Поиск максимума по строке

Здесь почти у всех все было хорошо, никаких сложностей быть не должно :)

Найдите все индексы, где достигаются максимумы по строке в массиве B. **(0.5 балла)**

Подсказка: `np.argmax` тут не подойдет. Почему?

In [160]:
```
B = [[0, 0, 3, 1, 3, 2],
     [2, 3, 0, 0, 3, 0],
     [0, 1, 2, 0, 0, 2],
     [2, 0, 2, 0, 2, 1],
     [1, 1, 0, 0, 1, 3]]
B = np.asarray(B)
B
```

Out[160]:
```
array([[0, 0, 3, 1, 3, 2],
       [2, 3, 0, 0, 3, 0],
       [0, 1, 2, 0, 0, 2],
       [2, 0, 2, 0, 2, 1],
       [1, 1, 0, 0, 1, 3]])
```

In [161]:
```
np.where(B == B.max(axis=1)[:, np.newaxis])
np.where(B == B.max(axis=1, keepdims=True))
```

Out[161]:
```
(array([0, 0, 1, 1, 2, 2, 3, 3, 3, 4]), array([2, 4, 1, 4, 2, 5, 0, 2, 4, 5]))
```

Комбинация матрицы A и бесконечности

Нужно было заглянуть в документацию np.where, там можно увидеть вот такой трюк

Рассмотрим массив A. Модифицируйте предыдущее решение так, чтобы получился массив C следующего вида **(0.5 балла)**:

- значения в матрице C на позициях, где условие выполнено, совпадает со значениями на позициях в матрице A;
- иначе – `-np.inf`.

Сохраните результат в переменную C.

In [23]:
```
A = [[3, 1, 4, 5, 1, 2],
     [3, 3, 2, 4, 2, 2],
     [4, 1, 0, 1, 0, 2],
     [1, 0, 4, 3, 4, 3],
     [4, 2, 0, 1, 3, 4]]
A = np.asarray(A)
A
```

Out[23]:
```
array([[3, 1, 4, 5, 1, 2],
       [3, 3, 2, 4, 2, 2],
       [4, 1, 0, 1, 0, 2],
       [1, 0, 4, 3, 4, 3],
       [4, 2, 0, 1, 3, 4]])
```

In [24]:
```
C = np.where(B == B.max(axis=1)[:,np.newaxis], A, -np.inf)
C
```

Out[24]:
```
array([[-inf, -inf,   4., -inf,   1., -inf],
       [-inf,   3., -inf, -inf,   2., -inf],
       [-inf, -inf,   0., -inf, -inf,   2.],
       [  1., -inf,   4., -inf,   4., -inf],
       [-inf, -inf, -inf, -inf, -inf,   4.]])
```

Умножение матрицы и скалярное произведение

Все понимают, что np.dot и np.matmul делают одно и то же. Тем не менее, я бы очень рекомендовал использовать np.dot, когда по смыслу идет скалярное произведение, а np.matmul – умножение матриц. Это сделает ваш код более читабельным и понятным

L1-норма

Еще одна рекомендация – не изобретайте велосипед. Все мы склонны ошибаться в мелочах. np.linalg.norm :)

Find Equal Lines

Хорошее упражнение на развитие пространственного мышления. Здесь подразумевалось, что вы разберетесь с тем, что такое приведение размерностей.

Даны два небольших массива A размера $M \times N$ и B размера $K \times N$.

Напишите функцию `find_equal_lines(A, B)`, которая найдет все индексы строк массива A, содержащиеся в B.

Запрещается использовать циклы. **(1 балл)**

Подсказка: попробуйте поэкспериментировать с размерностью массивов A и B.

In [162]:
```
def find_equal_lines(A, B):
    mask = (A[:,np.newaxis,:] == B[np.newaxis,:,:]).all(axis=2).any(axis=1)
    return np.where(mask)[0]
```

Минус данного решения в том, что оно не будет работать при матрицах большого размера, так как результат сравнения – трехмерная матрица. Пусть A.shape = (N, D), B.shape = (M, D), тогда результат сравнения имеет размеры (N, M D).

