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. 3342 _____ Легалов В. В.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург

2023

Цель работы

Изучить основные управляющие конструкции языка Python. С применением модуля Numpy, написать программу, состоящую из трёх функций, каждая из которых проводя операции над массивами, решает одну из поставленных задач.

Задание

Задача 1.

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).

В случае, если решение найти невозможно (например, из-за линейно зависимых векторов), функция должна вернуть ***None***.

Задача 2.

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).

В случае, если решение найти невозможно (невозможно найти коэффициенты плоскости из-за, например, линейно зависимых векторов), функция должна вернуть ***None***.

Задача 3.

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).

Выполнение работы

Для решения поставленных задач были использованы модули numpy и math, а также составленны 3 функции:

def check_collision(bot1, bot2): В качестве аргументов в функцию передаются кооффициенты двух уравнений прямых, функция возвращает координаты точки пересечения траекторий в виде кортежа 2 значений. При помощи методов *vstack((bot1, bot2)).T* из двух массивов коофициентов создаётся матрица размером 2 на 3, а затем транспонируется в матрицу размером 3 на 2 и присваивается переменной *data*. Таким образом первая и вторая строки матрицы *data* содержат коофициенты *[a1, a2]* и *[b1, b2]* соответственно, в третьей строке содержаться свободные члены уравнений *[c1, c2]*. Переменной *ratio1* присваевается транспонированная матрица, состоящая из первых двух строк матрицы *data*, в переменную *ratio2* записывается последняя строка матрицы *data* умноженная на *-1*. При помощи метода *linalg.matrix_rank(ratio1)* определяется ранг матрицы. В случае, если ранг матрицы не равен 2, решение найти невозможно и функция возвращает *None*, в случае, если решение возможно найти, система линейных уравнений решается с помощью метода *linalg.solve(ratio1, ratio2)*, значения округляются до 2 знаков после запятой и возвращаются в виде кортежа 2 значений.

def check_surface(point1, point2, point3): В качестве аргументов в функцию передаются координаты трёх точек в виде трёх массивов, функция возвращает коофициенты a, b, c в виде *ndarray* для уравнения плоскости вида ax+by+c=z. При помощи методов *vstack((point1, point2, point3)).T* из трёх массивов создаётся матрица размером 3 на 3, а затем транспонируется и присваивается переменной *data*. Первая, вторая и третья строки матрицы *data* содержат значения [x1, x2, x3], [y1, y2, y3] и [z1, z2, z3] соответственно. Переменной *ratio1* присваевается транспонированная матрица, состоящая из первых двух строк матрицы *data* и строки единиц, так каждая строка *ratio1* содержит значения: [x, y, 1.]. В переменную *ratio2* записывается последняя

строка матрицы *data*. При помощи метода *linalg.matrix_rank(ratio1)* определяется ранг матрицы. В случае, если ранг матрицы не равен 3, решение найти невозможно и функция возвращает *None*, в случае, если решение возможно найти, система линейных уравнений решается с помощью метода *linalg.solve(ratio1, ratio2)*, значения результата округляются до 2 знаков после запятой и возвращаются в массиве *ndarray*.

def check_rotation(vec, rad): В качестве аргументов в функцию подаются массив, содержащий координату и угол поворота в радианах. Функция возвращает повёрнутые вокруг оси z *ndarray* координаты. В *matrix_rotation* присваивается матрица поворота для оси z (см. рис. 1). Значения, полученные с помощью метода *dot(matrix_rotation, vec)*, округляются до 2 знаков после запятой и возвращаются в массиве *ndarray*.

$$M_z(\varphi) = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Рисунок 1 - Матрица поврота вокруг оси z

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	check_collision(np.array([-3, -6, 9]), np.array([8, -7, 0]))	(0.91, 1.04)
2.	check_surface(np.array([1, -6, 1]), np.array([0, -3, 2]), np.array([-3, 0, -1]))	[2., 1., 5.]
3.	check_rotation(np.array([1, -2, 3]), 1.57)	[2., 1., 3.]

Выводы

Были использованы методы модуля numpy для работы с матрицами, написаны функции, решающие алгебраические задачи.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
import numpy as np
from math import cos
from math import sin


def check_collision(bot1, bot2):
    data = np.vstack((bot1, bot2)).T
    ratio1 = data[0:2].T
    ratio2 = data[-1] * -1
    if np.linalg.matrix_rank(ratio1) != 2:
        return None
    else:
          return tuple([round(i, 2) for i in np.linalg.solve(ratio1,
ratio2)])



def check_surface(point1, point2, point3):
    data = np.vstack((point1, point2, point3)).T
    ratio1 = np.vstack((data[0:2:], np.ones(3))).T
    ratio2 = data[-1]
    if np.linalg.matrix_rank(ratio1) != 3:
        return None
    else:
         return np.array([round(i, 2) for i in np.linalg.solve(ratio1,
ratio2)])


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