### Листинг 2.1. Поиск профиля штрих-кода вдоль оси Ox.

**Профиль изображения**

Профилем изображения вдоль некоторой линии называется функция интенсивности изображения, распределенного вдоль данной линии (прорезки). Простейшим случаем профиля изображения является профиль строки:

Profile 𝑖(𝑥) = 𝐼(𝑥,𝑖), (1.8)

где 𝑖 — номер строки изображения 𝐼.

Профиль столбца изображения:

Profile 𝑗(𝑦) = 𝐼(𝑗,𝑦), (1.9)

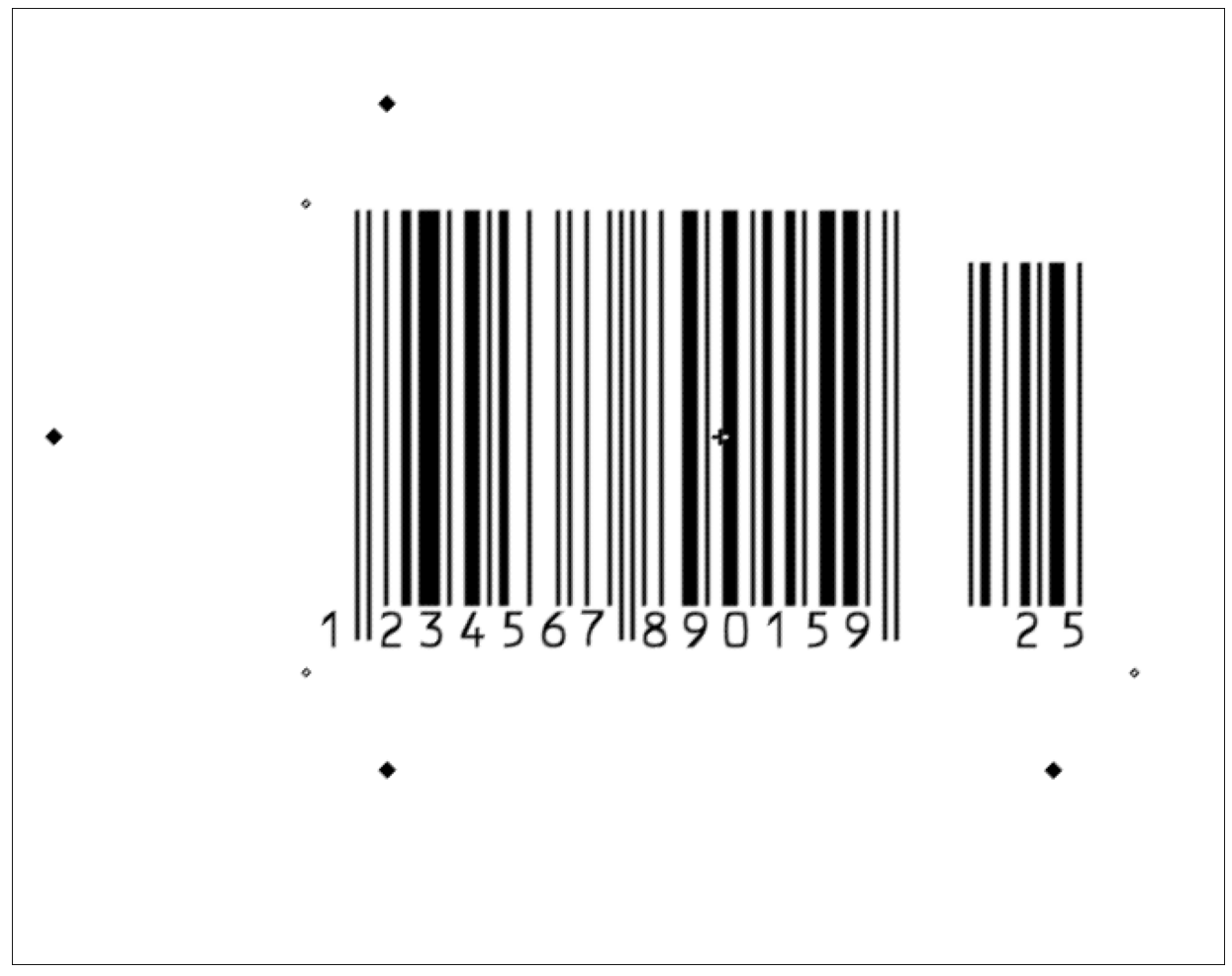
где 𝑗 — номер столбца изображения 𝐼.

В общем случае профиль можно рассматривать вдоль любой прямой, ломаной или кривой линии, пересекающей изображение. После формирования массива профиля изображения проводится его анализ стандартными средствами. Анализ позволяет автоматически выделять особые точки функции профиля, соответствующие контурам изображения, пересекаемым данной линией.

import cv2  
import numpy as np  
from numpy.linalg import norm  
from matplotlib import pyplot as plt  
from matplotlib.pyplot import figure  
from numba import njit, jit  
from math import \*  
import imutils  
import warnings  
from scipy.signal import find\_peaks  
warnings.filterwarnings('ignore')  
export\_parallel = True  
use\_graph = True

# Считаю обычный штрих код в чёрно-белом формате.  
image = cv2.cvtColor(cv2.imread("data/barcode.png"), cv2.COLOR\_BGR2GRAY)

