Университет ИТМО

Факультет: ПИиКТ

Дисциплина: Вычислительная математика

Лабораторная работа №4

Вариант “4”

Выполнили: **Кудлаков Роман**

Группа: **P3231**

Преподаватель: **Перл Ольга Вячеславовна**

Санкт-Петербург, 2021

Описание метода

Аппроксимация методом наименьших квадратов – метод, основанный на замене экспериментально полученных данных аналитической функцией, которая вычисляется при помощи построения аппроксимирующего многочлена n-степени. Точки данного многочлена проходят максимально близко к экспериментальным точкам. Близость исходной и аппроксимирующей функции определяется числовой мерой: сумма квадратов отклонений экспериментальных данных от аппроксимирующей кривой должна быть наименьшей.

$$\sum_{i=1}^{N} \xi_i^2 = \sum_{i=1}^{N} (F(x_i) - y_i)^2 \ \rightarrow min,$$

$F(x_i)$ − значения аппроксимирующей функции в узловых точках x_i,
y_i − заданный массив экспериментальных данных в узловых точках x_i,
N − количество экспериментальных точек

Аппроксимирующие функции могут быть представлены в виде:

1. Многочлена степени m

$$F_m(x) = a_0 + a_1 * x + \cdots + a_{m-1} * x^{m-1} + a_m * x^m$$

a − неизвестные коэффициенты аппроксимирующей функции

$$1 \le m \le N - 1$$

2. Логарифмической функции

$$F(x) = a + b * \ln(x)$$

3. Экспоненциальной функции

$$F(x) = a * e^{b*x}$$

4. Степенной функции

$$F(x) = a * x^b$$

Алгоритм реализации для многочлена степени m выглядит следующим образом:

1. Создается набор произвольных пар (x, y)
2. Задается степень многочлена
3. Определяются коэффициенты для построения системы уравнений размерностью (m+1)

$$c_{k,j} = \sum_{i=1}^{N} x_i^{k+j-2}$$

$c_{k,j}$ − коэффициенты системы уравнений

$$d_k = \sum_{i=1}^{N} (y_i * x^{k-1})$$

$$d_k - \text{свободные члены системы линейных уравнений}$$
$$k, j = 1, \dots, m+1$$

4. Формируется системы линейных уравнений размерностью (m+1)

$$\begin{bmatrix} c_{1,1} & \cdots & c_{1,j} \\ \vdots & \ddots & \vdots \\ c_{k,1} & \cdots & c_{k,j} \end{bmatrix} * \begin{vmatrix} a_0 \\ \vdots \\ a_m \end{vmatrix} = \begin{vmatrix} d_1 \\ \vdots \\ d_k \end{vmatrix}$$

5. Решая систему, получаем коэффициенты аппроксимирующего многочлена степени m

6. Подставляем коэффициенты в формулу многочлена степени m

В алгоритмах для логарифмической, экспоненциальной и степенной функций вместо того, чтобы формировать систему уравнений и потом ее решать, можно сразу найти коэффициенты a и b подставив значения в следующие формулы:

1. Для логарифмической функции

$$a = \frac{\sum_{i=1}^{N} y_i * \sum_{i=1}^{N} (\ln(x_i))^2 - \sum_{i=1}^{N} (y_i * \ln(x_i)) * \sum_{i=1}^{N} \ln(x_i)}{N * \sum_{i=1}^{N} (\ln(x_i))^2 - \left(\sum_{i=1}^{N} \ln(x_i)\right)^2}$$

$$b = \frac{N * \sum_{i=1}^{N} (y * \ln(x_i)) - \sum_{i=1}^{N} y_i * \sum_{i=1}^{N} \ln(x_i)}{N * \sum_{i=1}^{N} (\ln(x_i))^2 - \left(\sum_{i=1}^{N} \ln(x_i)\right)^2}$$

