Рассмотрим функцию с девятью значениями

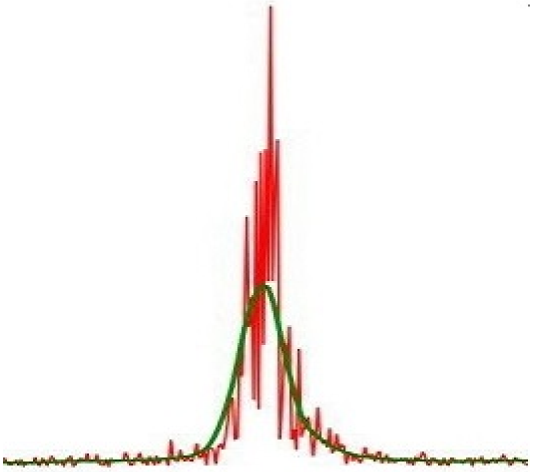
$$Y = y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9$$

Будем аппроксимировать её функцией W с помощью следующего отображения:

$$F : Y \to W : w[i] = C_1 y_i + C_2 y_{i+1} + C_3 y_{i+2}$$

с обрезанием, т.е. последние два элемента, которые нельзя корректно вычислить, будем удалять. Т.е. $|W| = 7$
Обозначим суммы:

$$S_y = \sum_{i=1}^{7} y_i,$$
$$S_w = \sum_{i=1}^{7} w_i$$



Для наглядности представьте, что Y - красненькая, а W - зелененькая. Тогда, если значения функции нанесены с шагом 1, то суммы по сути и будут интегралами соответствующих функций. Зададимся вопросом, какому условию должны удовлетворять аппроксимирующие коэффициенты C_1, C_2, C_3, чтобы преобразование сохраняло значение интеграла?

В соответствии с определением F, имеем:

$$S_w = w_1 + \ldots + w_7 = (C_1 y_1 + C_2 y_2 + C_3 y_3) + (C_1 y_2 + C_2 y_3 + C_3 y_4) +$$
$$+(C_1 y_3 + C_2 y_4 + C_3 y_5) + (C_1 y_4 + C_2 y_5 + C_3 y_6) + (C_1 y_5 + C_2 y_6 + C_3 y_7) +$$
$$+(C_1 y_6 + C_2 y_7 + C_3 y_8) + (C_1 y_7 + C_2 y_8 + C_3 y_9) =$$

перегруппируем члены:

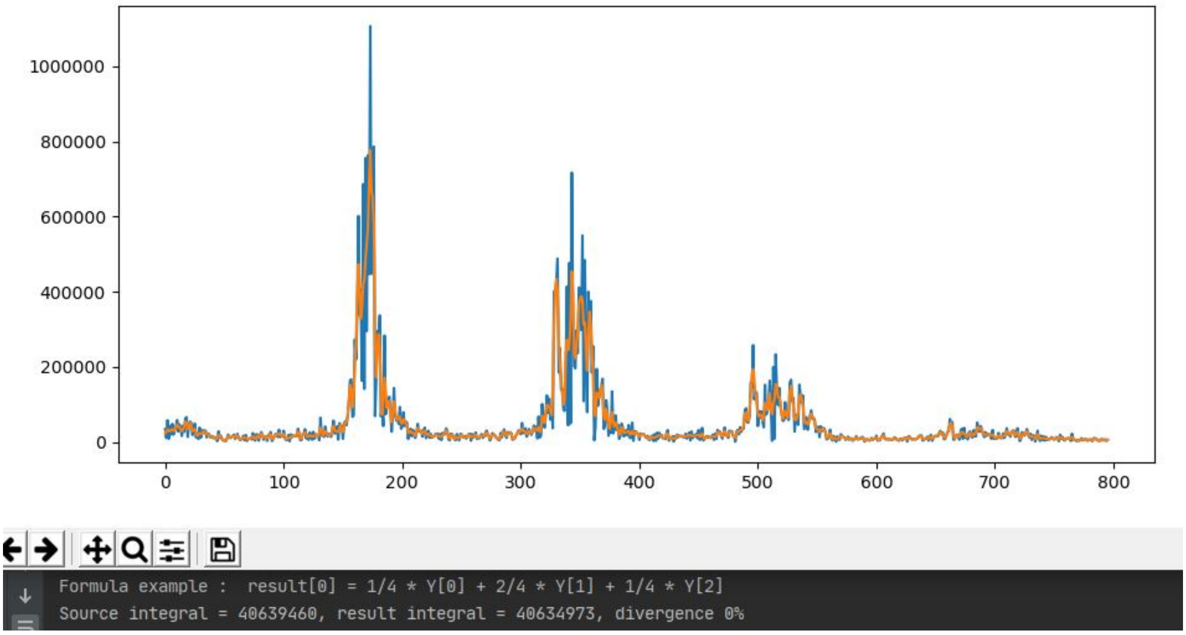
$$= C_1 y_1 + (C_1 + C_2) y_2 + (C_1 + C_2 + C_3) y_3 + (C_1 + C_2 + C_3) y_4 +$$
$$+(C_1 + C_2 + C_3) y_5 + (C_1 + C_2 + C_3) y_6 + (C_1 + C_2 + C_3) y_7 + (C_2 + C_3) y_8 + C_3 y_9$$

Заметим далее, что при $(C_1 + C_2 + C_3) = 1$ часть слагаемых $(y_3, \ldots, y_7)$ в S_y и S_w совпадает, в итоге:

$$S_y - (y_1 + y_2) = S_w - (C_1 y_1 + (C_1 + C_2) y_2 + (C_2 + C_3) y_8 + (C_2 + C3) y_9)$$

Нетрудно видеть, что при большой выборке (у нас она обычно 600-800 значений на 3-4 пика) интегралы исходной функции и её аппроксимации можно считать одинаковыми. В среднем, они будут отличаться на значение суммы нескольких точек. Т.е. эквивалентность единице суммы выбранных коэффициентов это, по сути, единственное условие, которое достаточно для этого выдержать.

Например, аппроксимация наших данных при $C_1 = \frac{1}{4}, C_2 = \frac{1}{2}$ и $C_3 = \frac{1}{4}$ для исходного $S_y = 40639460$ даёт $S_w = 40634973$