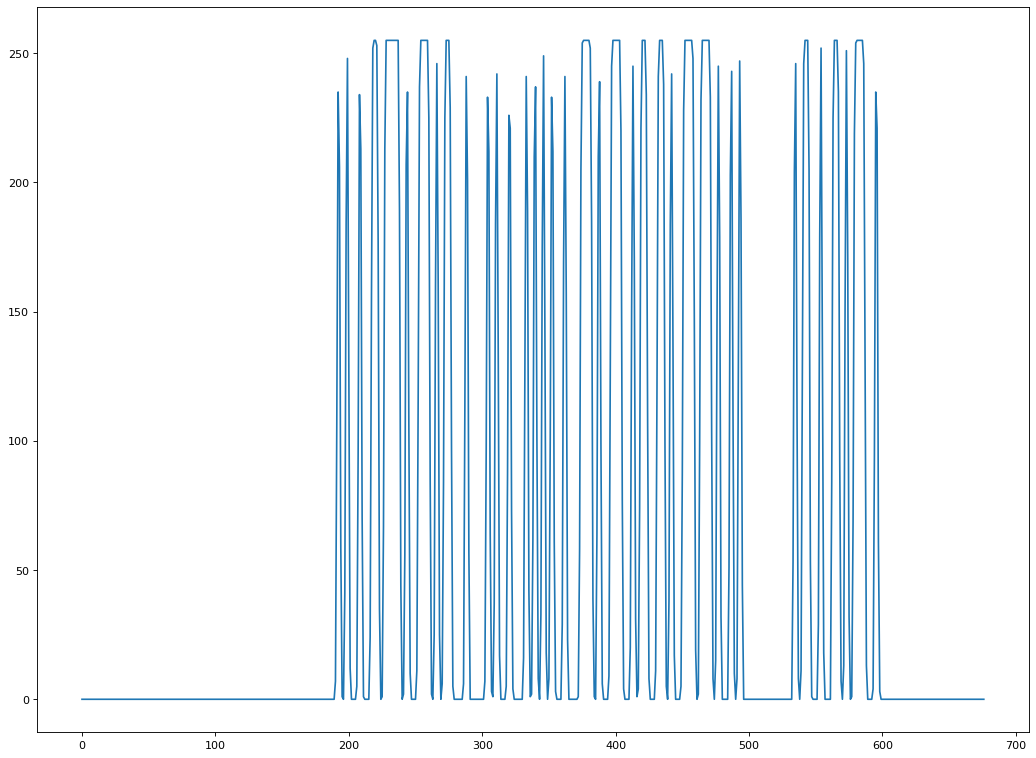
# Обычный штрих код  
if use\_graph:  
 figure(figsize=(32, 16), dpi=80)  
 plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))  
 plt.xticks([]),plt.yticks([])  
 plt.show()



png

# Для вычисления профиля буквально посчитаю интенсивности вдоль одной центральной линии.  
@njit(parallel=export\_parallel)  
def improfile(image, profile\_line\_raw=None, profile\_length=None):  
 if profile\_line\_raw is None:  
 profile\_line = np.array([image.shape[0] / 2.0, 0.0, image.shape[0] / 2.0, 1.0 \* image.shape[1]], dtype="float64")  
 else:  
 profile\_line = np.array(profile\_line\_raw, dtype="float64")  
 if profile\_length is None:  
 profile\_length = sqrt(pow(profile\_line[2] - profile\_line[0], 2.0) + pow(profile\_line[3] - profile\_line[1], 2.0))  
 image -= np.min(image)  
 max\_intensity = np.max(image)  
 intensity = np.zeros((ceil(profile\_length), ))  
 initial\_point = profile\_line[0:2]  
 vector = (profile\_line[2:4] - initial\_point)  
 for t in range(ceil(profile\_length)):  
 current\_point = initial\_point + vector \* (1.0 \* t / profile\_length)  
 if current\_point[0] > image.shape[0] - 1 or current\_point[1] > image.shape[1] - 1 or current\_point[0] < 0 or current\_point[1] < 0:  
 continue  
 for x in range(floor(current\_point[1]), floor(current\_point[1]) + 2):  
 for y in range(floor(current\_point[0]), floor(current\_point[0]) + 2):  
 multiplier = (1 - abs(x - current\_point[1])) \* (1 - abs(y - current\_point[0]))  
 intensity[t] += (max\_intensity - image[y][x]) \* multiplier  
 return intensity

# Выведу профиль изображения вдоль центральной линии.  
if use\_graph:  
 figure(figsize=(16, 12), dpi=80)  
 plt.plot(improfile(image))



