МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Информатика»

Тема: Основные управляющие конструкции языка Python

Студентка гр. 2381 _____ Слабнова Д.А.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2022

Цель работы.

Написать три функции, каждая из которых решает одну из следующих задач:

1. Определить, находится ли точка внутри прямоугольника, получая на вход координаты данной точки и вершин прямоугольника.

2. Определить список столкнувшихся дакиботов, получая на вход коэффициенты уравнений их движения.

3. Определить пройденный дакиботом путь, получая на вход координаты пройденных точек и считая, что между точками дакибот движется прямолинейно.

Основные теоретические положения.

Элементы линейной алгебры

Матрица — набор элементов(таблица) n*m, элемент матрицы при этом обозначается a_{ij}, где $0 \leq i \leq n$ $0 \leq j \leq m$ — номера строки и столбца соответственно. Ранг матрицы — количество линейно независимых строк. Строка линейно независима от некого множества других строк, если не может быть представлена в виде их линейной комбинации. Пусть есть система из n линейных уравнений с m неизвестных (так i-ое уравнение будет выглядеть как $a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 + \ldots + a_{im}x_m = b_i$). Тогда матрицей коэффициентов будет называться матрица (a_{ij}) n*m, где a_{ij} – jый коэффициент iого уравнения, столбец правых частей — (b_{j1}), где $0 \leq j \leq n$, столбец решений уравнения(если они есть) — (x_{j1}), где $0 \leq j \leq n$. Если ранг матрицы коэффициентов равен n то система имеет единственное решение, если меньше — бесконечное множество решений, если 0 — не имеет решений.

Библиотека NumPy. Используемые функции.

numpy.linalg.matrix_rank(A) — данная функция принимает в качестве аргумента ndarray — матрицу A и возвращает её ранг.

numpy.vstack(tup) — данная функция принимает в качестве аргумента кортеж ndarray — матриц с одинаковым количеством столбцов. Возвращает

новою матрицу, полученную «склейкой» (stack) вышеупомянутых матриц (каждая последующая матрица дописывается «внизу» результата).

numpy.linalg.norm(a) — данная функция принимает на вход ndarray — строку — вектор и возвращает его длину.

Ход работы.

check_crossroad().

Первая функция получает на вход координаты дакибота, а также вершины «перекрёстка» - прямоугольника. Переменные y1, y2, x1, x2 соответствуют «стенкам» прямоугольника, то есть функциям вида y = const и x = const, которые представляют собой прямые содержащие стороны прямоугольника. Далее функция с помощью операций сравнения определяет, находится ли координата робота в прямоугольнике ограниченном данными прямыми. Можно заметить, что при поиске значения «стенки» не используется координата четвёртой точки. Дело в том, что у соседних вершин прямоугольника всегда совпадает либо координата по оси OX, либо по оси OY. Таким образом легко заметить, что каждое значение в списке из координат x или y будет повторяться дважды, и следовательно, если мы уберём один элемент, то в списке всё равно останется его дубликат, и так как нам важны только минимальное и максимальное значения x и y, то на результат это не повлияет. Или, короче говоря, из геометрии известно что, чтобы задать прямоугольник, достаточно знать координаты трёх его вершин.

check_collision(), numpy.linalg.matrix_rank(), numpy.vstack()

Была написана вспомогательная функция check(), поверяющая траектории двух дакиботов на возможность столкновения и соответственно возвращающая True если столкновение произошло и False — если нет. В данной функции используется метод модуля numpy вычисляющий ранг матрицы A полученной на вход numpy.linalg.matrix_rank(A). Ранг матрицы — это количество линейно независимых строк(столбцов) квадратной матрицы. Ранг матрицы коэффициентов при неизвестных, если равен количеству строк(столбцов), говорит о том, что система имеет решение, а в противном

случае, что система не имеет решения или имеет бесконечное количество решений (в нашем случае бесконечное количество решений означало бы то, что дакиботы двигались по одной и той же траектории, но предполагается, что такая ситуация невозможна). check() получает ndarray содержащий квадратную матрицу 2*2, где первая строка коэффициенты при x и y для траектории первого дакибота, вторая — для траектории второго. check() возвращает True, если ранг матрицы равен двум, и False, если ранг меньше двух.

Функция check_collision() получает на вход ndarray с коэффициентами уравнений движения дакиботов. Далее с помощью двух циклов for функция перебирает все возможные пары, «склеивая» - два вектора коэффициентов при x и y (здесь можно заметить что коэффициент c неважен) с помощью модуля numpy numpy.vstack(). numpy.vstack() получает на вход кортеж с несколькими матрицами с одинаковым количеством столбцов и возвращает матрицу состоящую «дописанных вертикально» (vstack — vertically stack) строк всех введённых матриц. Далее полученная пара — матрица 2*2 содержащая коэффеценты при x и y проверяется ранее описанной функцией check(), и в зависимости от этого добавляется или не добавляется в массив результатов clsins. Также я считаю излишним писать повторяющиеся пары (например (2, 0) и (0, 2)).

check_path(), numpy.linalg.norm()

Функция получает на вход массив кортежей — пройденных дакиботом точек. Хоть в задании и не сказано, но предполагается, что от точки до точки дакибот двигается прямолинейно. Для удобства массив кортежей сразу преобразуется в ndarray. Далее с помощью цикла for функция рассчитывает расстояния между двумя соседними точками и прибавляет его к всему расстоянию a. points_list[i] — points_list[i+1] это вектор, модуль которого равен расстоянию между iой и i+1ой точкой. numpy.linalg.norm() - модуль numpy, который получает на вход вектор(некое множество значений x_1, x_2, … x_n), а возвращает его модуль($(x_1^2 + x_2^2 + \ldots + x_n^2)^{0.5}$).

Выводы.

Установленные цели работы были достигнуты. Были освоены простейшие методы модуля numpy, в частности пакета numpy.linalg.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

```
import numpy as np
def check_crossroad(robot, point1, point2, point3, point4):
  xfirst = max(point1[0], point2[0], point3[0])
  xsecond = min(point1[0], point2[0], point3[0])
  yfirst = max(point1[1], point2[1], point3[1])
  ysecond = min(point1[1], point2[1], point3[1])
  x, y = robot
  return  xfirst >= x >= xsecond and  yfirst >= y >=ysecond
def check(a):
  return np.linalg.matrix_rank(a) == 2
def check_collision(coefficients):
  clsins = []
  for i in range(len(coefficients)):
    tmp = coefficients[i][:2]
    for j in range(i, len(coefficients)):
      if check(np.vstack((tmp, coefficients[j][:2]))):
        clsins.append((i, j))
  return clsins
def check_path(points_list):
  points_list = np.array(points_list)
  length = 0
  for i in range(len(points_list) - 1):
    length += np.linalg.norm((points_list[i] - points_list[i+1]))
  return round(length, 2)
```