Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики
Факультет программной инженерии и компьютерной техники

Вычислительная математика
Лабораторная №3.1

Выполнил:
Беляков Дмитрий
Группа:
P3210
Преподаватель:
Перл О.В.

Санкт-Петербург
2020

Описание метода наименьших квадратов:

Основная задача аппроксимации — приближенная замена данной функции f(x) некоторой функцией $\phi(x)$, значения которой в заданной области мало чем отличались от опытных данных. Если более простыми словами: замена сложной функции более простой или выявление функциональной зависимости между данными.

1. Сначала выбирается общий вид функции (пример: линейная, квадратичная). Функция выбирается на основе входных данных. Например, исходя из физических характеристик исследуемых данных или геометрического представления экспериментальных точек.
2. Определение значений параметров аппроксимирующей функции: составляется мера отклонения $S(a_0, a_1, \dots, a_k)$, где $a_0, a_1, \dots, a_k$ - коэффициенты аппроксимирующей функции, и равной сумме квадратов разности между значениями аппроксимирующей функции и значениями y для каждого $x_0, x_1, \dots, x_n$ (грубо говоря, это расстояние между аппроксимирующей функцией и значениями y)

$$S = \sum_{i=1}^{n} (\phi(x_i) - y_i)^2 \to min$$

Задача состоит в поиске наилучших параметров $a_0, a_1, \dots, a_k$ - минимизация меры отклонения. Минимум найдём, приравнивая к нулю частные производные по $a_0, a_1, \dots, a_k$

$$\frac{\delta S}{\delta a_0} = \frac{\sum_{i=1}^{n} (\phi(x_i) - y_i)^2}{\delta a_0} = 2 \sum_{i=1}^{n} a_0 + a_1 x + \dots + a_{m-1} x_i^{m-1} - y_i = 0$$

$$\frac{\delta S}{\delta a_1} = \frac{\sum_{i=1}^{n} (\phi(x_i) - y_i)^2}{\delta a_1} = 2 \sum_{i=1}^{n} (a_0 + a_1 x + \dots + a_{m-1} x_i^{m-1} - y_i) x_i = 0$$

$$\dots$$

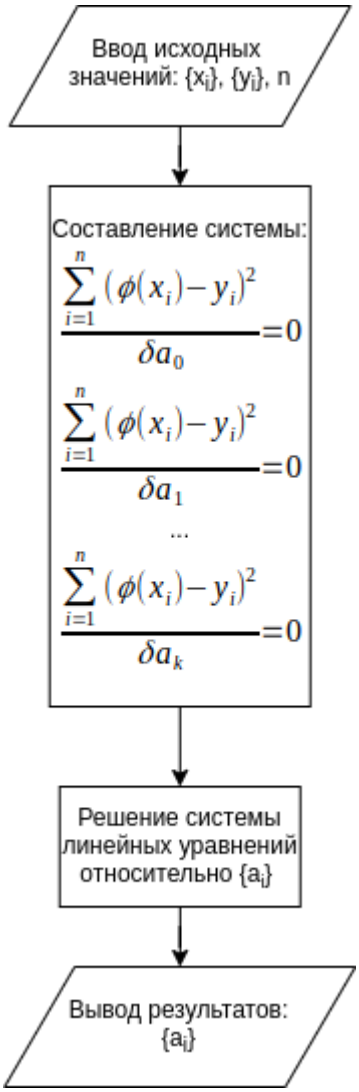
$$\frac{\delta S}{\delta a_k} = \frac{\sum_{i=1}^{n} (\phi(x_i) - y_i)^2}{\delta a_k} = 2 \sum_{i=1}^{n} (a_0 + a_1 x + \dots + a_{m-1} x_i^{m-1} - y_i) x_i^m = 0$$

Далее надо решить систему линейных уравнений относительно $a_0, a_1, \dots, a_k$.