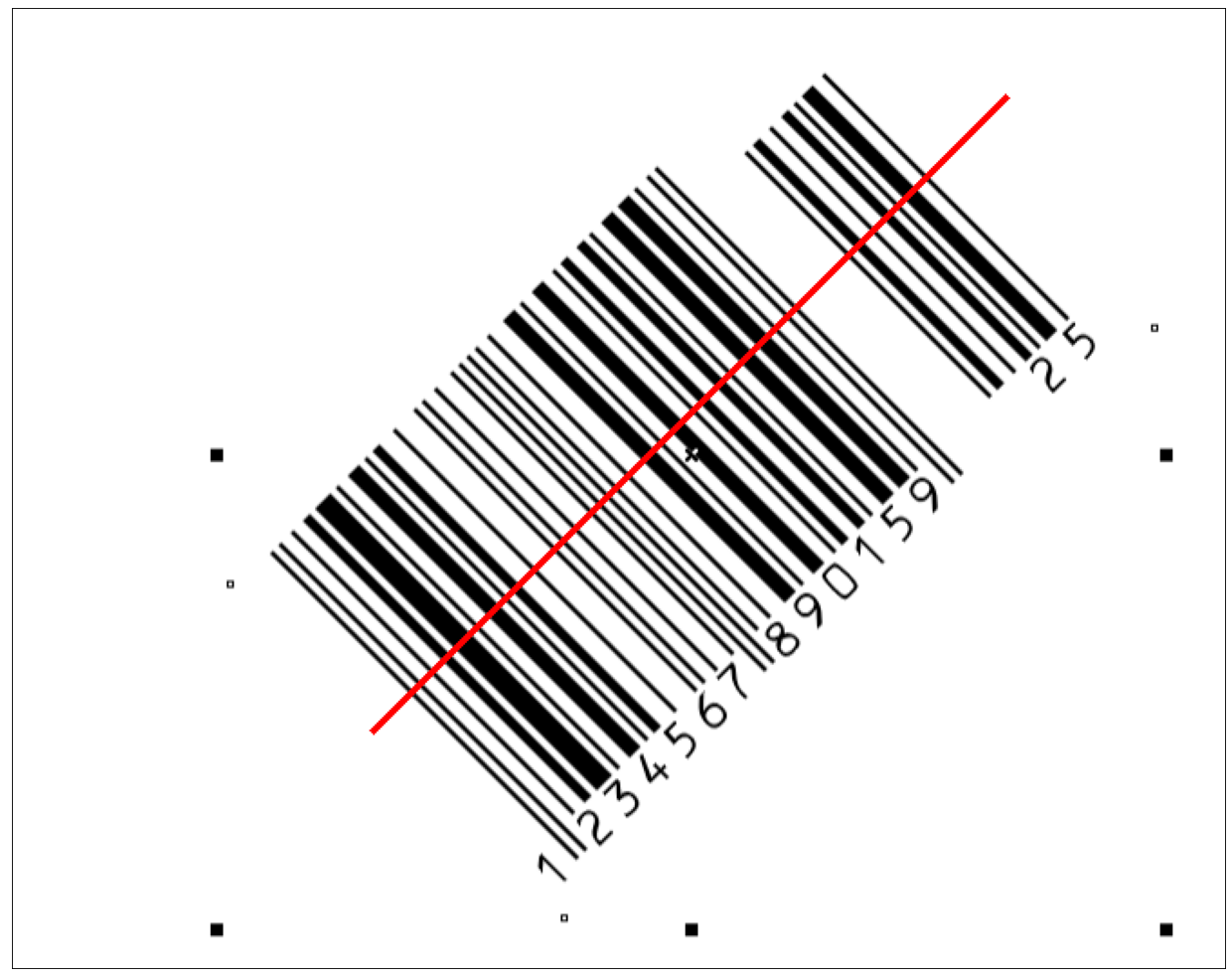
png

Полученные данные можно использовать для считывания самого штрих кода, но пока что сам считыватель получается не очень умным, так как не реагирует на повороты.

### Листинг 2.2. Интерактивное задание линии профиля.

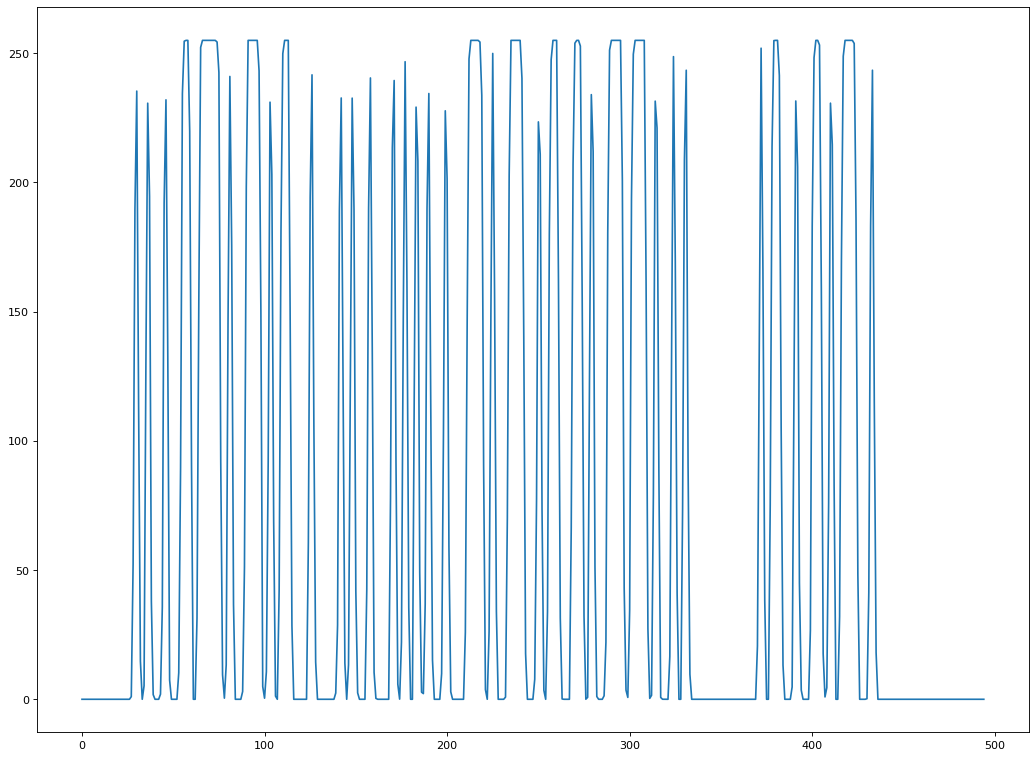
# Считаю повёрнутый штрих код в чёрно-белом формате.  
image = cv2.cvtColor(cv2.imread("data/barcode\_rotation.png"), cv2.COLOR\_BGR2GRAY)

