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. 3341 _____ Мальцев К.Л.

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

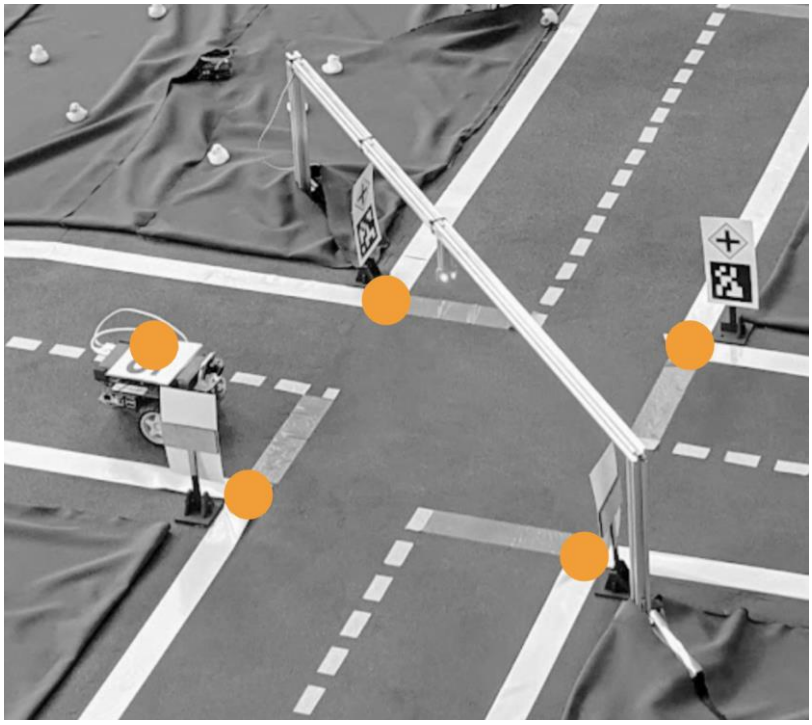
Вариант 2

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.

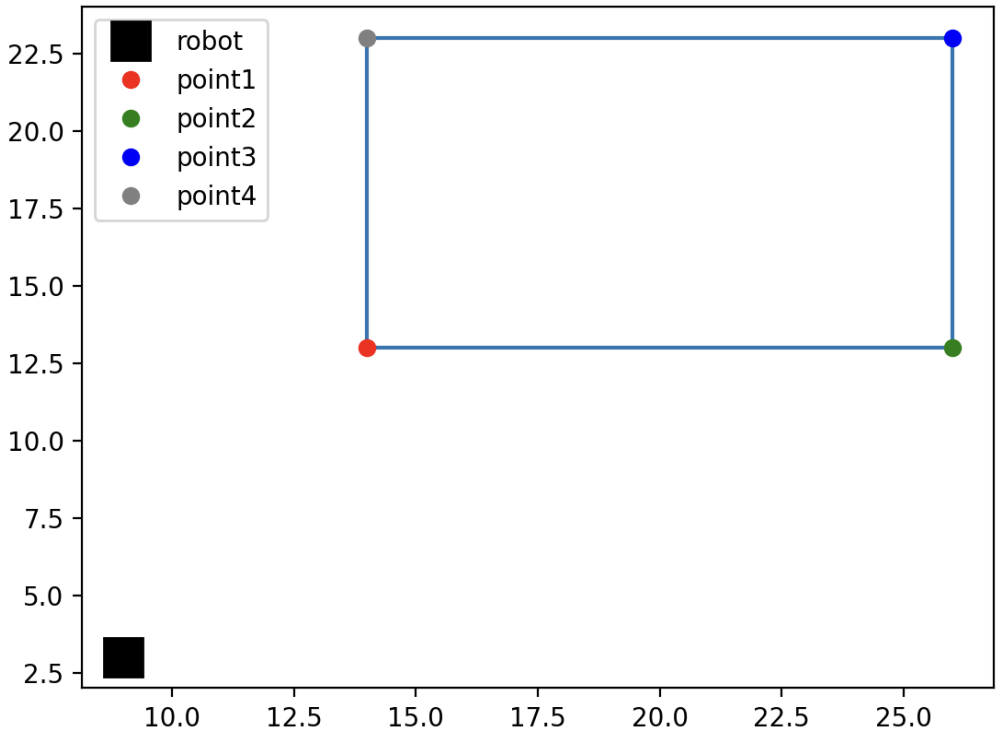
Задача 1. Содержательная постановка задачи