Также нетрудно заметить, что на таких объемах выборок по сути никакой разницы между

$$F : Y \to W : w[i] = C_1 y_i + C_2 y_{i+1} + C_3 y_{i+2}$$

и, например,

$$F : Y \to W : w[i] = C_1 y_{i-1} + C_2 y_i + C_3 y_{i+1}$$

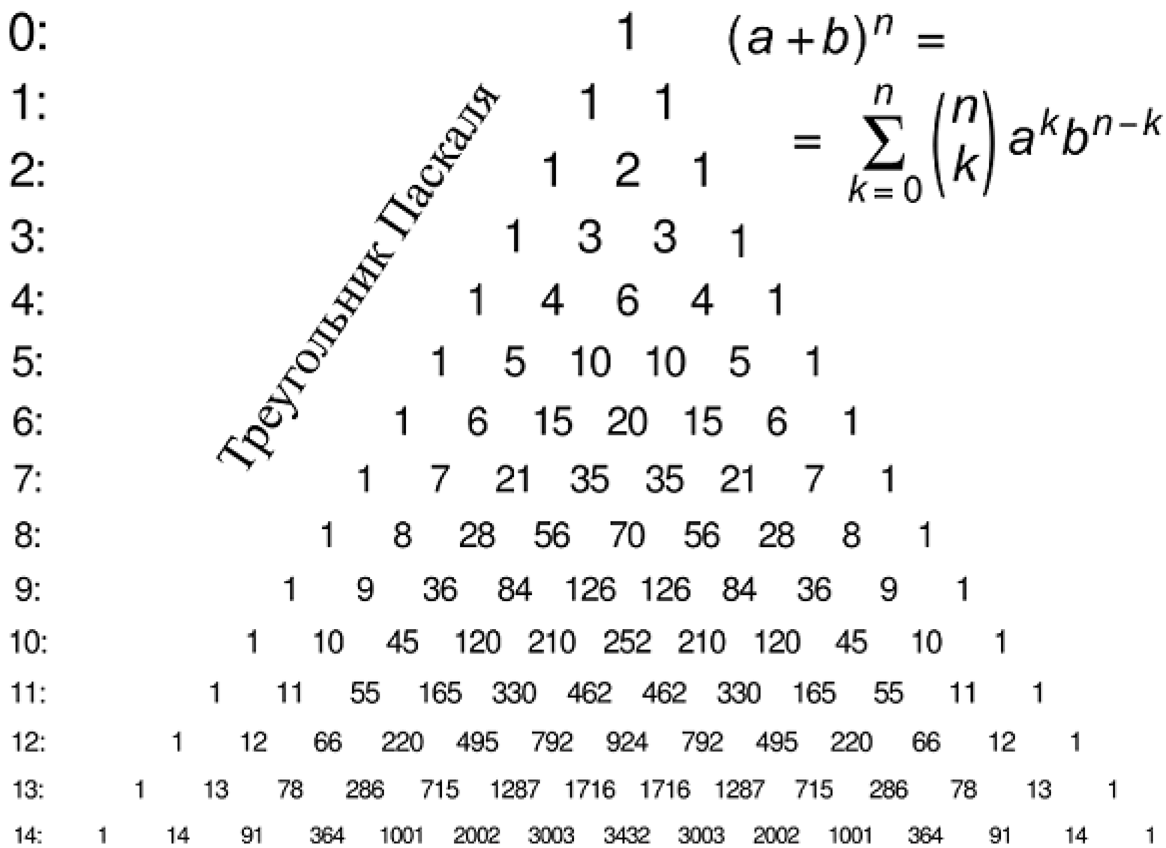
нет.

Глядя на последнюю картинку также можно заметить, что для наших данных с постоянно чередующимися рядом пиками и провалами (так называемые "горбы") аппроксимация по трем точкам явно недостаточна (этой первый вопрос - сколько точек будет достаточно?). Также вопрос состоит в том, каким образом подбирать коэффициенты? (это второй вопрос).

В аппроксимации цифровых сигналов, где картина в чём-то сравнима с нашей, склоняются к тому, что значимость коэффициентов должна убывать от центральной точки в обе стороны, т.е. $C_1 = \frac{1}{4}, C_2 = \frac{1}{2}$ и $C_3 = \frac{1}{4}$ лучше чем $C_1 = \frac{1}{3}, C_2 = \frac{1}{3}$ и $C_3 = \frac{1}{3}$. Это позволяет не утюжить холмы почем зря (большее значение получает больший коэффициент). Вместе с тем, пики у нас достаточно острые и есть предположение, что слишком сильное усиление коэффициентами не позволит их сгладить ни по трем точкам, ни по тридцати трем.

Ниже проведем ряд экспериментов и найдем золотую середину (далее я буду формулы брать из вывода своей программы, чтоб не тратить время на их ручное перерисовывание).

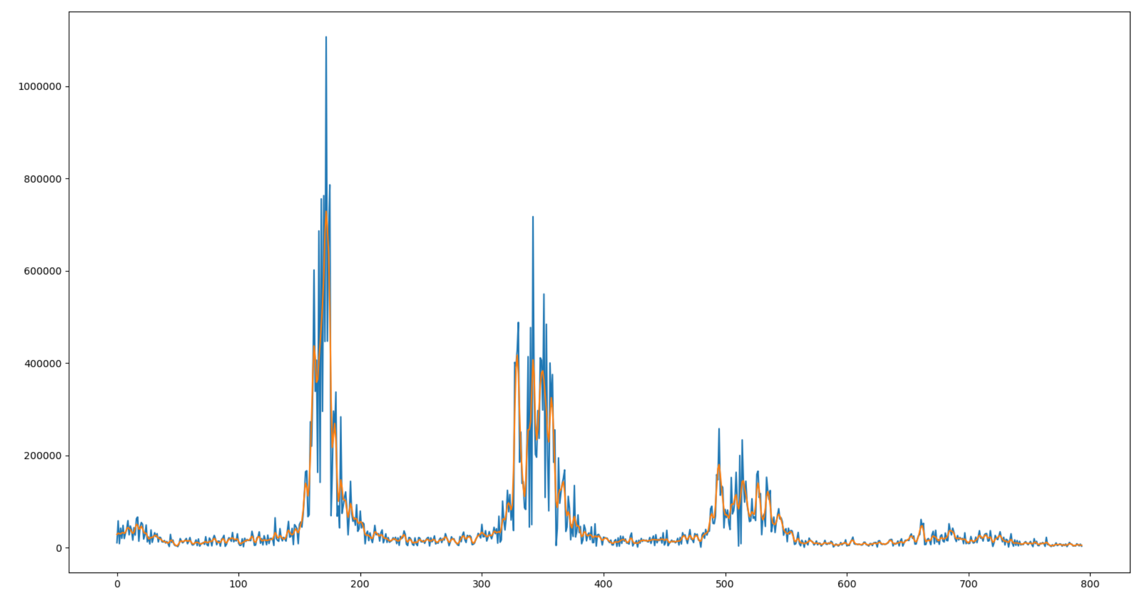
Для варианта с сильным усилением центральных коэффициентов, решил взять коэффициенты из треугольника Паскаля:



Например, для аппроксимации по 5-ти точкам коэффициенты берутся из 4-й строки треугольника:

```
[1, 4, 6, 4, 1]

Formula example :  result[0] = 1/16 * Y[0] + 4/16 * Y[1] + 6/16 * Y[2] + 4/16 * Y[3] + 1/16 * Y[4]

Source integral = 40599657, result integral = 40605265, divergence 0%
```

результат:

