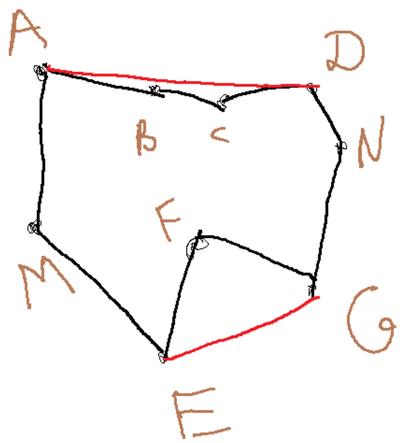
Работа с фигурой, заданной последовательностью точек

Леманский К.Ю., студент ВИШ РУТ МИИТ

Цель: Целью статьи является вывод и описание методики преобразования фигуры заданной последовательностью точек в набор точек принадлежащей фигуре.
Задачи: Проверка принадлежности точки выпуклой фигуре, проверка принадлежности точки не выпуклой фигуре, преобразование не выпуклой фигуры, заданной последовательностью точек, в набор точек принадлежащих ей.
Актуальность: На сегодняшний день техника обработки карты содержащий данные является весьма востребованной, поэтому методику можно пременить в большом спектре задач при работе с картами и данными карты(например GeoJson).

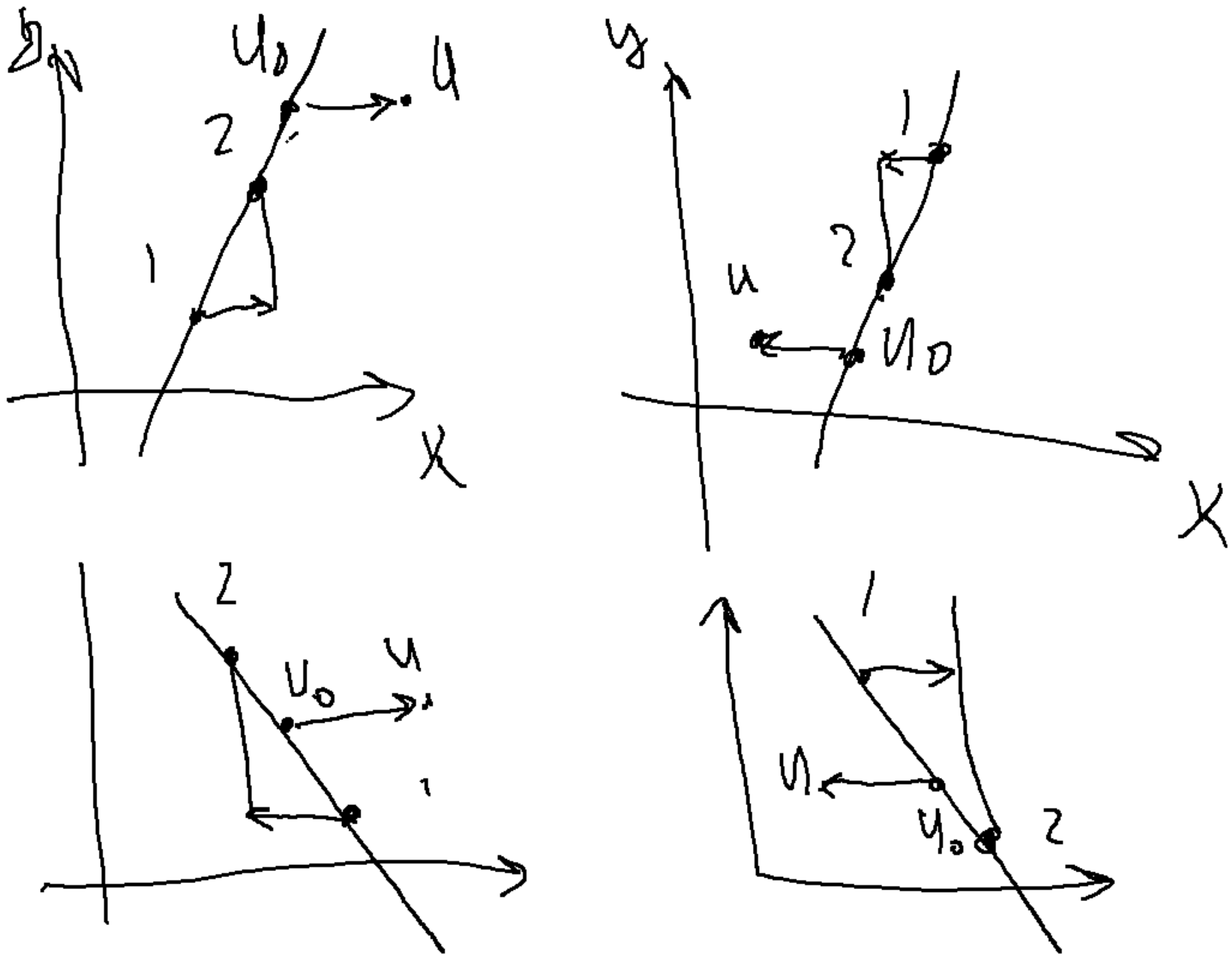
Проблема: Входные данные - не всегда выпуклые многоугольники занчит определить принадлежит ли точка многоугольнику нельзя просто проверив принадлежит ли точка всем полуплоскостям образованными гранями.