# Интерактивное задание линии профиля.  
line = [400, 200, 50, 550] # y0, x0, y1, x1  
  
# Повёрнутый штрих код  
if use\_graph:  
 figure(figsize=(32, 16), dpi=80)  
 image\_draw = cv2.imread("data/barcode\_rotation.png")  
 line\_draw = line[::-1]  
 image\_draw = cv2.line(image\_draw, tuple(line\_draw[0:2]), tuple(line\_draw[2:4]), color=(0, 0, 255), thickness=3)  
 plt.imshow(cv2.cvtColor(image\_draw, cv2.COLOR\_BGR2RGB))  
 plt.xticks([]),plt.yticks([])  
 plt.show()



png

# Выведу повёрнутый профиль изображения вдоль заданной линии.  
if use\_graph:  
 figure(figsize=(16, 12), dpi=80)  
 plt.plot(improfile(image, line))



png

Можно считывать штрих код даже в повернутом виде. Если вычислить местоположение штрих кода на изображении, можно даже автоматизировать этот процесс.

### Листинг 2.3. Определение положения текста

**Проекция изображения**

Проекцией изображения на некоторую ось называется суммаинтенсивностей пикселей изображения в направлении, перпендикулярном данной оси.

Простейшим случаем проекции двумерного изображения являются вертикальная проекция на ось 𝑂𝑥, представляющая собой сумму интенсивностей пикселей по столбцам изображения:

Proj 𝑋(x) = sum(𝐼(𝑥,𝑦) for y in range (0, len(𝑌)))

и горизонтальная проекция на ось 𝑂𝑦, представляющая собой сумму интенсивностей пикселей по строкам изображения:

Proj Y(y) = sum(𝐼(𝑥,𝑦) for x in range (0, len(X)))

Запишем выражение для проекции на произвольную ось. Допустим, что направление оси задано единичным вектором с координатами (𝑒𝑥,𝑒𝑦). Тогда проекция изображения на ось 𝑂𝑒 определяется следующим выражением:

Proj E(t) = sum(𝐼(𝑥,𝑦) for all i where x = ex \* t + ey \* i + and y = ey \* t + ex \* i)

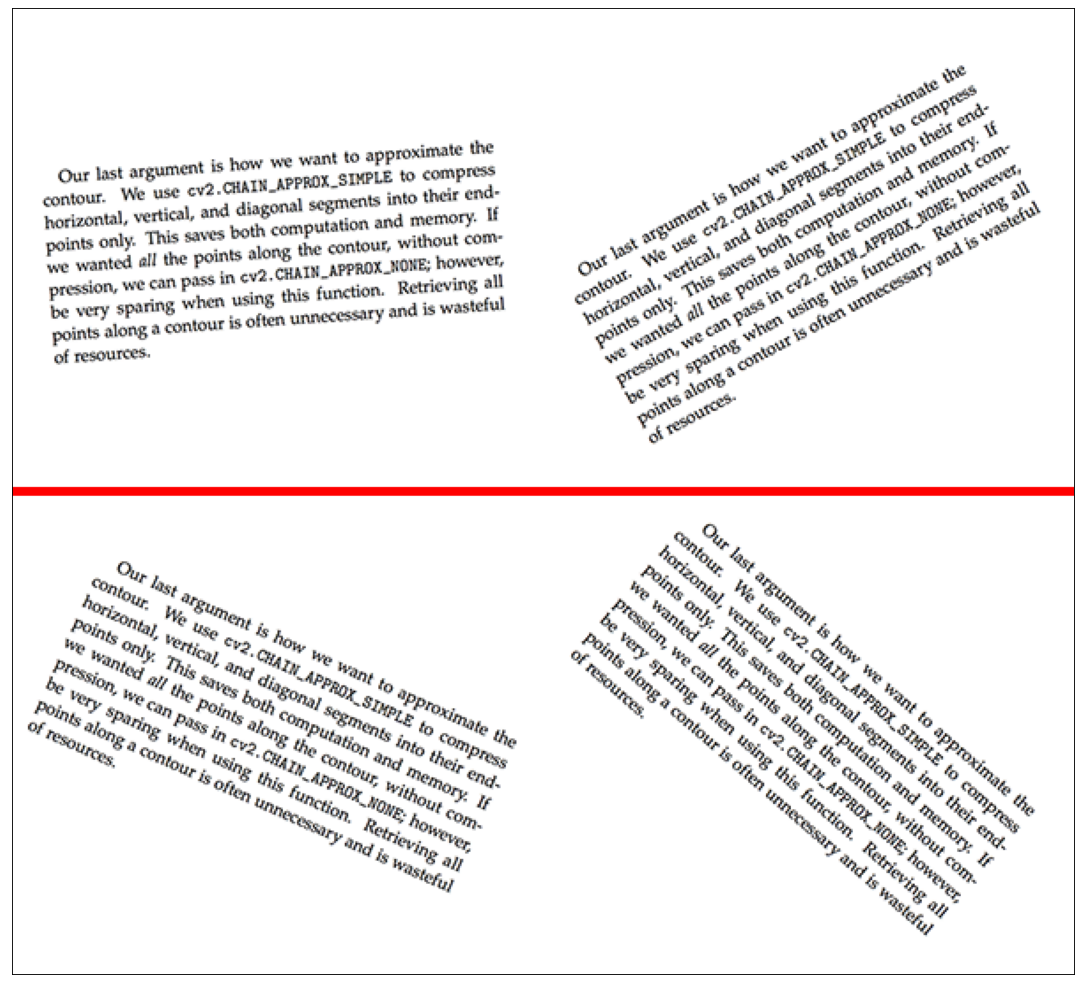
Анализ массива проекции позволяет выделять характерные точки функции проекции, которые соответствуют контурам объектов на изображении. Например, если на изображении имеются контрастные объекты, то в проекции будут видны перепады или экстремумы функции, соответствующие положению каждого из объектов.