Основные аппроксимирующие функции:

- Линейная: $\phi(x) = ax + b$
- Квадратичная: $\phi(x) = a_0 x^2 + a_1 x + a_2$
- Степенная: $\phi(x) = ax^b$
  Для данной функции решение аналогично линейной, только
  $Y = \ln(\phi(x)), A = b, B = \ln(a), X = \ln(x)$, из этого:
  $Y = AX + B$
- Экспоненциальная: $\phi(x) = ae^{bx}$
  Аналогично предыдущей:
  $Y = \ln(\phi(x)), A = b, B = \ln(a), X = x$, из этого:
  $Y = AX + B$
- Логарифмическая: $\phi(x) = a\ln(x) + b$
  $Y = \phi(x), A = a, B = b, X = \ln(x)$, из этого:
  $Y = AX + B$

Блок-схема



Листинг численного метода:

```
import numpy as np
from math import e
import matplotlib.pyplot as plt

def calc_linear_coefs(x, y):
    SX = np.sum(x)
    SXX = np.sum(x**2)
    SY = np.sum(y)
    SXY = np.sum(x*y)
    n = len(x)
    denominator = SXX*n - SX*SX
    if denominator == 0:
        raise Exception("Cann't calculate coefs: divizion by zero")
    a = (SXY*n - SX*SY)/denominator
    b = (SXX*SY-SX*SXY)/denominator
    return [a,b]
```

```
def calc_quadratic_coefs(x, y):
    X_1 = np.sum(x)
    X_2 = np.sum(x**2)
    X_3 = np.sum(x**3)
    X_4 = np.sum(x**4)
    Z_1 = np.sum(y)
    Z_2 = np.sum(x*y)
    Z_3 = np.sum(y*x**2)
    n = len(x)
    delta =   np.linalg.det([[  n, X_1, X_2],
                             [X_1, X_2, X_3],
                             [X_2, X_3, X_4]])
    if delta == 0:
        raise Exception("Cann't calculate coefs: divizion by zero")
    delta_0 = np.linalg.det([[Z_1, X_1, X_2],
                             [Z_2, X_2, X_3],
                             [Z_3, X_3, X_4]])
    delta_1 = np.linalg.det([[  n, Z_1, X_2],
                             [X_1, Z_2, X_3],
                             [X_2, Z_3, X_4]])
    delta_2 = np.linalg.det([[  n, X_1, Z_1],
                             [X_1, X_2, Z_2],
                             [X_2, X_3, Z_3]])

    return [delta_0/delta, delta_1/delta, delta_2/delta]

def calc_exp_coefs(x, y):
    if np.any(y <= 0):
        raise Exception("Cann't calculate coefs, y <= 0")
    a_1, a_0 = calc_linear_coefs(x, np.log(y))
    return [e**a_0, a_1]

def calc_pow_coefs(x, y):
    if np.any(x <= 0) or np.any(y <= 0):
        raise Exception("Cann't calulcate coefs, x <= 0 or y <= 0")
    a_1, a_0 = calc_linear_coefs(np.log(x), np.log(y))
    return [e**a_0, a_1]

def calc_log_coefs(x,y):
    if np.any(x <= 0):
        raise Exception("Cann't calculate coefs, x <= 0")
    a_0, a_1 = calc_linear_coefs(np.log(x), y)
    return [a_0, a_1]

def exclude_noise(func, x, y):
    index = np.argmax((func(x)-y)**2)
    return np.delete(x, index), np.delete(y, index)
```

```
class Function(object):

    def __init__(self, f_type):

        self.coefs = []
        self.recalc_coefs = []

        if f_type == 0:
            self.function = lambda a, b, x: a*x + b
            self.description = "{0:.3f}x + {1:.3f}"
        elif f_type == 1:
            self.function = lambda a_0, a_1, a_2, x: a_2*x**2 + a_1*x + a_0
            self.description = "{0:.3f} + {1:.3f}x + {2:.3f}x^2"
        elif f_type == 2:
            self.function = lambda a, b, x: a*e**(b*x)
            self.description = "{0:.3f}e^({1:.3f}x)"
        elif f_type == 3:
            self.function = lambda a, b, x: a*np.log(x)+b
            self.description = "{0:.3f}log(x) + {1:.3f}"
        else:
            self.function = lambda a,b, x: a*x**b
            self.description = "{0:.3f}x^{1:.3f}"

    def __call__(self, x, recalc_use = False):
        if recalc_use:
            coefs = self.recalc_coefs
        else:
            coefs = self.coefs
        return self.function(*coefs, x)

    def get_description(self, recalc_use = False):
        if recalc_use:
            return self.description.format(*self.recalc_coefs)
        else:
            return self.description.format(*self.coefs)

class LeastSquares(object):

    def __init__(self, f_type, x, y):
        self.x = x
        self.y = y
        self.function = Function(f_type)
        if f_type == 0:
            self.f_coefs = calc_linear_coefs
        elif f_type == 1:
            self.f_coefs = calc_quadratic_coefs
        elif f_type == 2:
            self.f_coefs = calc_exp_coefs
        elif f_type == 3:
```