2. Для экспоненциальной функции

$$\ln(a) = \frac{\sum_{i=1}^{N} \ln(y_i) * \sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} (\ln(y_i) * x_i) * \sum_{i=1}^{N} x_i}{N * \sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2}$$
$$b = \frac{N * \sum_{i=1}^{N} (\ln(y_i) * x_i) - \sum_{i=1}^{N} \ln(y_i) * \sum_{i=1}^{N} x_i}{N * \sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2}$$

3. Для степенной функции

$$\ln(a) = \frac{\sum_{i=1}^{N} \ln(y_i) * \sum_{i=1}^{N} (\ln(x_i))^2 - \sum_{i=1}^{N} (\ln(y_i) * \ln(x_i)) * \sum_{i=1}^{N} \ln(x_i)}{N * \sum_{i=1}^{N} (\ln(x_i))^2 - (\sum_{i=1}^{N} \ln(x_i))^2}$$
$$b = \frac{N * \sum_{i=1}^{N} (\ln(y_i) * \ln(x_i)) - \sum_{i=1}^{N} \ln(y_i) * \sum_{i=1}^{N} \ln(x_i)}{N * \sum_{i=1}^{N} (\ln(x_i))^2 - (\sum_{i=1}^{N} \ln(x_i))^2}$$

Листинг кода

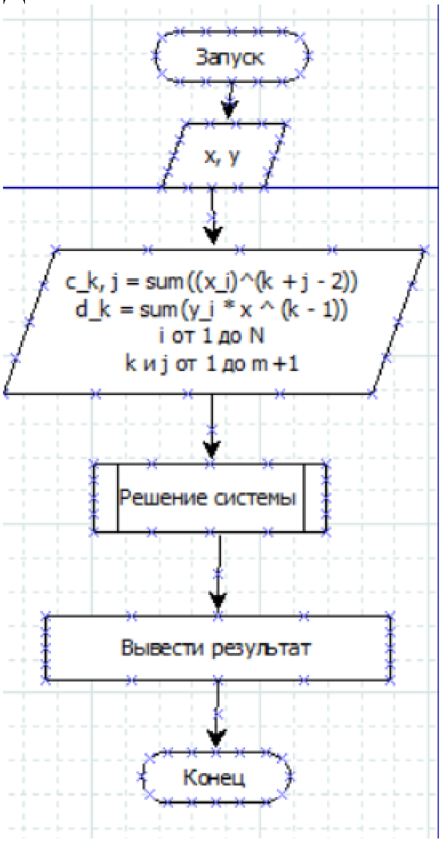
```
def make_linear_system(variables, sums, points):
    funcs = []
    for i in range(len(variables)):
        values = [variables[j] * sums[i + j]
                  for j in range(len(variables))]
```

```
        funcs.append(sum(values) - sum(points[:, 0] ** i * points[:, 1]))
    return funcs


def count_coefs(funcs, variables):
    coefs = linsolve(funcs, variables).args[0]
    coefs = rid_of_free_variables(coefs, variables)
    return coefs


def index_of_the_biggest_difference(differences):
    max = differences[0]
    index = 0
    for i in range(1, len(differences)):
        if max < differences[i]:
            max = differences[i]
            index = i
    return index


def count_differences(coefs, points):
    differences = []
    for i in range(len(points)):
        total = 0
        for j in range(len(coefs)):
            total += coefs[j] * points[i][0] ** j
        differences.append(abs(total - points[i][1]))
    return differences


def count_sums_x_in_different_pow(points, degree):
    sums = []
    for i in range(2 * degree + 1):
        sums.append(sum(points[:, 0] ** i))
    return sums


def rid_of_free_variables(coefs, variables):
    list_of_values = [(variables[i], 1) for i in range(len(variables))]
    new_coefs = []
    for i in range(len(coefs)):
        new_coefs.append(coefs[i].subs(list_of_values).evalf())
    return new_coefs

def do_polynomial_approximation(points, variables, degree):
    sums_x_in_different_pow = count_sums_x_in_different_pow(points, degree)

    functions = make_linear_system(variables, sums_x_in_different_pow, points)
    first_coefs = count_coefs(functions, variables)
    differences = count_differences(first_coefs, points)

    index = index_of_the_biggest_difference(differences)
    for i in range(len(sums_x_in_different_pow)):
        sums_x_in_different_pow -= points[index][0] ** i

    second_points = np.delete(points, index, axis=0)
    sums_x_in_different_pow = count_sums_x_in_different_pow(second_points, degree)

    functions = make_linear_system(variables, sums_x_in_different_pow,
second_points)
    second_coefs = count_coefs(functions, variables)
    differences = count_differences(second_coefs, second_points)
    return [first_coefs, second_coefs, index]


def do_logarithmic_approximation(points, variables, degree):
    for i in range(len(points)):
```

