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

Студент гр. 3343 _____ Малиновский А. А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Изучить основные управляющие конструкции языка Python, библиотеку NumPy, а также научиться их применять для разработки программ.

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.

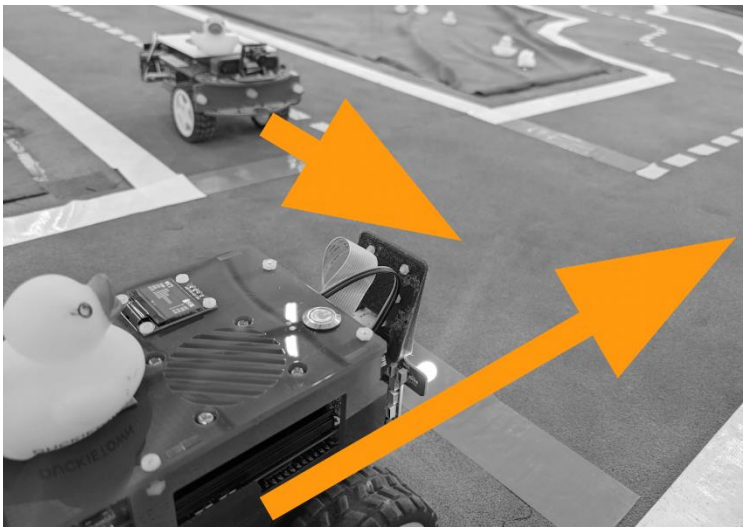

Рисунок 1 – Пример задачи 1

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)*.

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 местности, по которой они двигались. Для построения карты местности необходимо знать уравнение плоскости. Ваша задача -- помочь дакиботам найти уравнение плоскости, в которой они двигались.

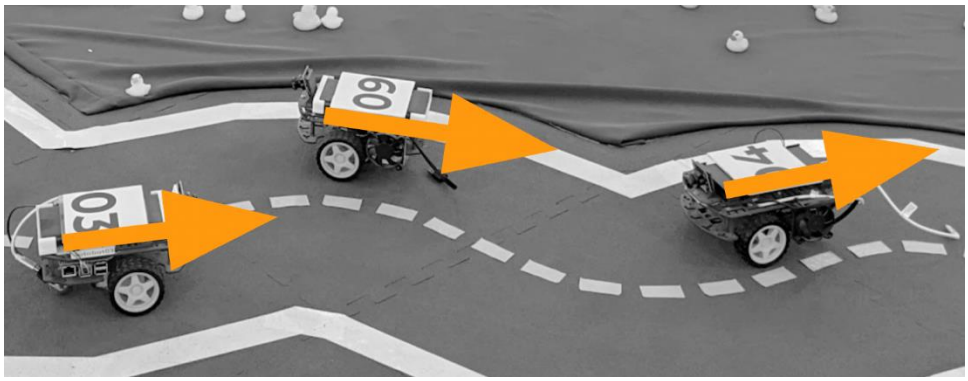

Рисунок 2 – Пример задачи 2

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)*.

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.

Рисунок 3 – Пример задачи 3

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)*..

Выполнение работы

Для решения задач была написана программа на языке Python, в которой реализуются три функции.

Решение первой представлено в функции check_collision. На вход подаются коэффициенты траекторий двух дакиботов в виде ndarray, функция возвращает точку пересечения траекторий дакиботов . Если у системы уравнений нет решения, то функция возвращает ***None***.

Вторая задача реализована с помощью функции check_surface, на вход которой подаются ndarray с координатами 3 точек. На выходе пользователь получает коэффициенты уравнения плоскости. Если у системы уравнений отсутствуют решения, то функция возвращает ***None***.

Для решения третьей задачи реализована функция check_rotation, на вход подаются ndarray с 3 координатами и угол поворота в радианах. Функция возвращает повернутые ndarray координаты .

В программе представлены следующие функции модуля NumPy:

1. *np.array()* - функция, которая создает объект типа nparray (массив/матрицу) из поданных на вход данных.
2. *np.linalg.matrix_rank()* – функция, которая вычисляет ранг матрицы.
3. *np.linalg.solve()* – функция, которая решает систему линейных уравнений, представленных в матричной форме. На вход подается матрица коэффициентов и вектор свободных членов.
4. *np.round()* – функция, округляющая все элементы объекта nparray.
5. *np.column_stack()* – функция, объединяющая одномерные массивы в качестве столбцов двумерного массива.

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.	[3,4,5] [2,4,7]	(-2.0 -2.75)	Функция 1
2.	[5,3,6] [2,5,6] [3,5,3]	[-3, -4.5, 34.5]	Функция 2
3.	[1,3,4] 4	[1.62, -2.72,4]	Функция 3

Выводы

Были изучены основные управляющие конструкции языка Python, а также модуль NumPy.

В написанной программе, содержатся три функции, предназначенные для решения трех задач. Первая функция позволяет найти точку пересечения траекторий двух различных дакиботов. Вторая функция вычисляет коэффициенты уравнения плоскости, в которой двигаются дакиботы. А третья функция помогает дакиботу вернуться в необходимое ему направление для продолжения движения.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np

def check_collision(bot1, bot2):

  #извлечение коэффицентов прямых
  a1,b1,c1=bot1
  a2,b2,c2=bot2

  # создание матрицы коэффицентов
  coeff_matrix=np.array([[a1,b1],[a2,b2]])
  #создание вектора свободных членов
  const_vector=np.array([-c1, -c2])

  #проверка ранга матрицы с коэффицентами
  if np.linalg.matrix_rank(coeff_matrix)!=2:
      return None

  #решение системы уравнений
  system_solution=np.round(np.linalg.solve
  (coeff_matrix,const_vector),2)
  intersects=(system_solution[0],system_solution[1])

  return intersects

def check_surface(point1, point2, point3):
  # создание матрицы координат точек
  points_matrix=np.array([point1,point2,point3])

  if np.linalg.matrix_rank(points_matrix)!=3:
      return None

  # извлечение координат x, y, z точек
  x=points_matrix[:,0]
  y=points_matrix[:,1]
  z=points_matrix[:,2]

  coeff_matrix= np.column_stack((x,y,np.ones(3)))
  coeff= np.round(np.linalg.solve(coeff_matrix,z),2)
  return (coeff)

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]])
  rotated_coord= np.round(np.dot(vec,rotation_matrix),2)
  return (rotated_coord)
```