерный массив данных, над которыми можно быстро и эффективно выполнять множество математических, статистических, логических и других операций)
2. *np.vstack* (вертикальное размещение массива, объединяет по вертикали в матрицу входящие аргументы)
3. *np.delete* (удаление элементов из массива вдоль указанной оси)
4. *np.ndarray.shape* (кортеж измерений массива)
5. *np.linalg.matrix_rank* (определение ранга матрицы(макисмального количества линейно независимых строк))
6. *np.linalg.norm* (матричная или векторная норма. Используется для вычисления нормы вектора или матрицы)
7. *np.round* (округление до заданного числа десятичных знаков)

Выполнение работы

Импортируется библиотека *numpy*: *import numpy as np*

В задании требуется оформить каждую из 3 задач в виде отдельной функции согласно условиям.

1. Решение задачи 1:

   def check_crossroad(robot, point1, point2, point3, point4)

   Функция получает на вход координаты дакибота (*robot)* и координаты точек, описывающих перекресток: *point1*, *point2*, *point3*, *point4*. Точка – это кортеж из двух целых чисел *(x, y).*

   Гарантируется, что точки – координаты прямоугольника, *значит x1=x4, x2=x3, y1=y2, y3=y4,* из этого следует, что нам нужны только 2 точки, в данном случае использованы *point1* и *point3.* Сначала присваиваем переменным *x* и *y, x1* и *y1, x3* и *y*3 координаты из кортежей *robot, point1*, *point3* соответственно. Для того чтобы дакибот находился в заданном прямоугольнике его абсцисса (x) должна находиться на числовой прямой между *x1* и *x3*, а его ордината(y) между *y1* и *y3*. Это проверяет условный оператор *if x1<=x and x<=x3 and y1<=y and y<=y3*. Если все условия выполняются возвращается *True*, а иначе *False.*

2. Решение задачи 2:

   def check_collision(coefficients)

   На вход функции подается матрица *ndarray Nx3* (*N* -- количество ботов, может быть разным) коэффициентов уравнений траекторий *coefficients*. Создается пустой список *crash*, далее в него будут записаны пары столкнувшихся ботов. С помощью вложенных циклов *for* перебираем все сочетания строк. *coefficients.shape[0]* вычисляет количество столбцов – это наше ограничение по перебору строк матрицы. Перед началом выполнения тела цикла проверяем, чтобы *i* и *j* не совпадали с помощью условного оператора *if i != j*, дабы не проверять сочетание строчки самой с собой. С помощью функции *np.array* преобразовываем строки матрицы в массивы типа *numpy* и передаем их списком в качестве

аргумента для функции *np.vstack*, которая объединяет входящие аргументы в матрицу *matrix*. Далее с помощью функции *np.delete(matrix, 2, axis=1)* удаляем последний(второй) столбец матрицы, в котором хранятся свободные коэффициенты, и сохраняем новую матрицу под названием *check_matrix*. Это нужно для того чтобы точно узнать пересекаются ли траектории ботов. Чтобы траектории ботов пересеклись, т. е. система уравнений имела решения, и случилось столкновение матрица *check_matrix* должна иметь ранг 2, так как в заданном уравнении две неизвестные. Если бы столбец со свободными коэффициентами не был удален в случае попарного равенства коэффициентов *a* и *b* уравнения и неравенства с матрице был бы ошибочно присвоен ранг два, хотя данные траектории не пересекаются. Далее с помощью условного оператора *if np.linalg.matrix_rank(check_matrix)==2* проверяется ранг матрицы (функция *np.linalg.matrix_rank*) и если он равен двум, то в список *crash* добавляются кортежем номера ботов. Функция возвращает список пар – номера столкнувшихся ботов, если же никто из ботов не столкнулся, возвращается пустой список.

3. Решение задачи 3:

   def check_path(points_list)

   На вход передается список двумерных точек *points_list*. Создается переменная *distance* и приравнивается к нулю. С помощью цикла *for i in range(0, len(points_list)-1)* перебираем все последовательные сочетания положений дакибота (точка 1 и точка 2, точка 2 и точка3 и т.д.). Диапазон цикла ограничен выражением *len(points_list)-1*, так как в каждой итерации проверяется *i*ый и *(i+1)*й элемент и *(i+1)* не должно выходить за пределы списка. Далее с помощью функции *np.array* point1 и *point2* присваиваем массивы типа *numpy,* созданные из *i*го и *(i+1)*го кортежей списка. Вычитая из *point2 point1* вычисляем координаты радиус-вектора перемещения. С помощью функции np.linalg.norm вычисляем

длину(модуль) этого радиус вектора, это и будет дистанцией, преодоленной дакиботом из точки с номером i в точку (i+1). Далее прибавляем эту дистанцию (local_distance) к дистанции за все точки distance+=float(local_distance). Функция возвращает число – длину, округленную до 2 знака с помощью *round*, пройденного дакиботом пути.

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.	(1, 3) (10, 13) (15, 13) (15, 23) (10, 23)	False	Задача 1
2.	(5, 8) (1, 3) (20, 3) (20, 38) (1, 38)	True	Задача 1
3.	[[-1 -4 0] [-7 -5 5] [1 4 2]]	[(0, 1), (1, 0), (1, 2), (2, 1)]	Задача 2
4.	[(4.0, 2.0), (8.0, 3.0)]	4.12	Задача 3
5.	[(4.0, 5.0), (8.0, 9.0)]	5.66	Задача 3

Выводы

В результате выполнения данной лабораторной работы были достигнуты следующие цели: решение задач, включающих использование модуля numpy и пакета numpy.linalg для работы с линейной алгеброй, а также разработка и тестирование трех функций, каждая из которых предназначена для решения конкретной задачи.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np

def check_crossroad(robot, point1, point2, point3, point4):
    x, y=(robot)
    x1, y1=(point1)
    x3, y3=(point3)
    if x1<=x and x<=x3 and y1<=y and y<=y3:
        return True
    else: return False


def check_collision(coefficients):
    crash=[]
    for i in range(0, coefficients.shape[0]):
        for j in range(0, coefficients.shape[0]):
            if i != j:
                matrix=np.vstack([np.array(coefficients[i]),
np.array(coefficients[j])])
                check_matrix=np.delete(matrix, 2, axis=1)
                if np.linalg.matrix_rank(check_matrix)==2:
                    crash.append((i,j))
    return crash


def check_path(points_list):
    distance=0
    for i in range(0, len(points_list)-1):
        point1=np.array(points_list[i])
        point2=np.array(points_list[i+1])
        local_distance=np.linalg.norm(point2 - point1)
        distance+=float(local_distance)
    return round(distance, 2)
```