Решение: Изменим многоугольник вырезая вершины так, что он станет выпуклым. Получим:



в даном случае остается проверить для точки $U = (x, y)$, $U \in ADGEM \cap U \notin FEG \cap U \notin ABCD$.

Как же найти такие EFG? Как мы знаем уравнение прямой выглядит так: $\dfrac{x - x_1}{x_2 - x_1} = \dfrac{y - y_1}{y_2 - y_1}$. Теперь как определить что точка лежит справа? Можно просто посмоттеть где прямая через т U пересекает ось X например:



Подставим y от U в уравнение получим: $x_3 = \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} + x_1$ тогда $U_0 = (x_3, y_U)$ тогда если $\dfrac{x_1 - x_2}{y_1 - y_2} > 0$ то U должна лежать правее точки U_0 получаем по знаку $x_U - \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} - x_1 \equiv x_2 - x_1$ иначе U должна лежать левее точки U_0 получаем по знак $x_U - \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} - x_1 \equiv -(x_2 - x_1)$ соеденяем

получаем $x_U - \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} - x_1 \equiv \dfrac{x_1 - x_2}{y_1 - y_2} \cdot (x_2 - x_1) \implies (x_U - x_1) - \dfrac{(y_U - y_1)(x_2 - x_1)}{y_2 - y_1} \equiv \dfrac{(x_2 - x_1)^2}{y_2 - y_1} \implies (x_U - x_1)(y_2 - y_1) - (y_U - y_1)(x_2 - x_1) \equiv (x_2 - x_1)^2 \implies (x_U - x_1)(y_2 - y_1) - (y_U - y_1)(x_2 - x_1) \equiv 1 \implies (x_U - x_1)(y_2 - y_1) > (y_U - y_1)(x_2 - x_1)$.
Случай $U \in \dfrac{x - x_1}{x_2 - x_1} = \dfrac{y - y_1}{y_2 - y_1}$ нужно расмотреть отдельно но пусть в таком счучае будем считать, что принадлежит.

Итак формулы мы получили начнем писать код. Для начала хорошо бы создать класс точка с полями x, y. Так же прописать функцию поиска растояния и функции отрезка. Функцию отрезка можно получить вида $y = f(x)$ из ранее уже выведеной.

```
import math
class point:
    def __init__(self, x:float, y:float):
        self.x = x
        self.y = y

    def lenf(self, other):
        return math.sqrt((self.x - other.x)** 2 + (self.y - other

    def equatian(self, other, **printer):
        x_1, y_1 = self.x, self.y
        x_2, y_2 = other.x, other.y
        equatin = f"x * {(y_2 - y_1) / (x_2 - x_1)} - {x_1 * (y_2
        if printer:
            print(f"y = {equatin}")
```

```
        return equatin

    def __hash__(self):
        return hash((self.x, self.y))
```

И для красоты добавим функции print-a.

```
    def __str__(self):
        return f"({self.x}, {self.y})"

    def __repr__(self):
        return f"<<{self.x}, {self.y}>>"
```

Проверим "лежит точка справа от отрезка?": Функция принисает 3 точки 2 из которых задают направленую прямую и 3-я это проверяемая точка. Все необходимые функции мы уже вывели.

```
def point_right_line(point1, point2, u) -> bool:
    return (u.x - point1.x) * (point2.y - point1.y) > (u.y - poin
```

Проверяем лежит ли точка в выпуклой фигуре.

```
def point_in_normal_figure(pl:List[Point], u:Point):
    if len(pl) < 3: print("errore!!!!! Beckend  line")
    orintation_right = point_right_line(pl[0], pl[1], pl[2])
    point1 = pl[0]
    for point2 in pl[1:]:
        if orintation_right != point_right_line(point1, point2, u)
            return False
        point1 = point2
```

```
    if orintation_right != point_right_line(pl[-1], plt[0], u):
        return False
    return True
```

А если многоугольник не выпуклый? Как уже было сказанно, будем делит на выпуклые фигуры и для них проверять лежит ли в них фигура. Самое сложное определить находится внутряняя часть фигуры справа или слева. Я предлагаю способ, который возможно будет работать не вегда, но в большинстве способов. Просто определить для каждого отрезка сколько точек лежит справа сколько слева. И так для всех и сложить. Получим два числа например r, l(r-спрва l-слева). И тогда если $r > l$ то спарава иначе слева.