Подобные проекции могут быть использованы в алгоритмах обнаружения и сегментации текстовых строк в системах распознавания текста.

def calc\_projection\_X(image):  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 (thresh, image) = cv2.threshold(image, 128, 255, cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)  
 image -= np.min(image)  
 max\_intensity = np.max(image)  
 vertical\_pixel\_sum = np.sum(max\_intensity - image, axis=0)  
 return vertical\_pixel\_sum  
  
def calc\_angle\_projection(image, angle):  
 image = 255 - image  
 image\_rotated = imutils.rotate\_bound(image, angle)  
 image\_rotated = 255 - image\_rotated  
 return calc\_projection\_X(image\_rotated)

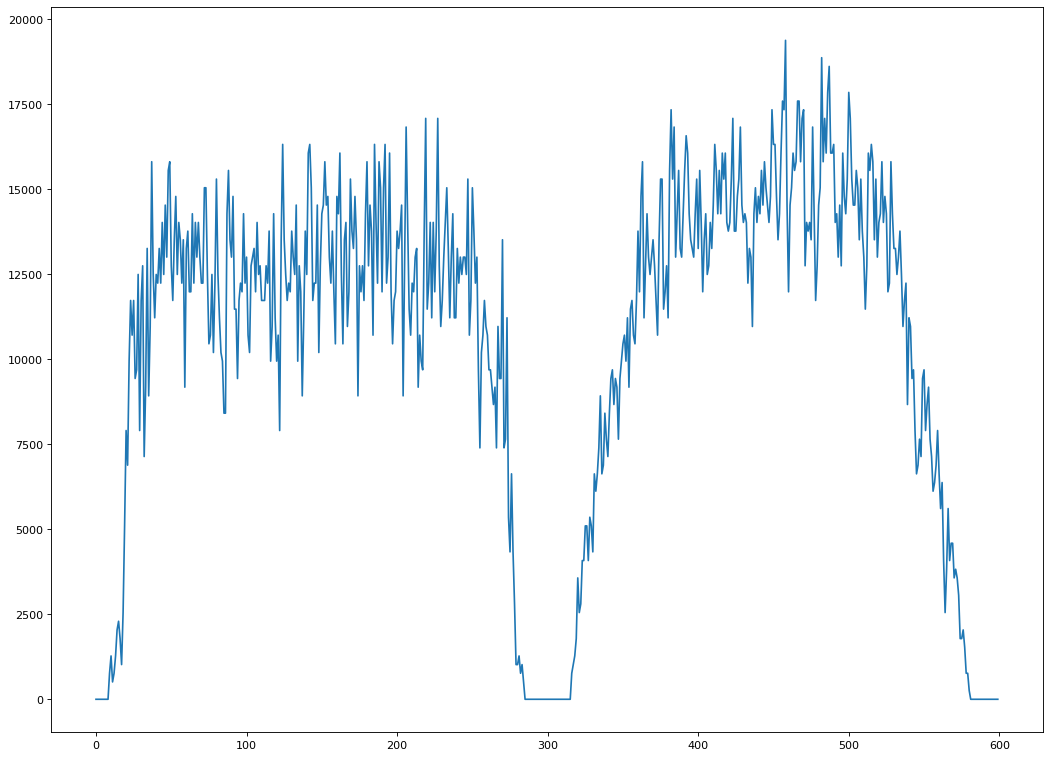
# Считаю текст  
image = cv2.imread("data/text\_skew\_inputs.png")

