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 _____ Самокрутов А.Р.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Целью лабораторной работы является отработка навыков работы с основными управляющими конструкциями языка Python и библиотекой NumPy. В ходе лабораторной работы необходимо разработать и протестировать три функции, каждая из которых решает определённую задачу.

Задание

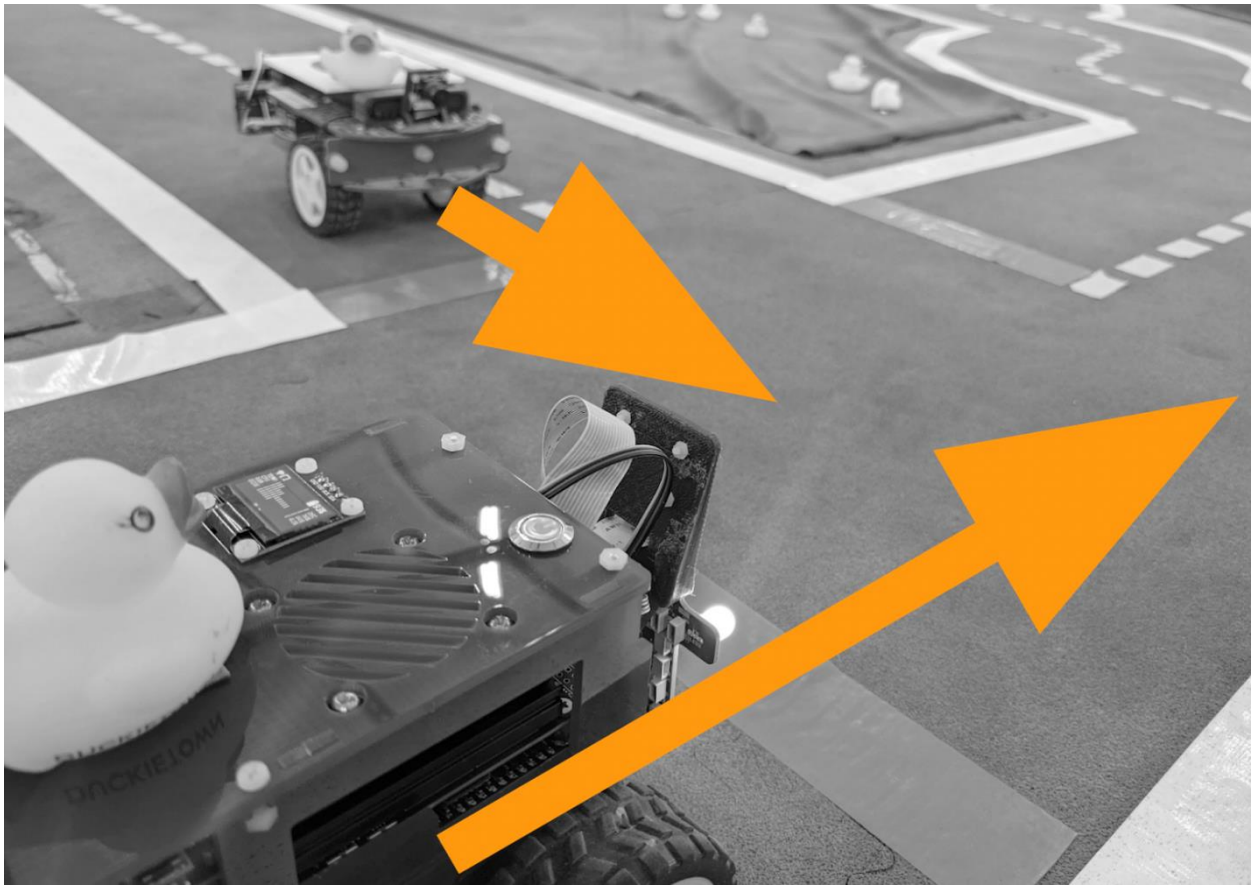
Вариант 1

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:

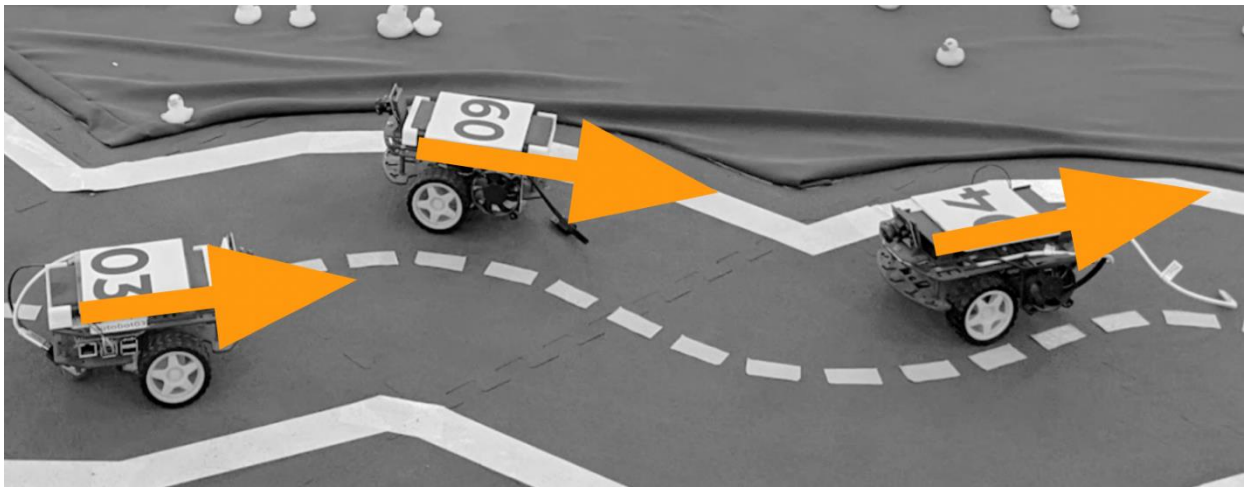
(0.91, 1.04)

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

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.]

Основные теоретические положения

В лабораторной работе была использована библиотека *NumPy*. *NumPy* — библиотека с открытым исходным кодом для языка программирования *Python*. Данная библиотека поддерживает многомерные массивы и высокоуровневые математические функции, предназначенные для работы с многомерными массивами, а также предоставляет реализации вычислительных алгоритмов, оптимизированные для работы с многомерными массивами. В программе библиотека была подключена следующим образом: *import numpy as np*, благодаря чему для обращения к методам библиотеки используется запись *np.<название метода>()* вместо *numpy.<название метода>()*.

В ходе работы были использованы следующие методы и атрибуты библиотеки *NumPy*:

- *numpy.array* — массив *numpy*, многомерный массив (*ndarray*) данных, оптимизированный для ряда операций.
- *numpy.dot* — скалярное произведение векторов, если аргументы являются одномерными массивами, или матричное произведение, если аргументы являются многомерными массивами.
- *numpy.hstack* — конкатенация по второй оси, кроме одномерных массивов (конкатенируются по первой оси).
- *numpy.linalg* — функции линейной алгебры, обеспечивает эффективную низкоуровневую реализацию стандартных алгоритмов линейной алгебры.
- *numpy.linalg.det* — вычисление определителя массива.
- *numpy.linalg.matrix_rank* — ранг квадратной матрицы.
- *numpy.linalg.solve* — решение системы линейных алгебраических уравнений. В случае неудачи вызывает ошибку *LinAlgError*.
- *numpy.round* — округление до заданного числа десятичных знаков.
- *numpy.cos* — косинус угла в радианах.
- *numpy.sin* — синус угал в радианах.

Выполнение работы

Библиотека *numpy* подключается как *np*: *import numpy as np*

Далее определяется функция *solution_exists(matrix_a, column_vector_b)*, которая проверяет, что система линейных алгебраических уравнений вида $\boldsymbol{A}x = b$, где $\boldsymbol{A}$ — некоторая матрица, b — столбец свободных членов, x — столбец неизвестных. На вход функция получает некоторую матрицу *matrix_a* и массив со свободными членами *column_vector_b*. Сначала с помощью метода *np.hstack()* создаётся расширенная матрица *ext_matrix_a*, получаемая приписыванием справа к исходной матрице столбца свободных членов. Далее с помощью оператора *if* проверяются два условия, гарантирующих наличие решения у данной СЛАУ — равенство рангов исходной и расширенной матриц (реализуется с использованием атрибута *np.linalg.matrix_rank*: *np.linalg.matrix_rank(matrix_a) == np.linalg.matrix_rank(ext_matrix_a)*) и невырожденность исходной матрицы (реализуется с использование атрибута *np.linalg.det*: *np.linalg.det(matrix_a) != 0*). При выполнении обоих условий функция возвращает значение *True*, иначе — *False*.