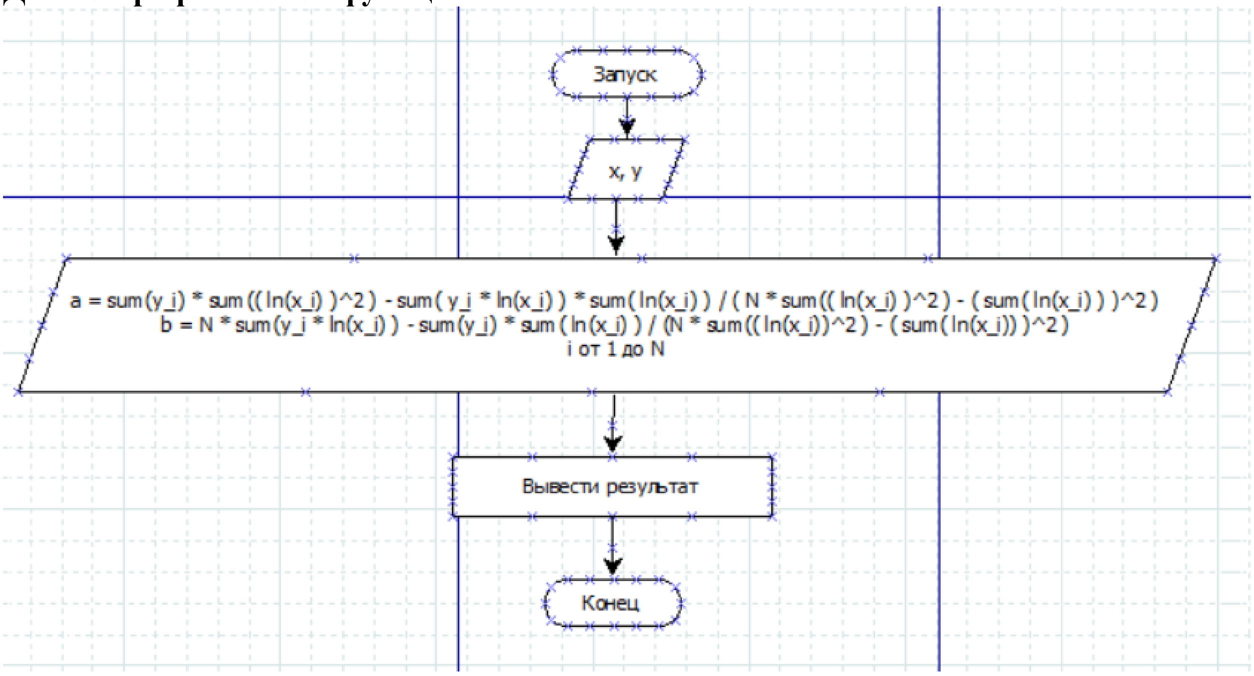
```
        points[i][0] = log(points[i][0])
    sums_x_in_different_pow = count_sums_x_in_different_pow(points, degree)

    functions = make_linear_system(variables, sums_x_in_different_pow, points)
    first_coefs = count_coefs(functions, variables)
    differences = count_differences(first_coefs, points)

    index = index_of_the_biggest_difference(differences)
    for i in range(len(sums_x_in_different_pow)):
        sums_x_in_different_pow -= points[index][0] ** i

    second_points = np.delete(points, index, axis=0)
    sums_x_in_different_pow = count_sums_x_in_different_pow(second_points, degree)

    functions = make_linear_system(variables, sums_x_in_different_pow,
second_points)
    second_coefs = count_coefs(functions, variables)
    return [first_coefs, second_coefs, index]


def do_exponential_approximation(points, variables, degree):
    for i in range(len(points)):
        points[i][1] = log(points[i][1])
    sums_x_in_different_pow = count_sums_x_in_different_pow(points, degree)

    functions = make_linear_system(variables, sums_x_in_different_pow, points)
    first_coefs = count_coefs(functions, variables)
    differences = count_differences(first_coefs, points)

    index = index_of_the_biggest_difference(differences)
    for i in range(len(sums_x_in_different_pow)):
        sums_x_in_different_pow -= points[index][0] ** i

    second_points = np.delete(points, index, axis=0)
    sums_x_in_different_pow = count_sums_x_in_different_pow(second_points, degree)

    functions = make_linear_system(variables, sums_x_in_different_pow,
second_points)
    second_coefs = count_coefs(functions, variables)
    return [first_coefs, second_coefs, index]

def do_power_approximation(points, variables, degree):
    for i in range(len(points)):
        points[i][0] = log(points[i][0])
        points[i][1] = log(points[i][1])
    sums_x_in_different_pow = count_sums_x_in_different_pow(points, degree)

    functions = make_linear_system(variables, sums_x_in_different_pow, points)
    first_coefs = count_coefs(functions, variables)
    differences = count_differences(first_coefs, points)

    index = index_of_the_biggest_difference(differences)
    for i in range(len(sums_x_in_different_pow)):
        sums_x_in_different_pow -= points[index][0] ** i

    second_points = np.delete(points, index, axis=0)
    sums_x_in_different_pow = count_sums_x_in_different_pow(second_points, degree)

    functions = make_linear_system(variables, sums_x_in_different_pow,
second_points)
    second_coefs = count_coefs(functions, variables)
    return [first_coefs, second_coefs, index]
```