# Вычислю проекцию на ось Ox  
angle\_max = 0  
if use\_graph:  
 figure(figsize=(32, 16), dpi=80)  
 image\_draw = image.copy()  
 line\_len = norm(np.array(image.shape)) / 2.0  
 line = [image.shape[1] / 2.0 - cos(angle\_max \* pi / 180) \* line\_len,  
 image.shape[0] / 2.0 - sin(-angle\_max \* pi / 180) \* line\_len,   
 image.shape[1] / 2.0 + cos(angle\_max \* pi / 180) \* line\_len,  
 image.shape[0] / 2.0 + sin(-angle\_max \* pi / 180) \* line\_len]  
 line = np.array(line, dtype="int64")  
 image\_draw = cv2.line(image\_draw, tuple(line[0:2]), tuple(line[2:4]), color=(0, 0, 255), thickness=3)  
 plt.imshow(cv2.cvtColor(image\_draw, cv2.COLOR\_BGR2RGB))  
 plt.xticks([]),plt.yticks([])  
 plt.show()



png

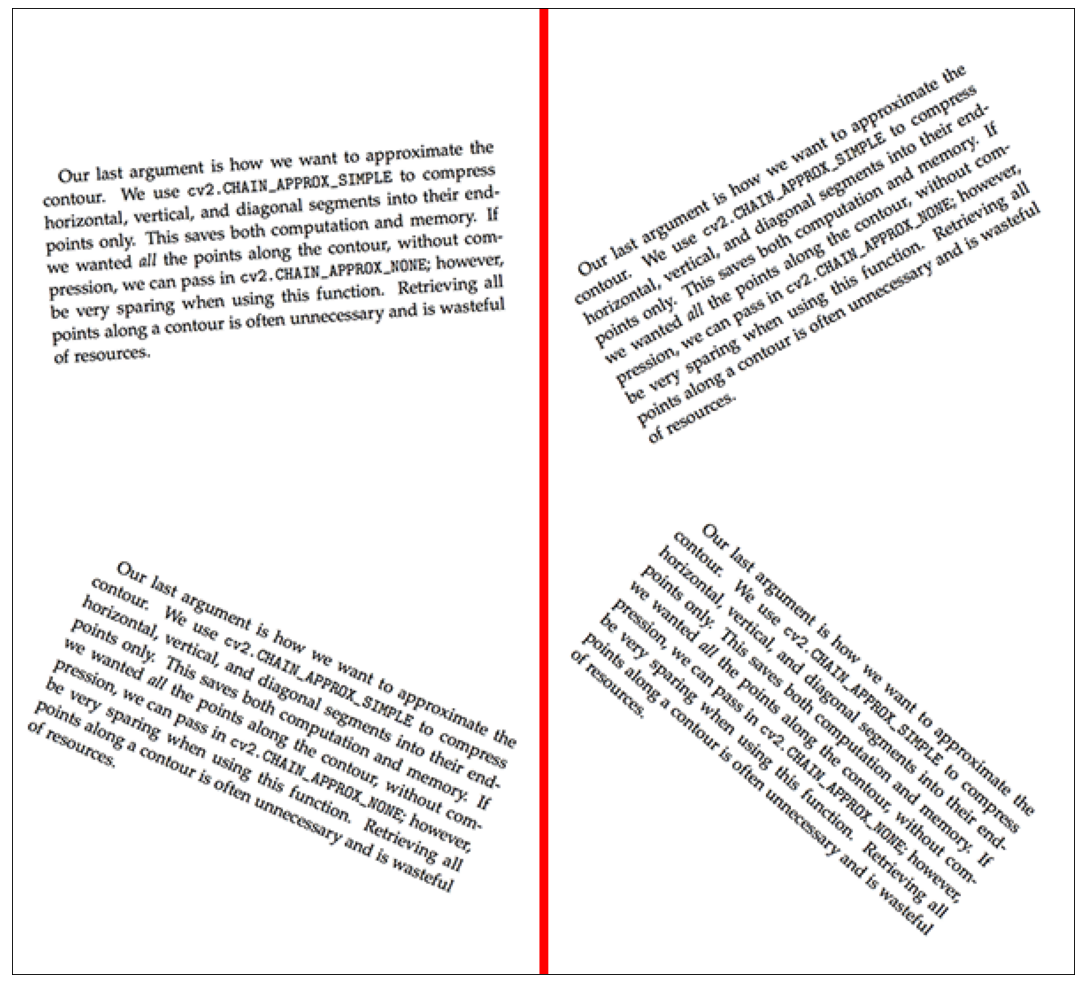
if use\_graph:  
 figure(figsize=(16, 12), dpi=80)  
 plt.plot(calc\_angle\_projection(image, angle\_max))