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

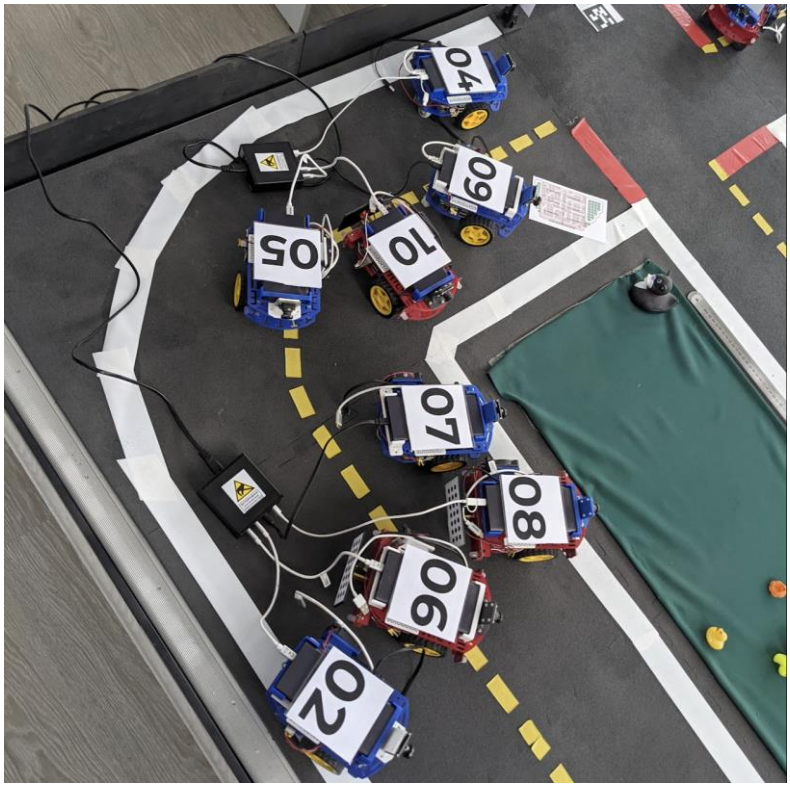
2. Входные аргументы: (5, 8) (0, 3) (12, 3) (12, 16) (0, 16)

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)]

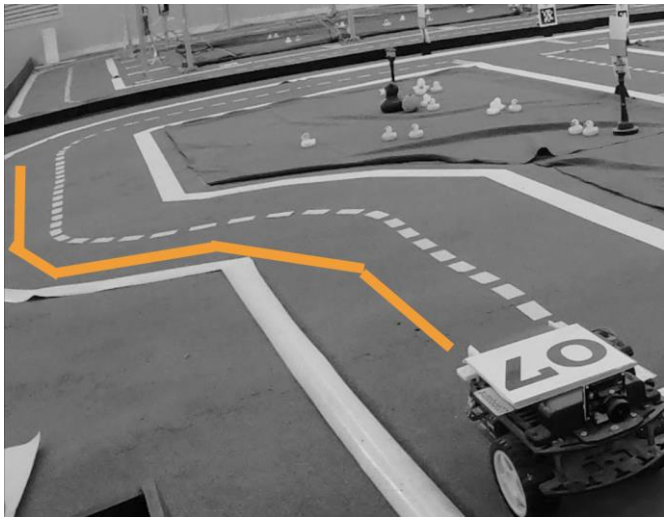
Первая пара в этом списке (0, 1) означает, что столкнулись 0-й и 1-й боты (то есть их траектории имеют общую точку).

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*. *NumPy* — это библиотека Python, которую применяют для математических вычислений: начиная с базовых функций и заканчивая линейной алгеброй. Для подключения библиотеки прописана строка *import numpy as np* (в программе для обращения к методам библиотеки используется следующая запись: *np.<название метода>)*

Использованные методы и атрибуты:

1. *np.array* (массив numpy, многомерный массив (*ndarray*, *n-dimensional array*) данных, над которыми можно быстро и эффективно выполнять множество математических, статистических, логических и других операций)
2. *np.ndarray.shape* (кортеж измерений массива)
3. *np.linalg* (функции линейной алгебры *numpy* для обеспечения эффективной низкоуровневой реализации стандартных алгоритмов линейной алгебры)
4. *np.linalg.norm* (матричная или векторная норма. Эта функция способна возвращать одну из восьми различных матричных норм или одну из бесконечного числа векторных норм (описанных ниже), в зависимости от значения параметра *ord*)
5. *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)

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

   Геометрическое представление (вид сверху со схематичным обозначением объектов; перекресток ограничен прямыми линиями) см. на рисунке 1:

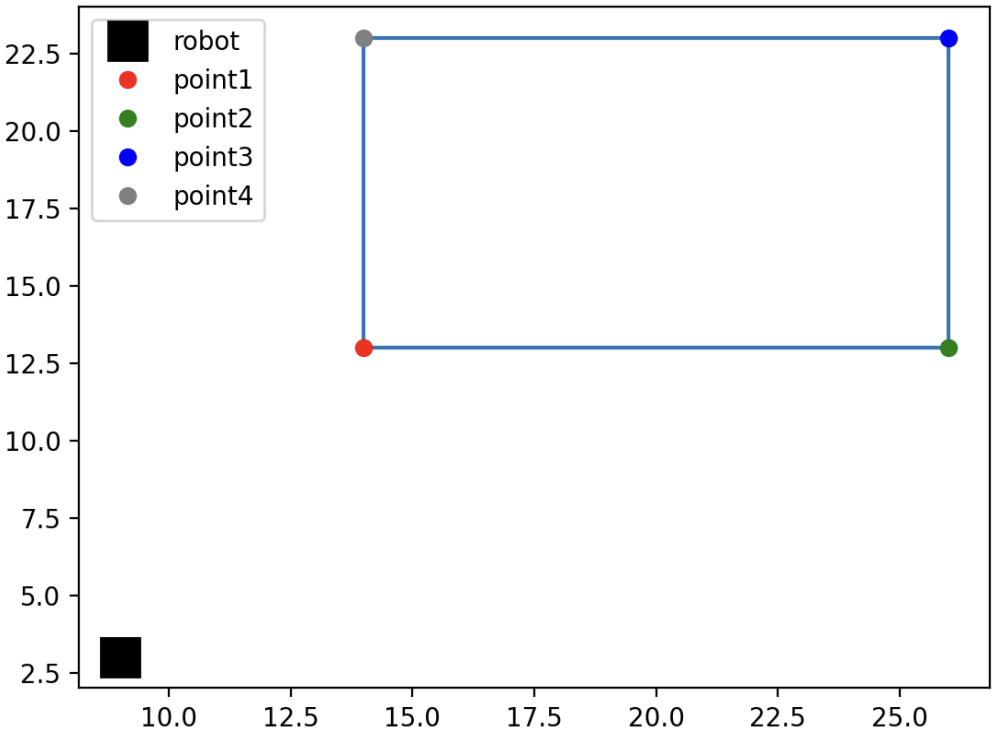


Рисунок 1 – Геометрическое представление

Т.к. гарантированно, что *point1*, *point2*, *point3*, *point4* – координаты прямоугольника, то достаточно проверить, что *robot(x, y)* находится правее $x=point1_x$, выше $y=point1_y$, левее $x=point4_x$, ниже $y=point4_y$ или лежит на данных прямых. Более формально:

R_{xy} *Є P1P2P3P4, если* $R_x \geq P1_x$ & $R_y \geq P1_y$ & $R_x \leq P4_x$ & $R_y \leq P4_y$. Аналогичную запись можно сделать и для *point2, point4.* Т.к. координаты точек передаются в виде кортежа *(x, y)*, то оси *x* будет соответствовать *[0]* индекс кортежа, а оси *y – [1]*. Создаются две переменные *f1* и *f2* типа *bool*, которые имеют значения:

(robot[0] >= point1[0]) and (robot[1] >= point1[1]) и

(robot[0] <= point3[0]) and (robot[1] <= point3[1]) соответственно. Функция возвращает *bool(f1 and f2)*, что равносильно условию описанному выше.

2. Решение задачи 2:

   def check_collision(coefficients)

   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).

   Создаётся пустой список *collisions*, в него будут записываться пары столкнувшихся ботов.

   Запускается цикл с ещё одним вложенным:

   for i in range(coefficients.shape[0]):

   for j in range(coefficients.shape[0]): ...

   Внутри цикла сравниваются попарно все элементы массива *coefficients*. *coefficients[i]* представляет собой функцию вида ax + by + c = 0, где a,b,c – передаваемые значения. Отсюда можно выразить y = -ax/b – c/b. Получаем уравнение вида y = ax + q. Следовательно, чтобы проверить столкнутся ли бот *i* и бот *j*, нужно сравнить k_i и k_j. Если они не равны, то траектории ботов пересекутся, в противном случае нужно сделать проверку q_i и q_j. Если они также равны, то траектории ботов совпадают. Получим условие столкновения: k_i != k_j | k_i == k_j & q_i == q_j. Та же запись, но с использованием коэффициентов a, b, c:

$a_i * b_i \neq a_i * b_i \mid a_i * b_i = a_i * b_i \ \& \ c_i = c_j$.