Блок-схема

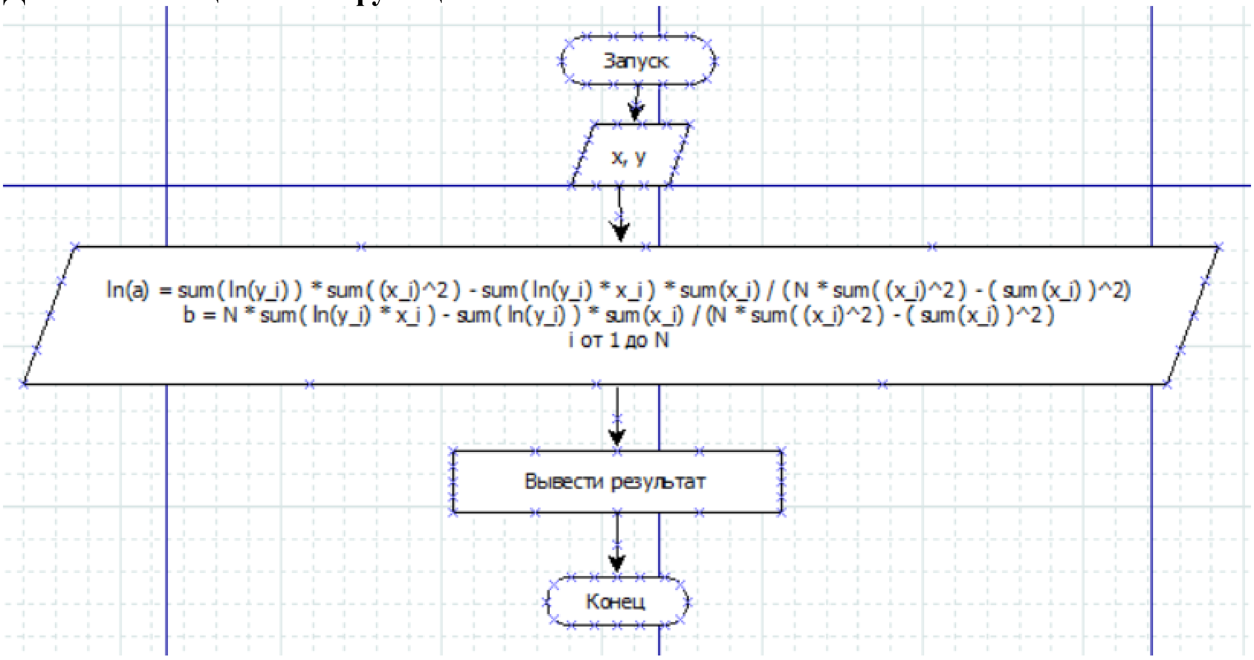
Для многочлена степени m



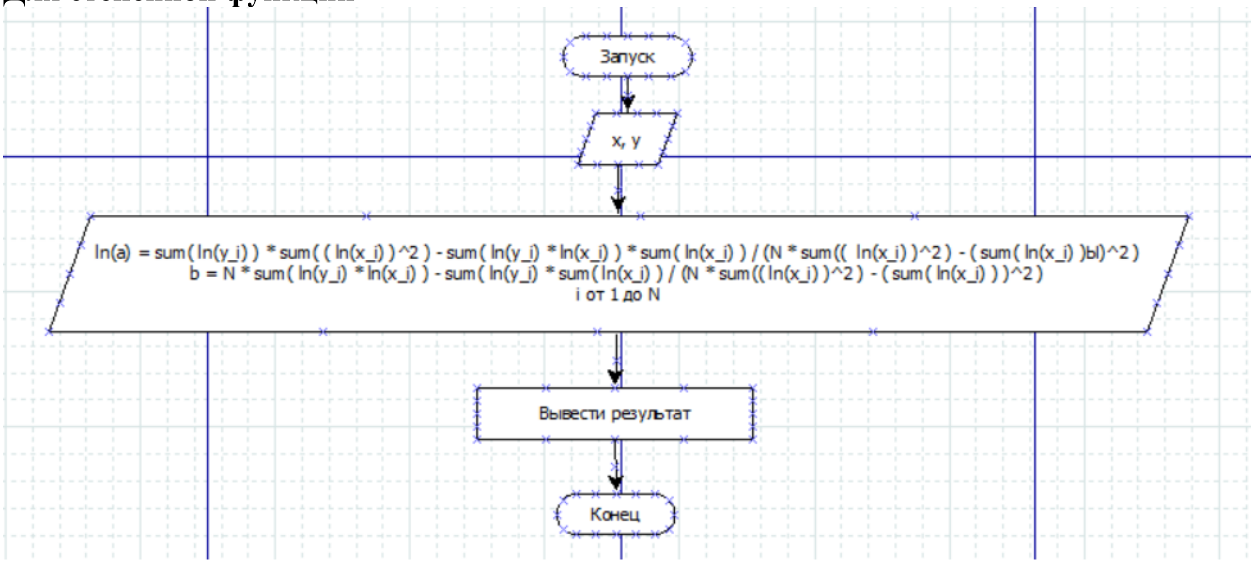
Для логарифмической функции



Для экспоненциальной функции



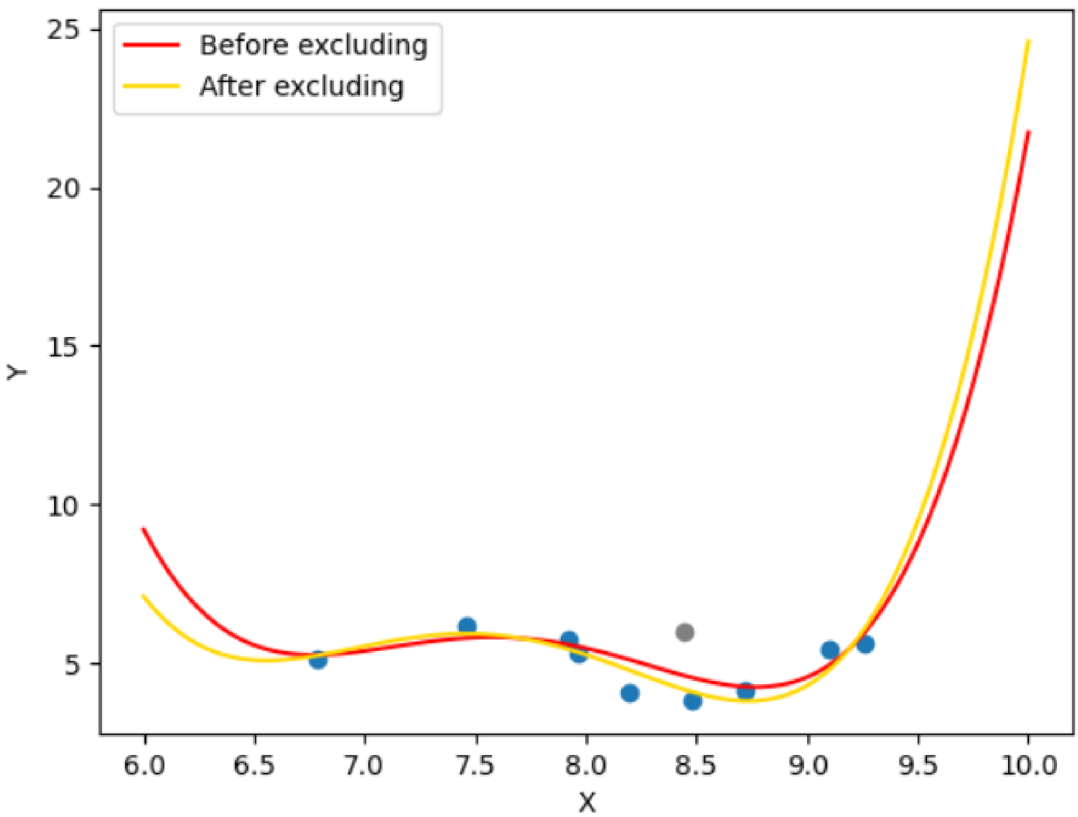
Для степенной функции