Для решения первой задачи была реализована функция *check_collision(bot1, bot2)*. Функция принимает два массива *ndarray*, каждый из которых состоит из трёх элементов — коэффициентов прямой, по которой движется дакибот. Элементы кортежей *bot1* и *bot2* распаковываются в тройки переменных *a1, b1, c1* и *a2, b2, c2* соответственно. Далее с помощью метода *np.array()* создаются матрица коэффициентов *matrix_a* (*ndarray* 2×2) и массив *column_b* (*ndarray* 1×2), содержащий свободный члены *c1* и *c2*, взятые со знаком “минус”. Далее производится проверка того, что уравнение *(matrix_a)·x = (column_b)* имеет решение *x*: *if solution_exists(matrix_a, column_b)*. Если условие выполняется, то вычисляется решение (*solution_matrix = np.linalg.solve(matrix_a, column_b)*), которое затем возвращает функция, округляя его значение до двух знаков после запятой, используя метод *np.round(value, 2)*; иначе программа возвращает значение *None*.

Для решения второй задачи была реализована функция *check_surface(point1, point2, point3)*. На вход функции подаются три *ndarray* 1×3, содержащие значения трёх координат соответствующего робота. Каждый из массивов распаковывается в три переменные, соответствующие значениям координат дакибота. Создаются матрица системы *matrix_a* (*ndarray* 3×3) и массив со свободными членами *column_b* (*ndarray* 1×3), решением этой СЛАУ будет являться массив искомых коэффициентов. Аналогично первой функции проводится проверка на существование решения, в зависимости от результата которой программа возвращает массив коэффициентов (*ndarray* 1×3) либо *None*.

Для решения третьей задачи была реализована функция *check_rotation(vec, rad)*. Функции подаются *ndarray* 1×3 — три координаты дакибота — и угол поворота в радианах. Создаётся матрица поворота вокруг оси *z rotation_matrix* (*ndarray* 3×3), значения элементов которой вычисляются с помощью методов *np.cos()* и *np.sin()*. Массив координат робота после поворота *vec_after_rotation* находится как матричное произведение (*np.dot()*) переменных *rotation_matrix* и *vec*, после чего его элементы с помощью *np.round(value, 2)* округляются до двух знаков после запятой. Полученный массив *ndarray* 1×3 служит выводом функции.

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, 2, 3], [2, -1, 9]	(-4.2, 0.6)	Задача 1
2.	[0, 0, 0], [1, 2, 3], [3, 2, 1]	[-1. 2. 0.]	Задача 2
3.	[1, 2, 3], 3.14	[-1. -2. 3.]	Задача 3
4.	[1, 1, 1], [2, 2, 2]	None	Задача 1
5.	[1, 1, 1], [2, 2, 2], [3, 3, 3]	None	Задача 2
6.	[42, 19, 0], 1. 57	[-18.97 42.02 0.]	Задача 3

Выводы

Были изучены основные управляющие конструкции языка Python, основы работы с библиотекой NumPy. С использованием методов и атрибутов данной библиотеки были реализованы три функции, выполняющие заданные задачи.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np

def solution_exists(matrix_a, column_vector_b):
    ext_matrix_a = np.hstack((matrix_a, [[x] for x in
column_vector_b]))
    if np.linalg.matrix_rank(matrix_a) ==
np.linalg.matrix_rank(ext_matrix_a) and np.linalg.det(matrix_a) != 0:
        return True
    else:
        return False

def check_collision(bot1, bot2):
    a1, b1, c1 = bot1
    a2, b2, c2 = bot2

    matrix_a = np.array([[a1, b1],
                         [a2, b2]])
    column_b = np.array([-c1, -c2])

    if solution_exists(matrix_a, column_b):
        solution_matrix = np.linalg.solve(matrix_a, column_b)vec
        return tuple(np.round(solution_matrix, 2))
    else:
        return None


def check_surface(point1, point2, point3):
    x1, y1, z1 = point1
    x2, y2, z2 = point2
    x3, y3, z3 = point3

    matrix_a = np.array(([x1, y1, 1],
                         [x2, y2, 1],
                         [x3, y3, 1]))
    matrix_b = np.array([z1, z2, z3])

    if solution_exists(matrix_a, matrix_b):
        solution_matrix = np.linalg.solve(matrix_a, matrix_b)
        return np.round(solution_matrix, 2)
    else:
        return None


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