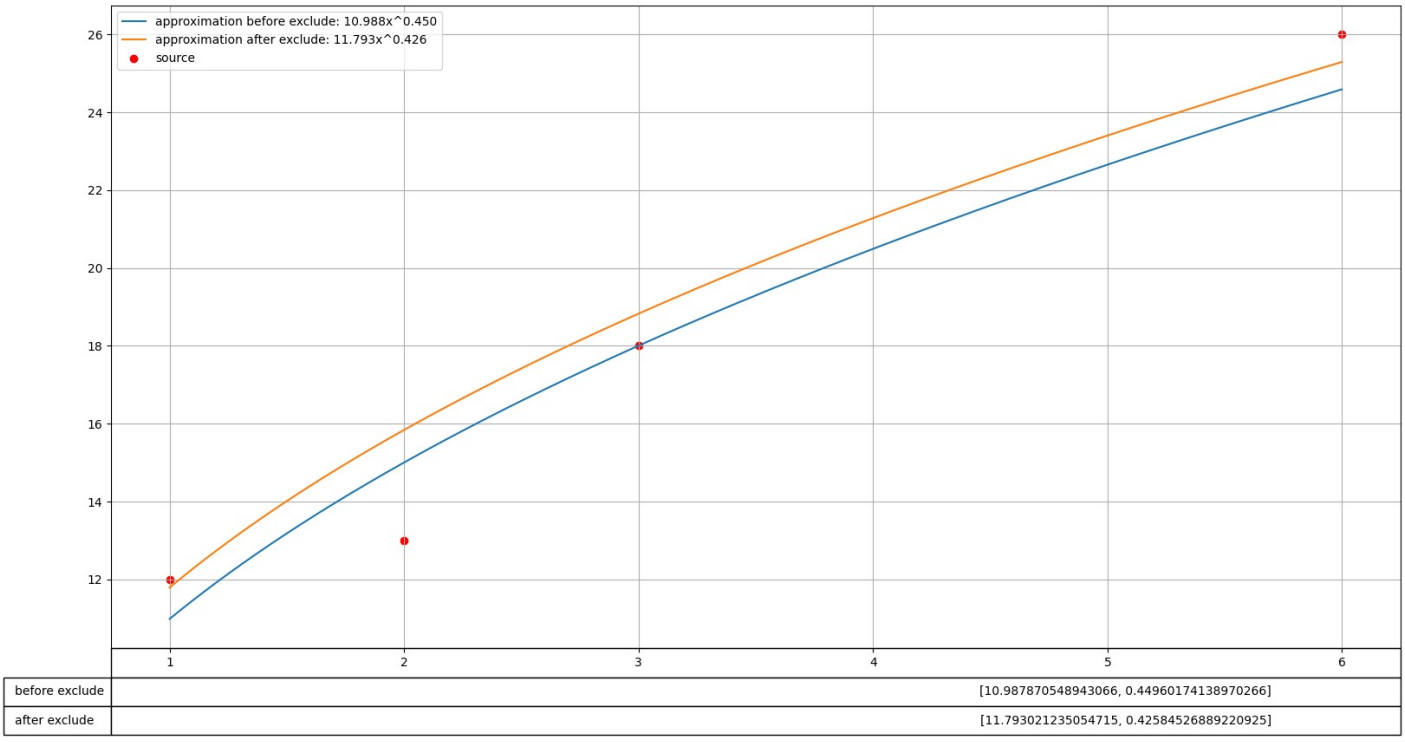
```
            self.f_coefs = calc_log_coefs
        else:
            self.f_coefs = calc_pow_coefs

    def calc_coefs(self):
        coefs = self.f_coefs(self.x, self.y)
        self.function.coefs = coefs

    def recalc_coefs(self):
        x, y = exclude_noise(self.function, self.x, self.y)
        coefs = self.f_coefs(x, y)
        self.function.recalc_coefs = coefs
```

Примеры и результаты работы:

Тестовые данные
1 12
2 13
3 18
6 26

Выбранный шаблон
ax^b

Результат

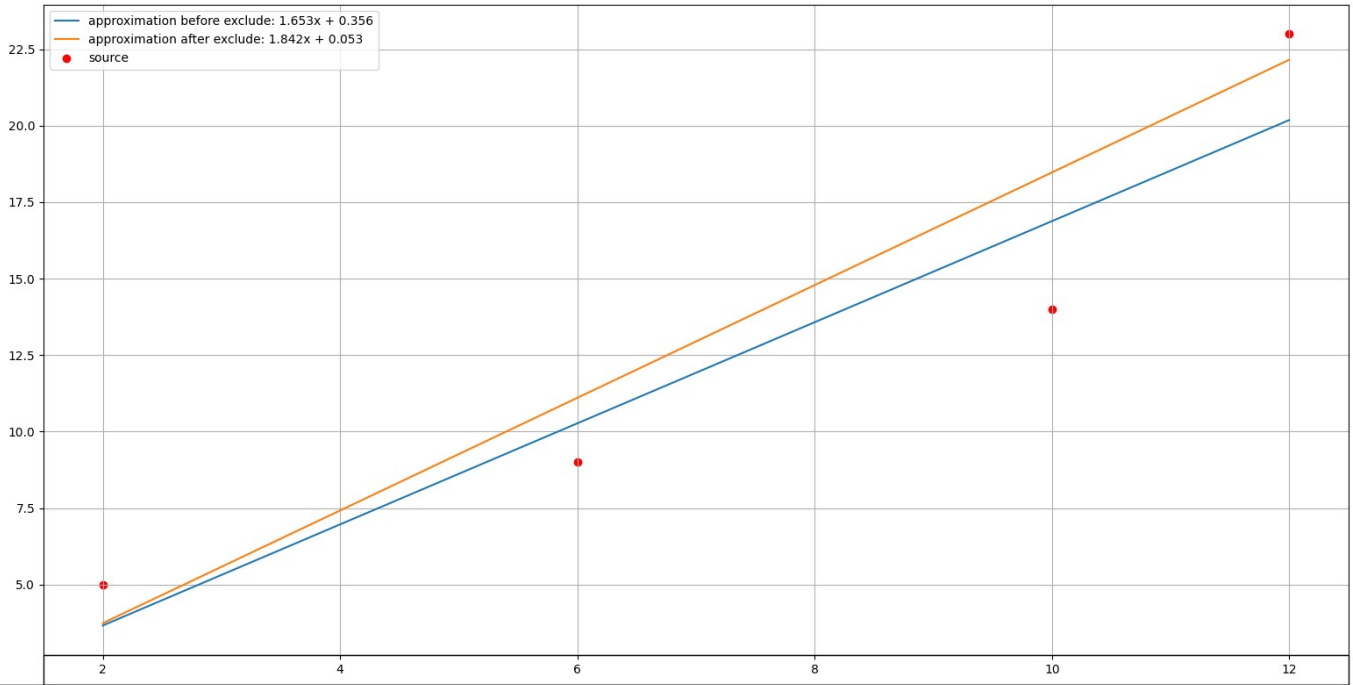


before exclude	[10.987870548943066, 0.44960174138970266]
after exclude	[11.793021235054715, 0.42584526889220925]

Тестовые данные
2 5
6 9
10 14
12 23

Выбранный шаблон
ax+b

Результат



before exclude	[1.652542372881356, 0.3559322033898305]
after exclude	[1.8421052631578947, 0.05263157894736842]

Вывод:
Аппроксимация — научный метод, отлично подходящий для решения целого набора задач: поиска функциональной зависимости, замена сложной функции на более простую, а так же поиск промежуточных значений, отличных от значений в узлах x_i .
Основной и самой сложной задачей при аппроксимации является подбор общего вида аппроксимирующей функции, так как от выбора зависит напрямую результат итоговой картины (например, если при взгляде на данные вы понимаете, что они растут по экспоненте, то гораздо рациональнее выбрать экспоненциальную функцию, а не, например, линейную). Что же касается исключения точки с наибольшим отклонением, то здесь ситуация неоднозначная. Точка с наибольшим отклонением может быть как ключевой: например, это вершина исходной параболы, а мы предположили, что зависимость между нашими точками линейная и на втором этапе исключили данную точку, так и выбросом, который надо исключить, после чего график аппроксимирующей функции заметно улучшается. Что ещё раз подчёркивает, что надо быть крайне внимательным при выборе общего вида аппроксимирующей функции.