Тесты и результаты

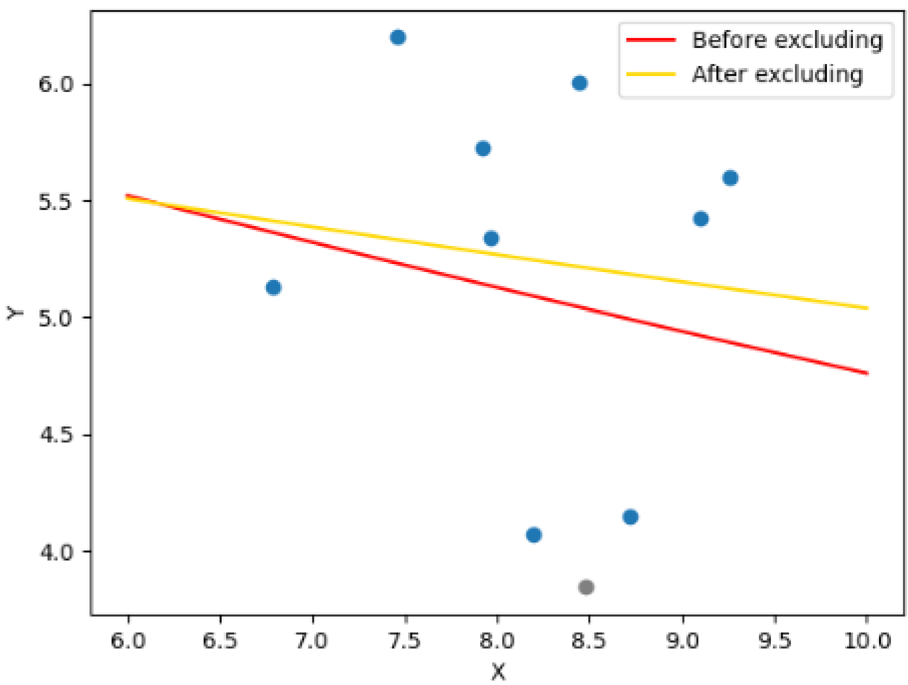
Тест 1.

```
Choose one of the approximation functions:
1) Polynomial
2) Logarithmic
3) Exponential
1
Input degree of an approximation function
4
First coefficients
x0 = 3413.47391332409
