Задание

Реализовать алгоритм SVD-разложения на цветных картинках для разных количеств r сингулярных значений.

Пункт первый. Подготовка

Импортируем нужные библиотеки. Из библиотеки PIL для работы с изображениями потребуется импортировать класс Image. Для работы с матрицами импортируем библиотеку numpy.

In [1]:
```
from PIL import Image
import numpy as np
```

Заранее в папку с jupyter блокнотом поместим картинку для последующей обработки в jpg формате, назвав ее image. Откроем картинку для работы с помощью метода open().

In [2]:
```
image = Image.open('image.jpg')
```

Пункт второй. Обработка исходного изображения

В переменную x занесем нашу картинку в виде массива numpy. Выделим цветовые каналы в отдельные переменные с помощью слайсов. Заметим, что каналы будут иметь размерность, отличную от изначальной картинки, а значит, для явного и понятного обратного преобразования потребуется reshape(). Сделать это можно только после разложения SVD, поскольку SVD предназначено только для работы с матрицами.

In [3]:
```
x = np.array(image, dtype = np.float32)
red = x[:,:,0]
green = x[:,:,1]
blue = x[:,:,2]
```

In [27]:
```
print(x.shape)
print(red.shape)
```

```
(3840, 2160, 3)
(3840, 2160)
```

Пункт третий. SVD-разложение RGB каналов

На этом шаге достаточно применить функцию np.linalg.svd. Для экономии места мы будем получать не полные матрицы, а только требуемые для дальнейшей обработки их части.

In [28]:
```
U_red, S_red, V_red = np.linalg.svd(red, full_matrices = False)
U_green, S_green, V_green = np.linalg.svd(green, full_matrices = False)
U_blue, S_blue, V_blue = np.linalg.svd(blue, full_matrices = False)
```

Пункт четвертый. Получение и сохранение изображений

Чтобы создать изображения, соответствующие разному количеству r сингулярных значений, нам потребуется цикл по r. Можно заметить, что хотя сингулярных значений всего 2160, наибольшая

разница между картинками видна на промежутке от 1 до 101. В дальнейшем разница практически не заметна. Поэтому я создала цикл, проходящий по полуинтервалу от 1 до 101 с более мелким шагом (10), а затем по полуинтервалу от 101 до 2160 с более крупным шагом (100).

Соответствующую каналу матрицу мы получаем с помощью умножения U, S и V матриц по нужным нам строкам и столбцам. Однако по причине того, что сейчас мы работаем с типом np.float32, а должны будем затем перевести в np.uint8, может возникнуть переполнение, а значит, артефакты изображения. Для борьбы с ними мы все отрицательные значения заменяем на ноли, а превышающие 255 - на 255. Затем нужно добавить третье измерение каналу. Эти операции мы проводим со всеми тремя каналами.

С помощью функции concatenate по оси 2 мы получаем матрицу, аналогичную той, что соответствует исходному изображению. Методом fromarray() и save() мы переводим получившиеся матрицы для каждого значения r в цветные картинки и сохраняем их в подкаталоге Images текущего каталога в формате png под именем, соответствующим значению r.

In [25]:
```
for r in list(range(1,101,10))+list(range(101,2160,100)):
    Y_r_red = U_red[:,:r].dot(np.diag(S_red[:r])).dot(V_red[:r,:])
    Y_r_red[Y_r_red < 0] = 0
    Y_r_red[Y_r_red > 255] = 255
    Y_r_red = Y_r_red.reshape(3840, 2160, 1)
    Y_r_green = U_green[:,:r].dot(np.diag(S_green[:r])).dot(V_green[:r,:])
    Y_r_green[Y_r_green < 0] = 0
    Y_r_green[Y_r_green > 255] = 255
    Y_r_green = Y_r_green.reshape(3840, 2160, 1)
    Y_r_blue = U_blue[:,:r].dot(np.diag(S_blue[:r])).dot(V_blue[:r,:])
    Y_r_blue[Y_r_blue < 0] = 0
    Y_r_blue[Y_r_blue > 255] = 255
    Y_r_blue = Y_r_blue.reshape(3840, 2160, 1)
    Y_r = np.concatenate((np.concatenate((Y_r_red, Y_r_green), axis = 2), Y_r_blue), axis = 2
    Image.fromarray(np.asarray(Y_r, dtype = np.uint8)).save(f'Images\\{r}.png')
```