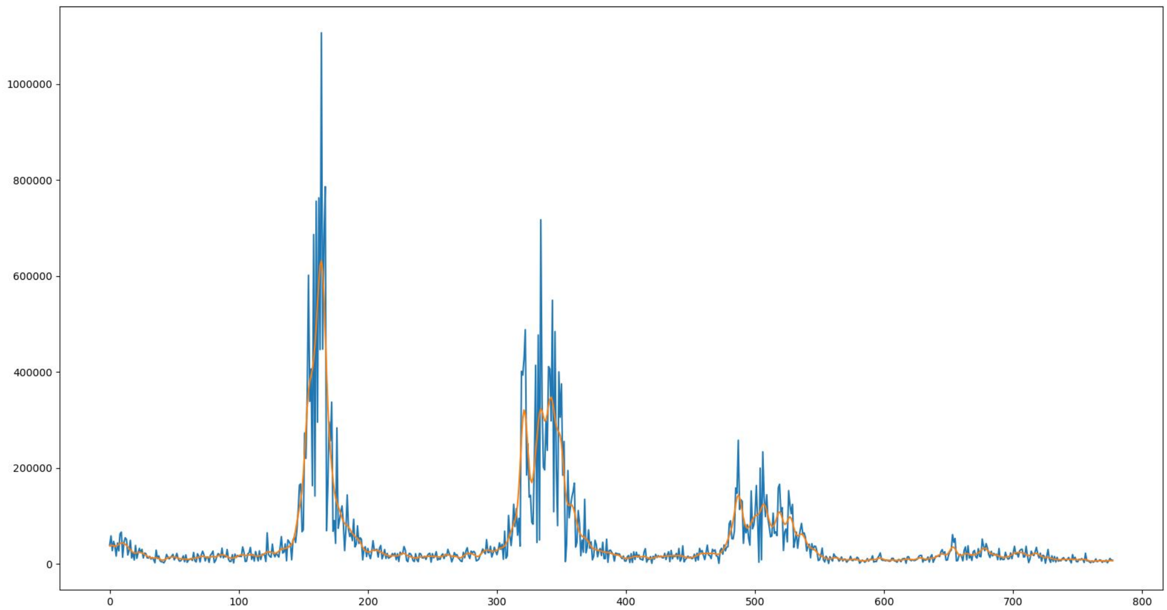


для аппроксимации по 11-ти точкам из 10-й строки:

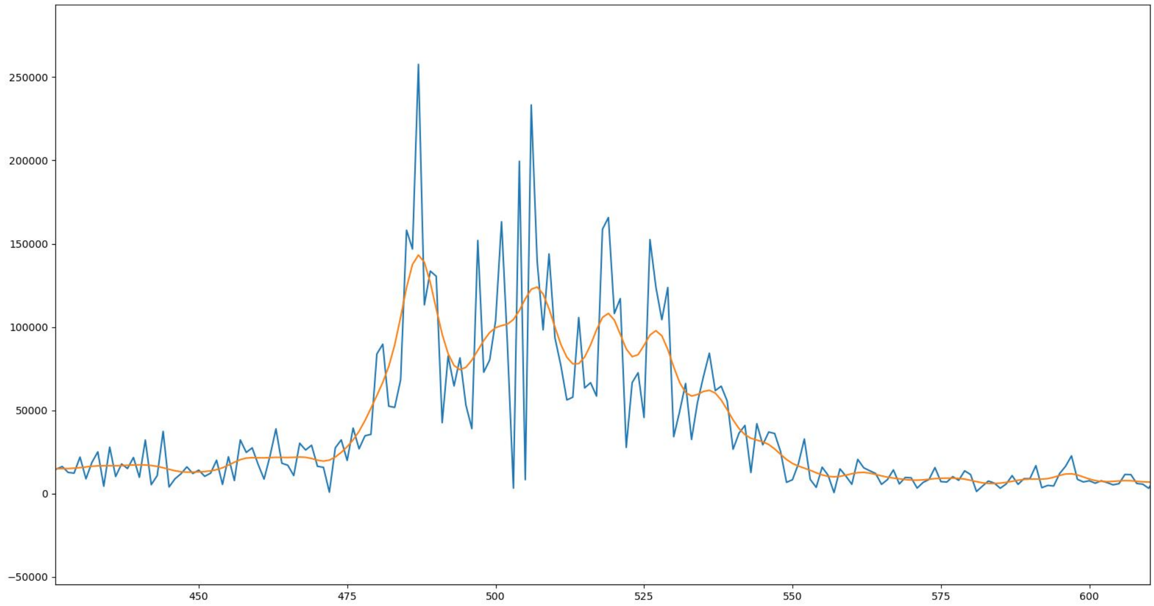
```
[1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]

Formula example :  result[0] = 1/1024 * Y[0] + 10/1024 * Y[1] + 45/1024 * Y[2] +
120/1024 * Y[3] + 210/1024 * Y[4] + 252/1024 * Y[5] + 210/1024 * Y[6] + 120/1024
* Y[7] + 45/1024 * Y[8] + 10/1024 * Y[9] + 1/1024 * Y[10]

Source integral = 40509389, result integral = 40500067, divergence 0%
```

Вот паскаль по 21-й точке

```
[1, 20, 190, 1140, 4845, 15504, 38760, 77520, 125970, 167960, 184756, 167960,
125970, 77520, 38760, 15504, 4845, 1140, 190, 20, 1]

Formula example :  result[0] = 1/1048576 * Y[0] + 20/1048576 * Y[1] +
190/1048576 * Y[2] + 1140/1048576 * Y[3] + 4845/1048576 * Y[4] + 15504/1048576 *
Y[5] + 38760/1048576 * Y[6] + 77520/1048576 * Y[7] + 125970/1048576 * Y[8] +
167960/1048576 * Y[9] + 184756/1048576 * Y[10] + 167960/1048576 * Y[11] +
125970/1048576 * Y[12] + 77520/1048576 * Y[13] + 38760/1048576 * Y[14] +
15504/1048576 * Y[15] + 4845/1048576 * Y[16] + 1140/1048576 * Y[17] +
190/1048576 * Y[18] + 20/1048576 * Y[19] + 1/1048576 * Y[20]

Source integral = 40323883, result integral = 40308500, divergence 0%
```



третий пик покрупнее:



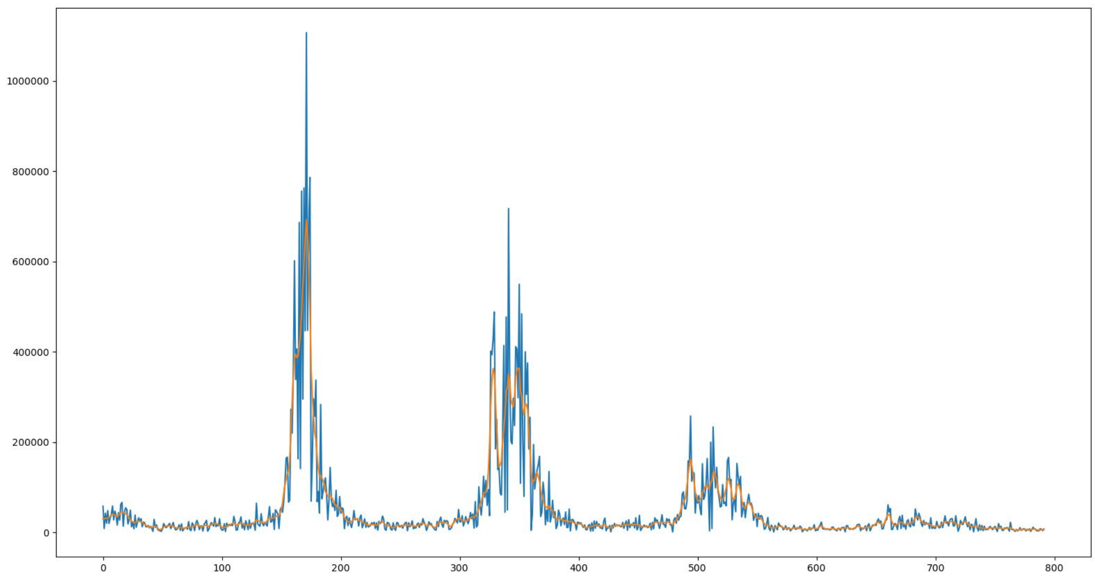
как видно, такой сильный акцент на усиление коэффициентов не даёт желаемого сглаживания, хотя интерполирует сам характер кривой прекрасно, да и с интегралами всё в порядке.