x1 = -1798.79781187870
x2 = 354.148338317871
x3 = -30.8185963580830
x4 = 1.00000000000000
Second coefficients
x0 = 3179.90392203366
x1 = -1708.73660076471
x2 = 342.658805847168
x3 = -30.3338190783382
x4 = 1.00000000000000
```



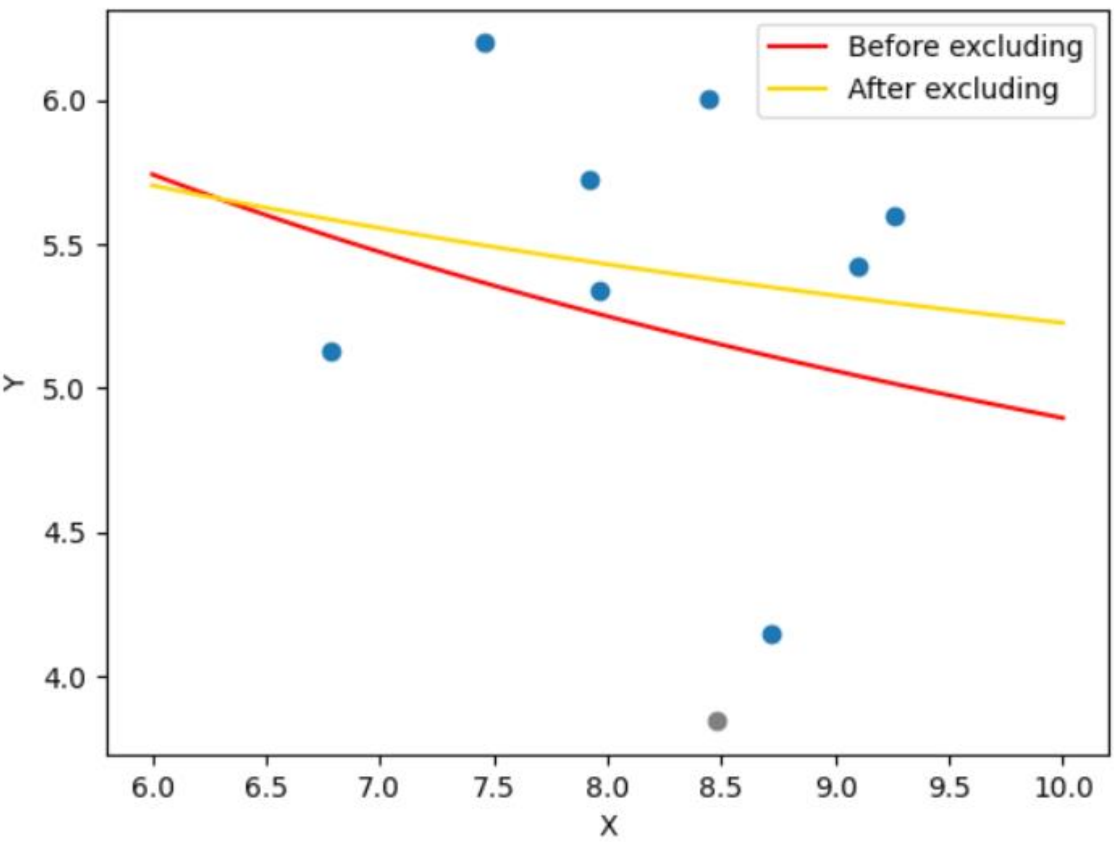
Тест 2.

```
Choose one of the approximation functions:
1) Polynomial
2) Logarithmic
3) Exponential
3
First coefficients
x0 = 1.93041761132990
x1 = -0.0369990682900864
Second coefficients
x0 = 1.83975203275482
x1 = -0.0222870437807916
```