Изображения

img.mean(axis=(0,1))
img.mean(axis=0).mean(axis=0)

Голосование

np.apply_along_axis(lambda r: np.bincount(r).argmax(), 1, y_pred)
scipy.stats.mode(y_pred, axis=1).mode.flatten()

Первый вариант лучше тем, что позволяет производить взвешенное голосование (у каждого объекта свой вес в голосовании). Если голосование обычное, то второй вариант, конечно, более элегантный

Снежинка Коха

С первыми двумя пунктами проблем не должно было возникнуть ни у кого. Там все очень просто

Шаг 1: Напишите функцию, которая извлекает корень из 1. **(0.25 балла)**

In [115]:
```
def complex_roots(n):
    angles = np.linspace(0, 2 * np.pi, n, endpoint=False)
    return np.exp(angles * 1j)

roots = complex_roots(5)
roots
```

Out[115]:
```
array([ 1.        +0.j        ,  0.30901699+0.95105652j,
       -0.80901699+0.58778525j, -0.80901699-0.58778525j,
        0.30901699-0.95105652j])
```

Кто-то делал первый пункт через формулу Муавра-Лапласа, тоже вполне допустимый вариант.

In [117]:
```
def koch_curve():
    arr = [0, 1/3, 0.5 + 0.5j * np.sqrt(3) / 3, 2/3, 1]
    return np.asarray(arr)[::-1]

curve = koch_curve()
curve
```

Out[117]:
```
array([1.        +0.j        , 0.66666667+0.j        ,
       0.5       +0.28867513j, 0.33333333+0.j        ,
       0.        +0.j        ])
```

У многих возникли проблемы как раз с 3-м пунктом. Нужно было вспомнить, что прямую можно задать параметрически через точку на прямой и направляющий вектор:

$$l(t) = a_0 + at$$

Самым простым решением было воспользоваться этой идеей: кривая Коха имеет направляющий вектор (0, 1), нужно его поменять на новый направляющий вектор (ребро многоугольника).

Шаг 3: Заменим стороны в пятиугольнике выше на кривые Коха первого порядка. **(0.5 балла)**

Замечание: Чтобы всё точно получилось, проверьте, что обход кривых на шагах 1 и 2 совпадает (в обоих случаях либо по часовой, либо против).

In [153]:
```
def change_edges_to_koch_curves(vertices):
    curve = koch_curve()[:-1]

    p1 = vertices
    p2 = np.roll(p1, shift=1)
    dp = p2 - p1

    point = p1[:,np.newaxis] + dp[:,np.newaxis] * curve[np.newaxis,:]
    return point.ravel()

vertices = complex_roots(3)
line = change_edges_to_koch_curves(vertices)
line.shape
```

Out[153]:
```
(12,)
```

Здесь p1 – аналог a_0, dp – новый направляющий вектор.

Дальше все работает из коробки :)

Ранжирование и сортировка

Это задачка для вашей домашки по kNN. Кто догадался, что ее можно применить там, тот молодец и мог получить реализацию, которая работает быстрее sklearn.

Шаг 3: (0.25 балла)

Поддержите в функции `get_best_ranks` аргумент `return_ranks=True`. В этом случае функция должна возвращать еще и значения ранков, см. реализацию `get_best_ranks_slow`.

Замечание: поддержка аргумента не должна приводить к увеличению сложности, т.е. реализация в функции `get_best_ranks_slow` неверная.

In [78]:
```
_, ranks_result = get_best_ranks_slow(ranks_raw, top=15, return_ranks=True)
```

In [79]:
```
%%time

def get_best_ranks(ranks, top=10, return_ranks=False):
    indices = np.argpartition(ranks, -top, axis=1)[:,-top:]
    ranks_top = np.take_along_axis(ranks, indices, axis=1)
    indices = np.take_along_axis(indices, ranks_top.argsort(axis=1)[:,::-1], axis=1)
    result = (indices, )

    if return_ranks:
        ranks = np.take_along_axis(ranks, indices, axis=1)
        result += (ranks, )

    return result if len(result) > 1 else result[0]

_, ranks = get_best_ranks(ranks_raw, top=15, return_ranks=True)
```

```
CPU times: user 31.5 ms, sys: 12 ms, total: 43.5 ms
Wall time: 42.8 ms
```

In [80]:
```
assert np.isclose(ranks, ranks_result).all()
```

Сложность алгоритма: O(N) + O(K log K).

Если кто-то хочет еще поломать мозги, предлагаю подумать над введением параметра axis, отвечающего за ось по которой ищутся наибольшие значения.

Были варианты вида

```
np.argpartion(ranks, np.arange(-top, 0))
```

Такое решение более короткое (не нужно добавлять сортировку), но более медленное, так как работает за O(K N). При K << N оно проигрывает реализации выше.