Тогда код будет выглядеть так:

```
def orintation_righ(main_fig:List[Point]):
    r, l = 0, 0
    temp_ln = len(main_fig)
    for i in range(temp_ln):
        for j in range(temp_ln):
            if j != i and j != (i + 1) % temp_ln:
                if point_right_line(main_fig[i], main_fig[(i + 1)
                r += 1
            else:
                l += 1
    return r > l
```

```
def point_in_bad_figure(main_fig:List[Point], u:Point):
    added_fig = []
    orintation_right = orintation_righ(main_fig)
    count = 0
    i = 0
    while True:
        points = [main_fig[i], main_fig[(i + 2)% len(main_fig)],(
         main_fig[(i + 1)% len(main_fig))]]
        if (point_right_line(points[0], points[1], points[2])and(
        not orintation_right)) or (
        (not point_right_line(points[0], points[1], points[2]))and
         orintation_right):
            count += 1
        else:
            added_fig += [points]
            main_fig.remove(main_fig[(i + 1)% len(main_fig)])
            count = 0
        i += 1
        if count + 3 == len(main_fig):
            break
    in_fig = point_in_normal_figure(main_fig, u)
    for i in added_fig:
        in_fig = infig and not point_in_normal_figure(i, u)
    return in_fig
```

И наконец функция разбиенияя на точки.

```
def bad_fig_to_point(main_fig:List[Point], step:int):
    fig_list = []
    maxx, maxy = minx, miny =  (
    list(main_fig.keys())[0].x, list(main_fig.keys())[0].y)
```

```
for i in main_fig.keys():
    if i.x > maxx:
        maxx = i.x
    if i.x < min.x:
        minx = i.x
    if i.y > maxy:
        maxy = i.y
    if i.y < min.y:
        miny = i.y
x_list = list(x * step for x in range(minx / step, maxx / step)
y_list = list(y * step for y in range(miny/ step, maxy / step))
for x, y in itertools.product(x_list, y_list):
    point = Point(x, y)
    if point_in_bad_figure(main_fig, point):
        fig_list += [point]
return fig_list
```

Ок, а что на счет 3D? Думаю о направлении тут говорить нет смысла, так что прийдется использовать какой-то другой метод. Можно через каждую точку проводить прямую параллельную оси x и смотреть пересечения с фигурой если пересечения с каждой стороны нечетное кол-во то точка внутри иначе снаружи. Стоит учесть что это работает потому, что можно говорить о вхождении/выходе прямой из фигуры, но нужно сказать, что есть вид пересечение 2 граней в месте ребра и в таком месте может быть выход прямой из фигуры или его может и не быть. Эта проблема выпуклости, и решать ее будем так же, те разделем невыпуклую фигуру на выпуклые и обработаем включения/искючения.

Итак, начнем опять с точки.

```
class Point3:
    def __init__(self, x:float , y:float, z:float):
        self.x = x
        self.y = y
        self.z = z
```

Теперь класс отрезка. Определять мы будем по 2 точкам. Так же добавим функцию проверки лежит ли точка на прямой. Только есть проблема с типом float во всем програмировании, но ее можно решить просто добавив метод round тогда прямая конечно станет условно широкаой, но для нашей задачи ширинва 10^{-5} не так уж и страшно. Алгоритм аналогичен 2D проверка истиности $\dfrac{x - x_1}{x_2 - x_1}$

```
class line:
    def __init__(self, p1:Point3, p2:Point3):
        self.p1 = p1
        self.p2 = p2

    def include(self, p:Point3):
        return round((self.p.y - self.p2.y) *
         (self.p1.x - self.p2.x) - (self.p.x - self.p2.x)
          * (self.p1.y - self.p2.y), 5) == round(
          (self.p2.y - self.p.y) *  (self.p1.z - self.p2.z) -
          (self.p2.z - self.p.z)
          * (self.p1.y - self.p2.y), 5) == 0
```

Теперь плоскость.

```
class plane:
    def __init__(self, *lines):
```