В теле вложенного цикла есть условие проверки сравнения траектории бота не с самим собой: *if (i != j): ...*. Если оно не выполняется, то такая пара *i* и *j* пропускается, т.к. в данном случае *i==j*. Если же условие выполнилось, то создаются 2 ссылки *Ri* и *Rj* на *coefficients[i]* и *coefficients[j]*. Это нужно для того чтобы не загружать код программы длинными названиями переменных. Далее следует условие: *if (Ri[0]*Rj[1] != Rj[0]*Ri[1] or Ri[0]*Rj[1] == Rj[0]*Ri[1] and Ri[2] == Rj[2]): ...*, которое соответствует описанному выше условию проверки пары ботов на столкновение. Если оно выполнилось, то пара *(i, j)* записывается в массив *collisions*: *collisions.append((i, j))*.

После того, как внешний цикл выполнится функция возвращает список *collisions*.

Примечание: данную задачу можно было выполнить также с помощью методов библиотеки *numpy*. При таком методе решения понадобилось бы подобрать 2 точки для каждой из функций вида

y = -ax/b – c/b. Из полученных точек составить координаты вектора $\{x_i, y_i\}$. Совершить аналогичное действие для бота j и получить: $\{x_j, y_j\}$. Составить матрицу вида: $[[x_i, y_i], [x_j, y_j]]$. Далее или с помощью *np.linalg.matrix_rank* проверить равен ли ранг матрицы 1, следовательно, траектории ботов не пересекутся, или с помощью *np.linalg.det* узнать равен ли определитель матрицы 0, тогда траектории ботов также не пересекутся. См рисунок 2:

```
>>> import numpy as np
>>> M = np.array( [[1, 1], [2, 2]] )
>>> np.linalg.matrix_rank(M)
1
>>> np.linalg.det(M)
0.0
>>>
```

Рисунок 2 – Метод решения через numpy

Метод решения через сравнение коэффициентов лучше, т.к. в нем полностью отсутствуют операции деления, которая могла бы повлиять на результат работы программы.

Решение задачи 3:

def check_path(points_list)

На вход передается последовательность (список) двумерных точек (пар) *points_list*. Функция должна возвращать число – длину пройденного дакиботом пути (с округлением до 2 знака с помощью *round(value, 2)*).

Массив *points_list* переводится в тип *ndarray* с типом данных *float*: *points_list = np.array(points_list, dtype=float)* (это нужно для дальнейшего удобства работы с *points_list*).

Создаётся переменная *sm*, равная 0, в которую будет записываться длина пройденного пути.

Т.к. *points_list[0]* – это начальные координаты дакибота, то для расчёта суммарной длины пути нужно посчитать сумму модулей длин векторов образованных парами координат *points_list[i]* и *points_list[i-1]*. Для этого удобно использовать цикл *for i in range(1, len(points_list)): …*. Цикл начинается с индекса 1, т.к. будут браться пары, стоящие на позициях *i* и *i-1*. В теле цикла рассчитывается модуль вектора образованного путём вычитания координат точки *i-1* из координат точки *i*: $(x, y)_i - (x, y)_{i-1}$. Модуль вектора рассчитывается с помощью метода *np.linalg.norm* и прибавляется к переменной *sm*:

sm += np.linalg.norm(points_list[i] - points_list[i-1])

После конца цикла функция возвращает значение sm, округленное до 2 знаков после запятой: *return np.round(sm, 2)*.

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.	(9, 3) (14, 13) (26, 13) (26, 23) (14, 23)	False	Задача 1
2.	(5, 8) (0, 3) (12, 3) (12, 16) (0, 16)	True	Задача 1
3.	[[-1 -4 0] [-7 -5 5] [1 4 2] [-5 2 2]]	[(0, 1), (0, 3), (1, 0), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1), (3, 2)]	Задача 2
4.	[(1.0, 2.0), (2.0, 3.0)]	1.41	Задача 3
5.	[(2.0, 3.0), (4.0, 5.0)]	2.83	Задача 3

Выводы

В результате выполнения данной лабораторной работы были достигнуты следующая цель: 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):
    f1 = (robot[0] >= point1[0]) and (robot[1] >= point1[1])
    f2 = (robot[0] <= point3[0]) and (robot[1] <= point3[1])
    return bool(f1 and f2)


def check_collision(coefficients):
    collisions = []
    for i in range(coefficients.shape[0]):
        for j in range(coefficients.shape[0]):
            if (i != j):
                Ri = coefficients[i]
                Rj = coefficients[j]
                if (Ri[0]*Rj[1] != Rj[0]*Ri[1] or
                    Ri[0]*Rj[1] == Rj[0]*Ri[1] and Ri[2] == Rj[2]):
                    collisions.append( (i, j) )
    return collisions


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