png

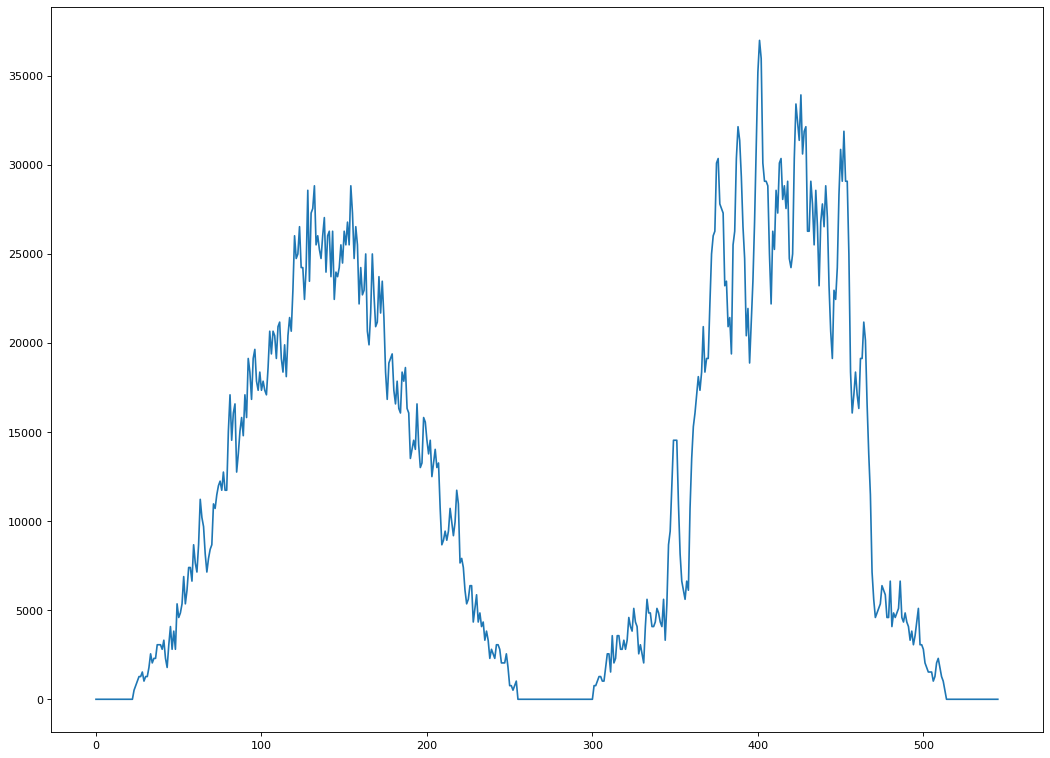
# Считаю текст  
image = cv2.imread("data/text\_skew\_inputs.png")

# Вычислю проекцию на ось Oy  
angle\_max = 90  
if use\_graph:  
 figure(figsize=(32, 16), dpi=80)  
 image\_draw = image.copy()  
 line\_len = norm(np.array(image.shape)) / 2.0  
 line = [image.shape[1] / 2.0 - cos(angle\_max \* pi / 180) \* line\_len,  
 image.shape[0] / 2.0 - sin(-angle\_max \* pi / 180) \* line\_len,   
 image.shape[1] / 2.0 + cos(angle\_max \* pi / 180) \* line\_len,  
 image.shape[0] / 2.0 + sin(-angle\_max \* pi / 180) \* line\_len]  
 line = np.array(line, dtype="int64")  
 image\_draw = cv2.line(image\_draw, tuple(line[0:2]), tuple(line[2:4]), color=(0, 0, 255), thickness=3)  
 plt.imshow(cv2.cvtColor(image\_draw, cv2.COLOR\_BGR2RGB))  
 plt.xticks([]),plt.yticks([])  
 plt.show()



png

if use\_graph:  
 figure(figsize=(16, 12), dpi=80)  
 plt.plot(calc\_angle\_projection(image, angle\_max))