Поигравшись таким образом с разными вариантами я остановился на самом простом методе, в котором числитель коэффициентов плавно уменьшается от центрального коэффициента в обе стороны на единицу при одинаковом знаменателе, а число точек аппроксимации нечётное. Знаменатель при этом легко вычисляется как квадрат от числа половины точек аппроксимации, округленной в большую сторону.

т.е. для трёх точек это $C_1 = \frac{1}{4}, C_2 = \frac{2}{4} = \frac{1}{2}$ и $C_3 = \frac{1}{4}$,

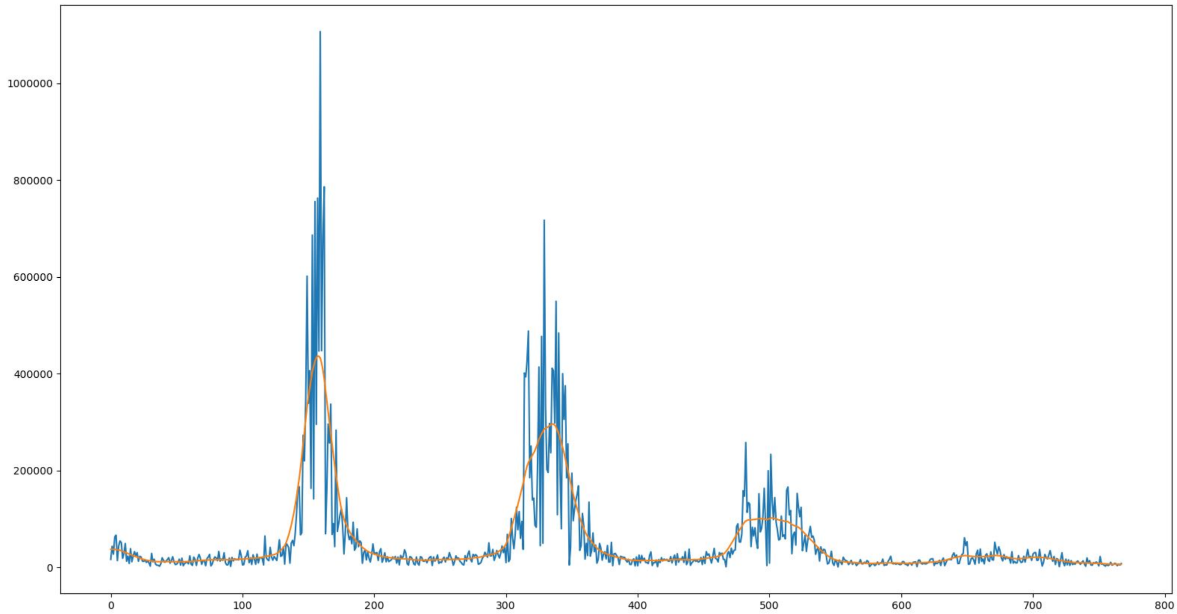
для семи $C_1 = \frac{1}{16}, C_2 = \frac{2}{16}, C_3 = \frac{3}{16}, C_4 = \frac{4}{16}, C_5 = \frac{3}{16}, C_6 = \frac{2}{16}$ и $C_7 = \frac{1}{16}$

Вот как работает на предыдущих данных этот метод на семи точках:



на 31-й он уже даёт приемлемый результат:

```
Formula example :  result[0] = 1/256 * Y[0] + 2/256 * Y[1] + 3/256 * Y[2] +
4/256 * Y[3] + 5/256 * Y[4] + 6/256 * Y[5] + 7/256 * Y[6] + 8/256 * Y[7] + 9/256
* Y[8] + 10/256 * Y[9] + 11/256 * Y[10] + 12/256 * Y[11] + 13/256 * Y[12] +
14/256 * Y[13] + 15/256 * Y[14] + 16/256 * Y[15] + 15/256 * Y[16] + 14/256 *
Y[17] + 13/256 * Y[18] + 12/256 * Y[19] + 11/256 * Y[20] + 10/256 * Y[21] +
9/256 * Y[22] + 8/256 * Y[23] + 7/256 * Y[24] + 6/256 * Y[25] + 5/256 * Y[26] +
4/256 * Y[27] + 3/256 * Y[28] + 2/256 * Y[29] + 1/256 * Y[30]

Source integral = 40079961, result integral = 40079927, divergence 0%
```



К этому моменту осталось решить вопрос - как определять автоматически параметры для сглаживающего метода (по какому количеству точек аппроксимировать), чтобы уже было хорошо, но еще не прямая линия.

Пожалуй, поясню вкратце как делал я прямо по листингу программы.

Основные функции в программе:

get_pascal_coefficients(row_number) - вычисляет паскалевские коэффициенты по введенной строке

sm_filter(source, smooth_size, method) - основная функция-фильтр - принимает на вход **source** - исходную выборку данных, **smooth_size**-ширину сглаживания (сколько точек брать), и **method**-какой метод использовать (тривиальный или паскалевский). Функция возвращает тройку значений: обрезанный на несколько точек исходный **source**, результирующий сглаженный **result** и в качестве сервиса возвращает вид формулы, по которой в итоге производится сглаживание **formula_example**

union(input_extremums, lower_cutoff, precision) - используется для объединения экстремумов "дрожащей" кривой. Принимает на вход **input_extremums** - все найденные для кривой экстремумы, **lower_cutoff** - абсолютное значение отсечения нижних экстремумов и **precision** - точность слияния.

Поясняю зачем все это мне понадобилось. Для того чтобы автоматически определить правильный **smooth_size** для фильтра **sm_filter**, нужно делать несколько прогонов этого фильтра, начиная с трех точек аппроксимации с шагом две точки до тех пор пока результат будет приемлемый. Интегралы у нас всегда прекрасные, на любом шаге. Поэтому следующий критерий - красота пиков. Которую я расшифровываю для себя как ожидание четких трех (возможно четырех) пиков. Посмотрим на предыдущую картинку. Во-первых, алгоритм, который ищет экстремумы (о нём будет ниже), он покажет не только холмики, но и впадинки между холмиками. Я решил с этим бороться просто их отсечением из рассмотрения. Из последней картинки видно, что для данной кривой линия отсечения будет примерно на уровне 30000-35000, собственно это значение и нужно задать в качестве **lower_cutoff**.