Тест 3

```
Choose one of the approximation functions:
1) Polynomial
2) Logarithmic
3) Exponential
4) Power
4
First coefficients
x0 = 2.30711085524467
x1 = -0.312057489078826
Second coefficients
x0 = 2.04814138574922
x1 = -0.171257935233950
```



Вывод.

Аппроксимация – метод, основанный на замене экспериментально полученных данных аналитической функцией наиболее близкой или совпадающей в узловых точках с исходными значениями. То есть задача состоит в том, чтобы данную функцию f(x) приближенно заменить некоторой функцией φ(x), такой что

$$f(x) \approx \varphi(x)$$

Одним из способов аппроксимации функций является интерполяция. Данный способ используется, когда основная информация о приближаемой функции дается в виде таблицы значений. В итоге получается кривая, соответствующая интерполирующей функции, которая обязательно будет проходить через все точки исходных данных. Чаще всего в качестве интерполяционной функции выбирают алгебраический многочлен n-й степени, удовлетворяющий условию интерполяции:

$$P_n(x_i) = f(x_i)$$

$$i = 0, 1, \dots, n$$

$$P_n(x_i) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Геометрический смысл интерполяции состоит в том, что графики функций $y = f(x)$ и $y = P_n(x)$ должны проходить через все точки $(x_i, y_i), i = 0, 1, \dots, n$. В широком смысле это означает, что для заранее известных пар точек (x, y), для которых неизвестна связь в виде некоторой зависимости $y = f(x)$, ее требуется установить, так как зачастую на практике нам могут понадобиться не только значения найденные в данной области, а также и значения из любой другой, и способ нахождения таких значений называется экстраполяцией. Однако значения, полученные при помощи экстраполяции, могут рассматриваться только как вероятностные оценки, так как в действительности тенденция движения функции непостоянна.

Интерполяцией данные описываются точнее, чем при аппроксимации, но в некоторых случаях все-таки лучше применять именно аппроксимацию:

1. При значительном количестве табличных данных

2. Если интерполирующей функцией невозможно описать данные при повторении эксперимента в одних и тех же начальных условиях

3. Для сглаживания погрешностей эксперимента, так как обычно данные содержат выбросы, а это значит, что интерполяционная формула будет повторять эти значения, чего бы нам не хотелось

Аппроксимация методом наименьших квадратов дает такие математические свойства для полученной функции как:

1. Непрерывная дифференцируемость (может пригодится для минимизации значения квадратов)

2. Достаточная статистика для гауссовского распределения

3. Квадратная норма (может быть использована для доказательства сходимости функции)

Также аппроксимация может быть проведена при помощи абсолютных значений, однако функция уже не будет иметь свойства, указанные выше.