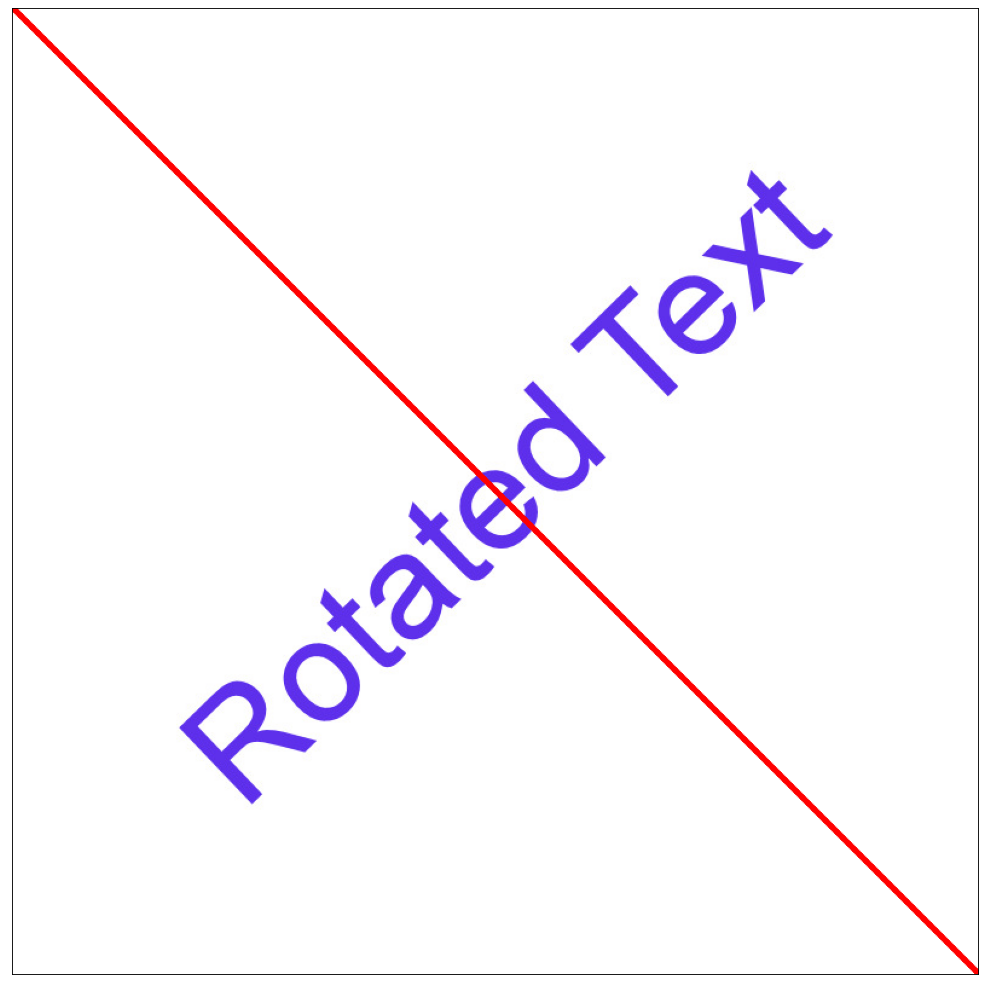


png

Из преокции можно сделать вывод, что текста 4, и они находятся по 4 стороны от изображения.

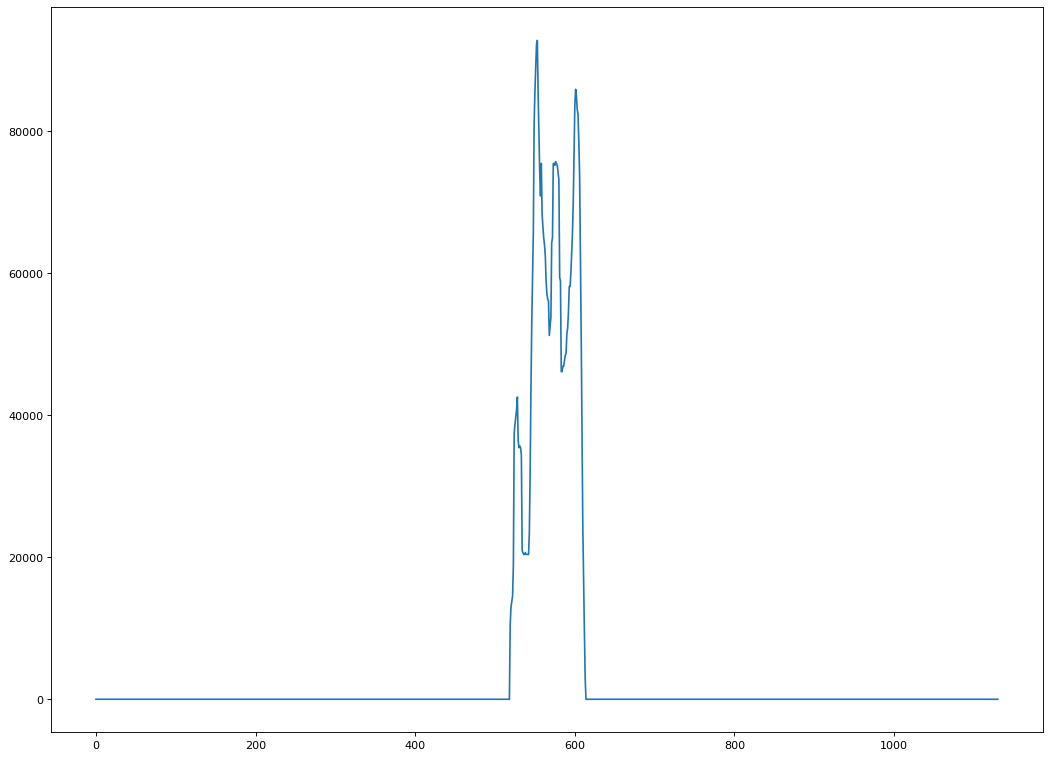
# Считаю повёрнутый текст  
image = cv2.imread("data/rotatedtext.jpg")

# Вычислю проекцию на ось, перпендикулярную оси текста  
angle\_max = -45  
if use\_graph:  
 figure(figsize=(32, 16), dpi=80)  
 image\_draw = image.copy()  
 line\_len = norm(np.array(image.shape)) / 2.0  
 line = [image.shape[1] / 2.0 - cos(angle\_max \* pi / 180) \* line\_len,  
 image.shape[0] / 2.0 - sin(-angle\_max \* pi / 180) \* line\_len,   
 image.shape[1] / 2.0 + cos(angle\_max \* pi / 180) \* line\_len,  
 image.shape[0] / 2.0 + sin(-angle\_max \* pi / 180) \* line\_len]  
 line = np.array(line, dtype="int64")  
 image\_draw = cv2.line(image\_draw, tuple(line[0:2]), tuple(line[2:4]), color=(0, 0, 255), thickness=3)  
 plt.imshow(cv2.cvtColor(image\_draw, cv2.COLOR\_BGR2RGB))  
 plt.xticks([]),plt.yticks([])  
 plt.show()



png

if use\_graph:  
 figure(figsize=(16, 12), dpi=80)  
 plt.plot(calc\_angle\_projection(image, angle\_max))



png

По единичному пику можно сделать вывод о местоположении текста.