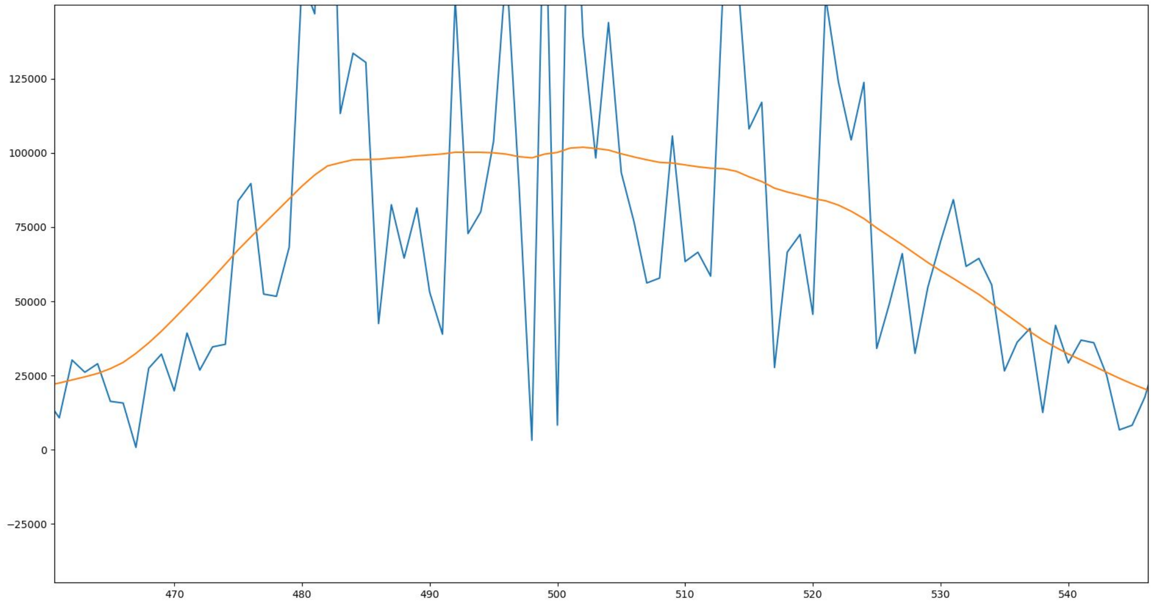
Следующая проблема - "дрожание" кривой. Как бы хорошо мы ни сгладили, экстремумов никогда не будет 3 или 4. Вот сколько их на самом деле для последней картинки:

```
Extremums =  [ 36875.95802295  37021.94146366  36972.84920363  36973.8228897
  36619.49282081  11591.17331774  11578.38278612  11320.46371685
  11269.25424982  15205.02474532  15146.35519232  15514.74936359
  15511.6928502   15532.73613893  15478.5449506  436598.36969867
 435718.82522536  16082.53277721  16069.05014026 296067.57663246
 295823.65312603  13887.49775071  13833.41850946  13858.90343195
  13711.24545277  16034.01441072  16028.90030172  15541.4715635
  15415.17021021 100203.09387793 100169.64789549 100184.86581912
 100023.56930372 101888.5721785  101494.9211035    9026.38692809
   8915.59327496  24000.49015755  23824.90456656  21931.23404931
  21842.93570638  24315.30050422  24088.1134164   20671.98377036
  20607.94385661]
```

Даже если мы отсечем некоторые из них линией отсечения, то останется достаточное количество. Если присмотреться, видно, что почти все из них - результаты статистического колебания кривой, ну например

```
100203.09387793 100169.64789549 100184.86581912 100023.56930372 101888.5721785
101494.9211035
```

это вот эти неровности третьего холма:



И очевидно, что при некоторых допущениях, можно считать эти несколько холмиков за один холм. Собственно функция **union** как раз и занимается тем, что сливает подобные неровности в один экстремум с максимальной высотой из представленного набора вариантов, а **precision** - это как раз допуск, в рамках которого значения будем считать одинаковыми. Для этого холма оно где-то в районе 2000.

В итоге, для этой картинки для указанного большого числа экстремумов функция вернула:

```
United extremums =  [436598.36969866656, 296067.5766324578, 101888.57217850279]
```

В дальнейшем этот факт (что было найдено три четких холма) и будет основным признаком останова алгоритма по подбору окончательной сглаживающей функции.

И, наконец, последняя функция, собирающая всё воедино и автоматизирующая подбор сглаживающей функции и её итоговый запуск:

smooth(sample, method, lower_cutoff, precision, step_limit, extremums_count_limit)

она возвращает полученные от **sm_filter** обрезанный на несколько точек исходный **source** и результирующий сглаженный **result**. Принимает на вход **sample** - исходную выборку данных, **method**-какой метод использовать (тривиальный или паскалевский), **lower_cutoff, precision**, которые прокидывает в метод **union**, ограничение сверху числа шагов подбора сглаживающей функции **step_limit** и **extremums_count_limit** - какое количество холмов мы в итоге планируем обнаружить (обычно 3 или 4)

Эта функция единственная, который использует функции pandas, поэтому я наверное поясню её прямо по тексту:

```
def smooth(sample, method, lower_cutoff, precision, step_limit,
extremums_count_limit):
    for step in range(1, step_limit + 1):
        source, result, formula_example = sm_filter(sample, step*2+1, method)
        print('Formula example : ', formula_example)
        print('Source integral = %d, result integral = %d, divergence %d%%' %
              (sum(source), sum(result), (100 - 100 / sum(source) *
sum(result))))
        df = pd.Series(result)
        grp =
df.groupby((np.sign(df.diff().fillna(0)).diff().fillna(0).ne(0)).cumsum())
        extremums = grp.apply(lambda x: x.max())
        print("Extremums = ", extremums.values)
        extremums = union(extremums.values, lower_cutoff, precision)
        print("United extremums = ", extremums)
        if (len(extremums) <= extremums_count_limit): break
    return source, result
```

Функция начинает в цикле подбирать число точек сглаживания. Для этого она на каждом шаге вызывает фильтр, передавая ему последовательно 3,5,7 и т.д. точек, пока её не устроит результат фильтрации. Массив сглаженных значений, возвращаемых фильтром, засовывается в панда-серию

```
df = pd.Series(result)
```

потом к серии применяется группировка **df.groupby** с рядом последовательных преобразований для того, чтобы получить группы - наборы монотонных значений. Дальше по каждой группе с помощью **grp.apply** ищется максимум. Если нужно будет транслировать в джаву, то целесообразно позапускать преобразования по отдельности, с выводом лога в файл как-нибудь так:

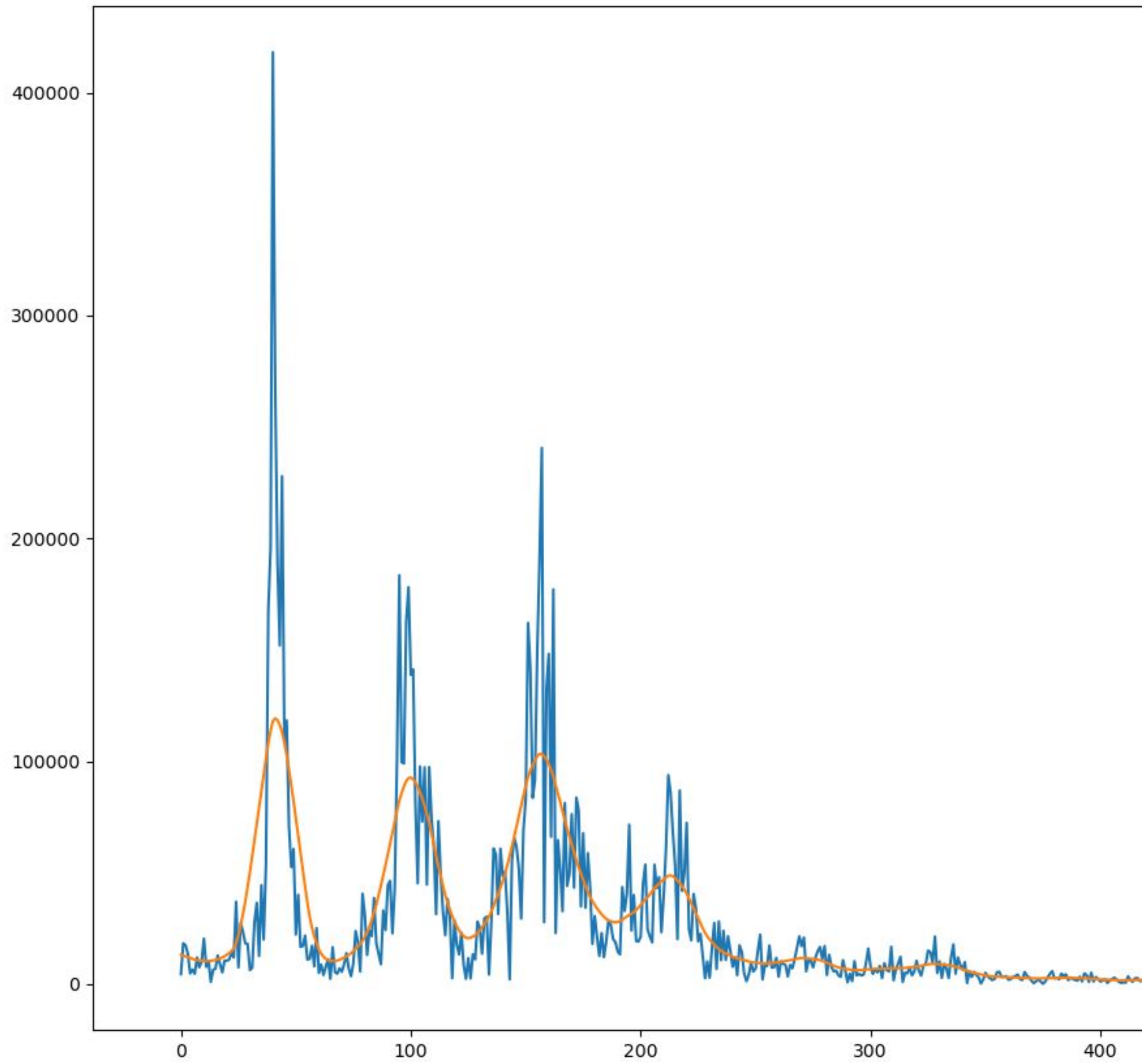
```
with open('result.txt', 'w') as file:
    print(*((np.sign(df.diff().fillna(0)).diff().fillna(0))), file=file,
sep="\n")
```

В итоге получаем набор всех экстремумов, которые потом пропускаем через наш метод **union** и, если в конце концов, получили **extremums_count_limit** холмиков либо достигли **step_limit**, то считаем что аппроксимировали достаточно и возвращаем результат.

Параметры **lower_cutoff** и **precision**, кажется, поддаются эвристике. По крайней мере для наших сэмплов хорошо себя показали lower_cutoff как 1/16 от самого большого значения в исходной выборке и precision как 1/512 от самого большого значения в исходной выборке. Пример запуска см. в коде, проект Idea прилагаю, но по сути от проекта нужны только pulse.py с исходными данными и установленные pandas, numpy и matplotlib.

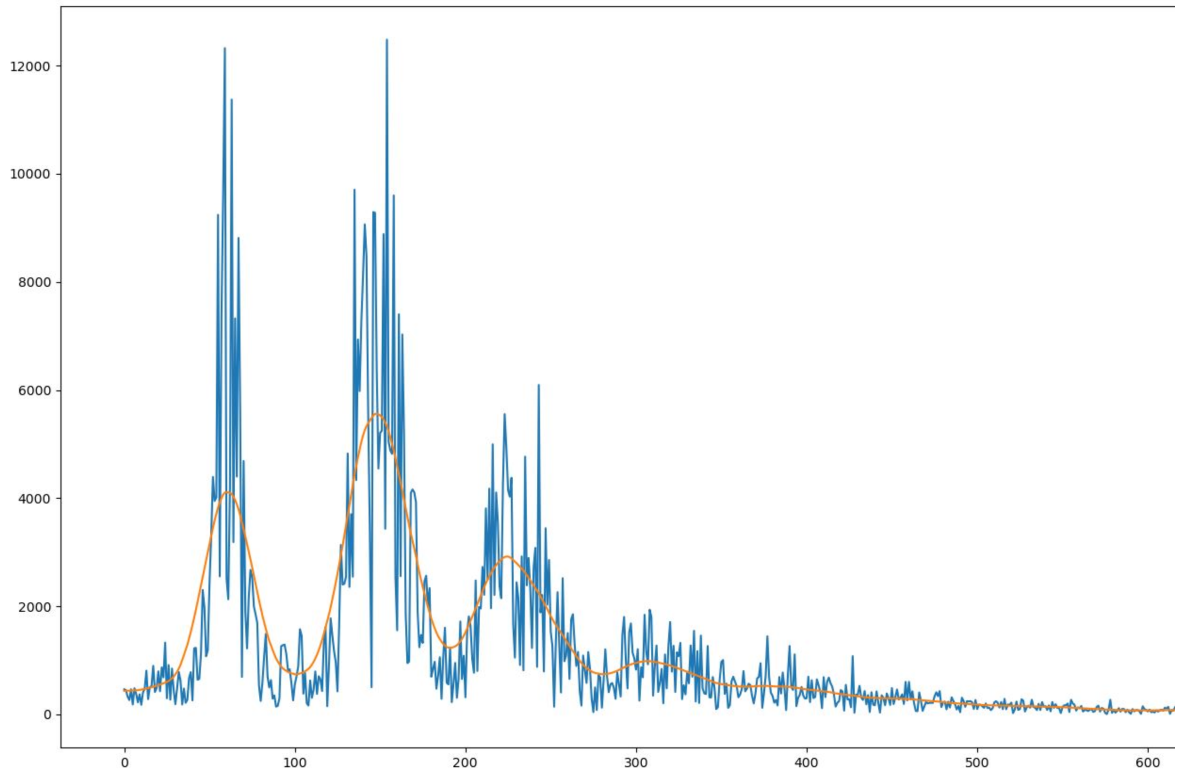
Ниже результаты работы программы по нашим сэмплам:

spectrum.csv



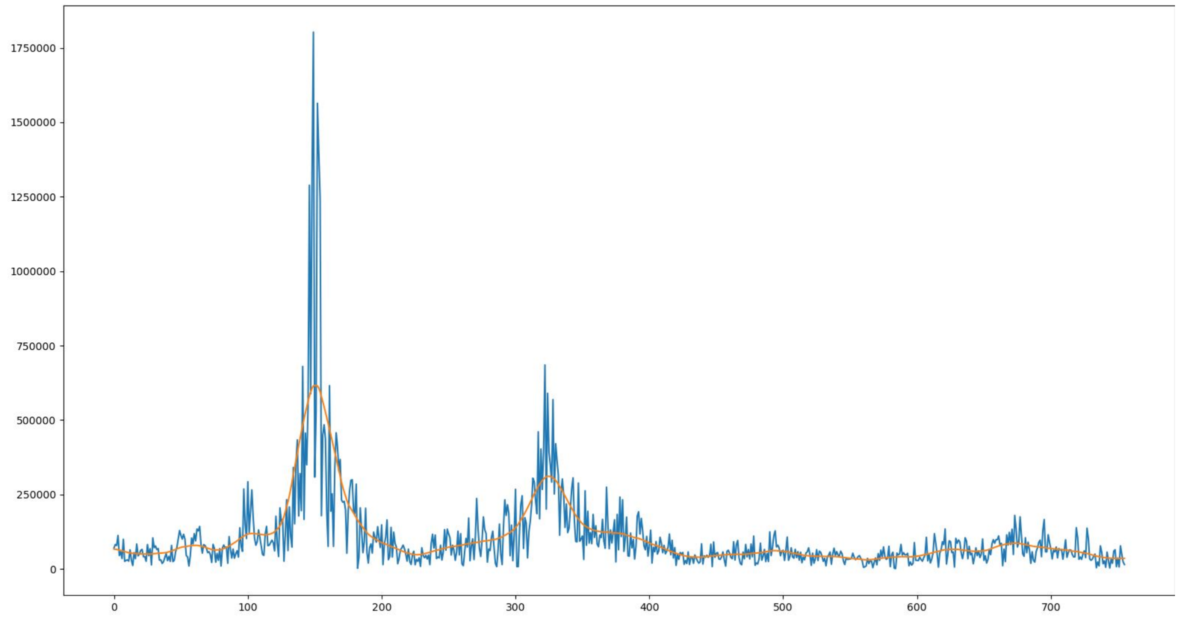
```
Formula example :  result[0] = 1/400 * Y[0] + 2/400 * Y[1] + 3/400 * Y[2] +
4/400 * Y[3] + 5/400 * Y[4] + 6/400 * Y[5] + 7/400 * Y[6] + 8/400 * Y[7] + 9/400
* Y[8] + 10/400 * Y[9] + 11/400 * Y[10] + 12/400 * Y[11] + 13/400 * Y[12] +
14/400 * Y[13] + 15/400 * Y[14] + 16/400 * Y[15] + 17/400 * Y[16] + 18/400 *
Y[17] + 19/400 * Y[18] + 20/400 * Y[19] + 19/400 * Y[20] + 18/400 * Y[21] +
17/400 * Y[22] + 16/400 * Y[23] + 15/400 * Y[24] + 14/400 * Y[25] + 13/400 *
Y[26] + 12/400 * Y[27] + 11/400 * Y[28] + 10/400 * Y[29] + 9/400 * Y[30] + 8/400
* Y[31] + 7/400 * Y[32] + 6/400 * Y[33] + 5/400 * Y[34] + 4/400 * Y[35] + 3/400
* Y[36] + 2/400 * Y[37] + 1/400 * Y[38]

Source integral = 12368939, result integral = 12391951, divergence 0%

United extremums =  [101808.98088147749, 83088.12768917497, 94990.17712530504,
45124.10458270749]
```

spectrum1.csv



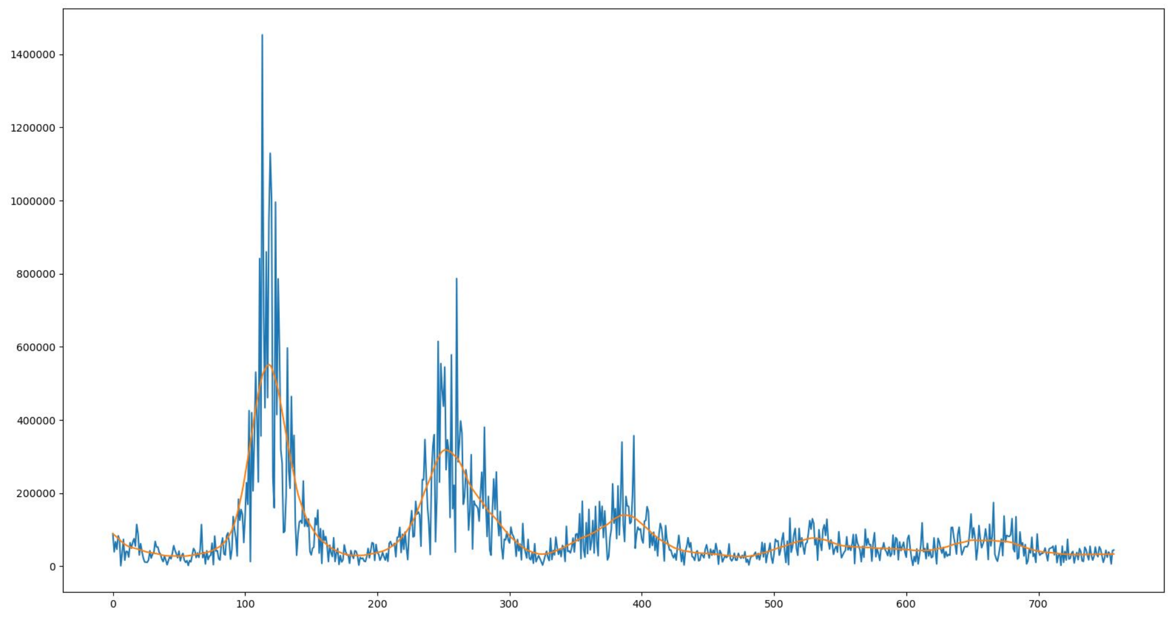

```
Formula example :  result[0] = 1/676 * Y[0] + 2/676 * Y[1] + 3/676 * Y[2] +
4/676 * Y[3] + 5/676 * Y[4] + 6/676 * Y[5] + 7/676 * Y[6] + 8/676 * Y[7] + 9/676
* Y[8] + 10/676 * Y[9] + 11/676 * Y[10] + 12/676 * Y[11] + 13/676 * Y[12] +
14/676 * Y[13] + 15/676 * Y[14] + 16/676 * Y[15] + 17/676 * Y[16] + 18/676 *
Y[17] + 19/676 * Y[18] + 20/676 * Y[19] + 21/676 * Y[20] + 22/676 * Y[21] +
23/676 * Y[22] + 24/676 * Y[23] + 25/676 * Y[24] + 26/676 * Y[25] + 25/676 *
Y[26] + 24/676 * Y[27] + 23/676 * Y[28] + 22/676 * Y[29] + 21/676 * Y[30] +
20/676 * Y[31] + 19/676 * Y[32] + 18/676 * Y[33] + 17/676 * Y[34] + 16/676 *
Y[35] + 15/676 * Y[36] + 14/676 * Y[37] + 13/676 * Y[38] + 12/676 * Y[39] +
11/676 * Y[40] + 10/676 * Y[41] + 9/676 * Y[42] + 8/676 * Y[43] + 7/676 * Y[44]
+ 6/676 * Y[45] + 5/676 * Y[46] + 4/676 * Y[47] + 3/676 * Y[48] + 2/676 * Y[49]
+ 1/676 * Y[50]
Source integral = 755763, result integral = 755811, divergence 0%
United extremums =  [4107.817928033912, 5560.504520978088, 2920.24559544887,
987.0502142876047]
```

spectrum2.csv



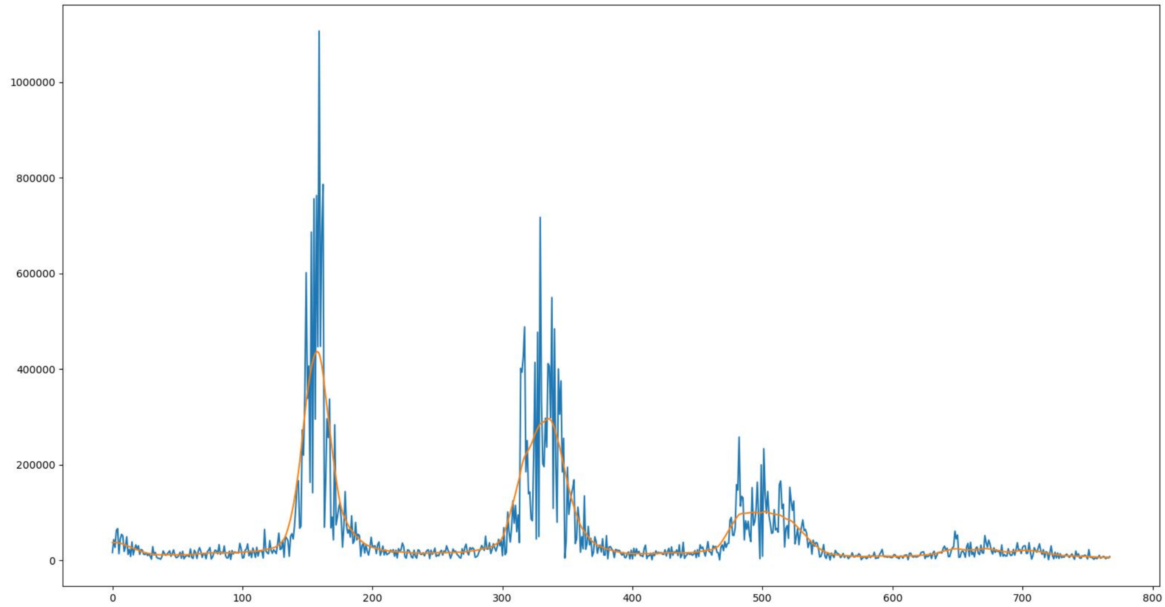

```
Formula example :  result[0] = 1/484 * Y[0] + 2/484 * Y[1] + 3/484 * Y[2] + 4/484 * Y[3] + 5/484 * Y[4] + 6/484 * Y[5] + 7/484 * Y[6] + 8/484 * Y[7] + 9/484 * Y[8] + 10/484 * Y[9] + 11/484 * Y[10] + 12/484 * Y[11] + 13/484 * Y[12] + 14/484 * Y[13] + 15/484 * Y[14] + 16/484 * Y[15] + 17/484 * Y[16] + 18/484 * Y[17] + 19/484 * Y[18] + 20/484 * Y[19] + 21/484 * Y[20] + 22/484 * Y[21] + 21/484 * Y[22] + 20/484 * Y[23] + 19/484 * Y[24] + 18/484 * Y[25] + 17/484 * Y[26] + 16/484 * Y[27] + 15/484 * Y[28] + 14/484 * Y[29] + 13/484 * Y[30] + 12/484 * Y[31] + 11/484 * Y[32] + 10/484 * Y[33] + 9/484 * Y[34] + 8/484 * Y[35] + 7/484 * Y[36] + 6/484 * Y[37] + 5/484 * Y[38] + 4/484 * Y[39] + 3/484 * Y[40] + 2/484 * Y[41] + 1/484 * Y[42]
Source integral = 78619697, result integral = 78702614, divergence 0%
United extremums =  [119307.26699311301, 615874.3932366815, 311373.83915023284]
```

spectrum3.csv



```
Formula example :  result[0] = 1/441 * Y[0] + 2/441 * Y[1] + 3/441 * Y[2] +
4/441 * Y[3] + 5/441 * Y[4] + 6/441 * Y[5] + 7/441 * Y[6] + 8/441 * Y[7] + 9/441
* Y[8] + 10/441 * Y[9] + 11/441 * Y[10] + 12/441 * Y[11] + 13/441 * Y[12] +
14/441 * Y[13] + 15/441 * Y[14] + 16/441 * Y[15] + 17/441 * Y[16] + 18/441 *
Y[17] + 19/441 * Y[18] + 20/441 * Y[19] + 21/441 * Y[20] + 20/441 * Y[21] +
19/441 * Y[22] + 18/441 * Y[23] + 17/441 * Y[24] + 16/441 * Y[25] + 15/441 *
Y[26] + 14/441 * Y[27] + 13/441 * Y[28] + 12/441 * Y[29] + 11/441 * Y[30] +
10/441 * Y[31] + 9/441 * Y[32] + 8/441 * Y[33] + 7/441 * Y[34] + 6/441 * Y[35] +
5/441 * Y[36] + 4/441 * Y[37] + 3/441 * Y[38] + 2/441 * Y[39] + 1/441 * Y[40]
Source integral = 69240690, result integral = 69477941, divergence 0%
United extremums =  [551323.5126452298, 318377.5669786369, 140271.45777360263]
```

spectrum4.csv




```
Formula example :  result[0] = 1/256 * Y[0] + 2/256 * Y[1] + 3/256 * Y[2] +
4/256 * Y[3] + 5/256 * Y[4] + 6/256 * Y[5] + 7/256 * Y[6] + 8/256 * Y[7] + 9/256
* Y[8] + 10/256 * Y[9] + 11/256 * Y[10] + 12/256 * Y[11] + 13/256 * Y[12] +
14/256 * Y[13] + 15/256 * Y[14] + 16/256 * Y[15] + 15/256 * Y[16] + 14/256 *
Y[17] + 13/256 * Y[18] + 12/256 * Y[19] + 11/256 * Y[20] + 10/256 * Y[21] +
9/256 * Y[22] + 8/256 * Y[23] + 7/256 * Y[24] + 6/256 * Y[25] + 5/256 * Y[26] +
4/256 * Y[27] + 3/256 * Y[28] + 2/256 * Y[29] + 1/256 * Y[30]
Source integral = 40079961, result integral = 40079927, divergence 0%
United extremums =  [436598.36969866656, 296067.5766324578, 101888.57217850279]
```