Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по дисциплине

Вычислительная математика

Выполнил:
Студент группы P3230
Толстых М.А.

Преподаватель:
Перл И. А.
Перл О. В.

Санкт-Петербург, 2024

Описание метода

Метод наименьших модулей для построения линейной аппроксимации. Поиск значения модуля наибольшего отклонения среди заданных точек относительно полученной аппроксимации.

Изначально имеем набор данных x, y. X – независимая переменная, Y – зависимая переменная.

Виды аппроксимаций помимо линейной:

экспоненциальная $y = ae^{bx}$, логарифмическая $y = a + b * ln(x)$, полиномиальная, степенная $y = ax^b$ и т.д. Формулы для коэффициентов будут иметь различия в функциях над x и y. Например, для степенной зависимости расчетные формулы параметров a и b имеют такой же вид, однако добавится натуральный логарифм. Для наглядного понимания формула коэффициента b:

$$b = \frac{\sum(\ln(x_i) - \ln(x_0))(\ln(y_i) - \ln(y_0))}{\sum(\ln(x_i) - \ln(x_0))^2}$$

Соответственно, для каждой формулы аппроксимации будут свои подобные особенности в расчете параметров, но в нашем варианте требуется линейная аппроксимация.

Формула линейной выглядит следующим образом:

$$y = b + ax$$

a – угловой коэффициент, b – свободный член
Требуется вычислить эти коэффициенты методом наименьших модулей.

$$E_{LAD}(a, b) = \sum_i |y_i - b - ax_i| \to min$$

Используем формулу для нахождения коэффициента наклона a
(надо проверить, что это условие не выполняется: $\sum(x_i - x_0)^2 = 0$, если выполнится, то a = 0):

$$a = \frac{\sum(x_i - x_0)(y_i - y_0)}{\sum(x_i - x_0)^2}$$

Таким образом минимизируется сумма модулей отклонений точек от линии аппроксимации.
Далее находим свободный член по формуле, он позволит сместить линию аппроксимации вертикально:

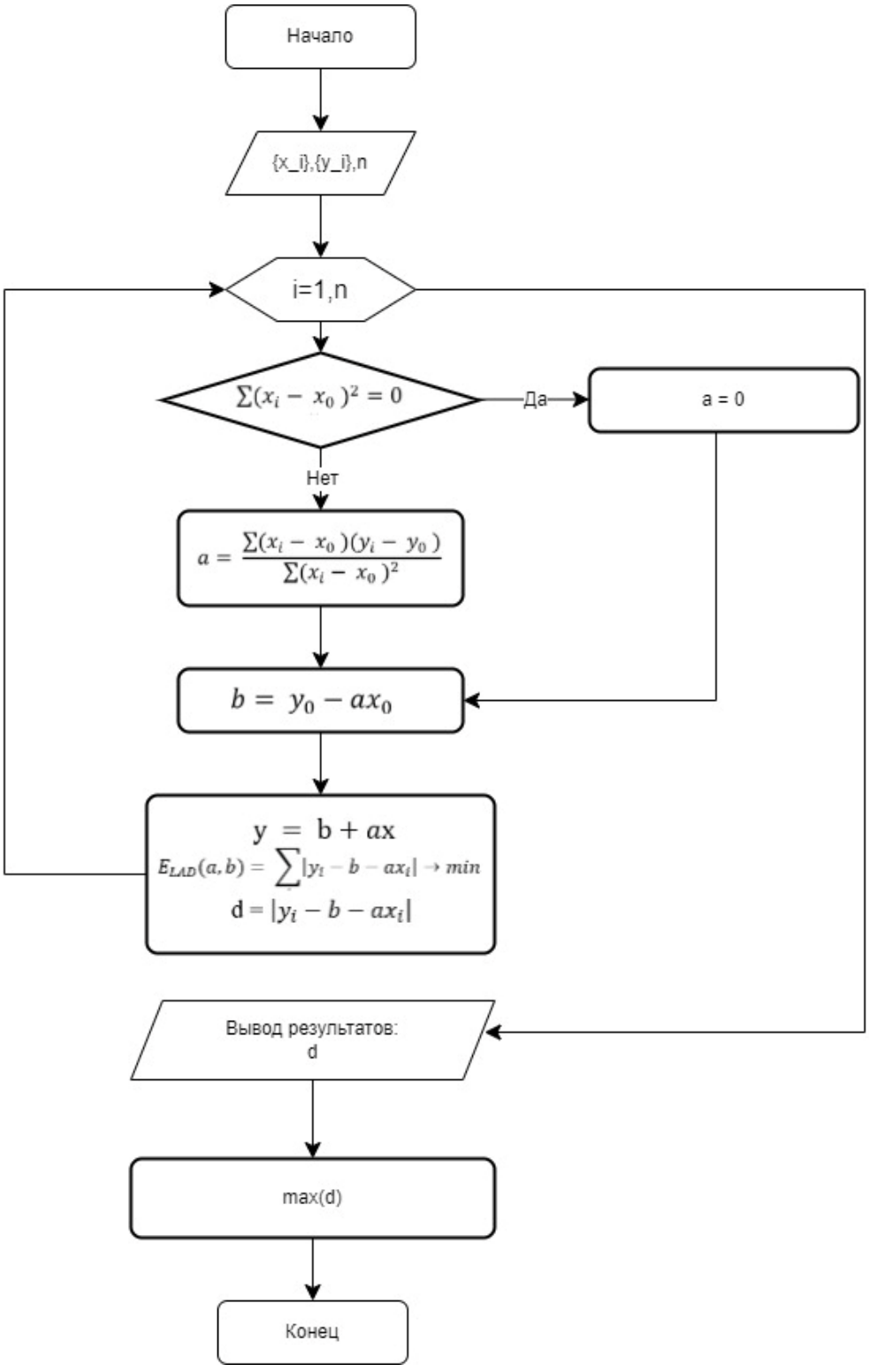
$$b = y_0 - ax_0$$

Итак, благодаря выше написанному алгоритму построения линейной аппроксимации методом наименьших модулей найдены коэффициенты. По условию задания нужно найти значение модуля наибольшего отклонения среди заданных точек относительно полученной аппроксимации. Значит, считаем модули по этой формуле и находим максимальный из них(max).

$$d = |y_i - b - ax_i|$$

d – отклонение заданных точек относительно полученной аппроксимации

Блок-схема численного метода



Код

```
def linear_least_absolute_approximation(x_axis, y_axis):
    n = len(x_axis)
    a = 0
    b = 0
    for i in range(n):
        a += (x_axis[i] - x_axis[0]) * (y_axis[i] - y_axis[0])
        b += (x_axis[i] - x_axis[0]) ** 2

    if b != 0:
        a /= b
    else:
        a = 0

    b = y_axis[0] - a * x_axis[0]

    return a, b


def approximate_linear_least_modules(x_axis, y_axis):
    a, b = linear_least_absolute_approximation(x_axis, y_axis)

    deviations = [abs(y - (a*x + b)) for x, y in zip(x_axis, y_axis)]
    max_deviation = max(deviations)

    return max_deviation
```

Этот вариант кода был загружен мной на code-n-test (поскольку загрузить больше нельзя исправленный код (1 попытка загрузки рабочего кода), я добавила исправленный вариант в отчет, чтобы обрабатывать ошибки и исключения корректно, чтобы они не выводились пользователю.)

```
def linear_least_absolute_approximation(x_axis, y_axis):
    try:
        n = len(x_axis)
        if n != len(y_axis) or n < 2:
            raise ValueError("Неверное количество элементов в массивах
данных")
        a = 0
        b = 0
        for i in range(n):
            a += (x_axis[i] - x_axis[0]) * (y_axis[i] - y_axis[0])
            b += (x_axis[i] - x_axis[0]) ** 2

        if b != 0:
            a /= b
        else:
            a = 0

        b = y_axis[0] - a * x_axis[0]

        return a, b
    except Exception as e:
        print(f"Ошибка: {e}")
        return None, None
```

```python
def approximate_linear_least_modules(x_axis, y_axis):
    try:
        a, b = linear_least_absolute_approximation(x_axis, y_axis)

        if a is None or b is None:
            return None

        deviations = [abs(y - (a*x + b)) for x, y in zip(x_axis, y_axis)]
        max_deviation = max(deviations)

        return max_deviation

    except Exception as e:
        print(f"Ошибка: {e}")
        return None
```

Примеры работы программы

1. Граничный случай с одним элементом

```
>? 1
>? 1
>? 2
0.0
```

2. Различные значения, но линейная аппроксимация не подходит (в линейной зависимости значения y должны изменяться пропорционально значениям x с постоянным коэффициентом)

```
>? 3
>? 1 2 3
>? 2 4 6
0.0
```

3. Положительные и отрицательные значения

```
>? 5
>? -1 2 -3 4 -5
>? 2 -3 5 -7 11
1.4444444444444446
```

4. Нулевой делитель (коэффициента a)

```
>? 2
>? 1 2
>? 3 6
0.0
```

5. Обычные данные, подходящие для линейной аппроксимации

```
>? 5
>? 1 2 3 4 5
>? 1.9 3.1 4.1 4.9 5.1
0.45333333333333314
```

Вывод

Этот метод поиска значения модуля наибольшего отклонения среди заданных точек относительно полученной аппроксимации, построенной методом наименьших модулей может быть удобен и полезен, когда данные содержат выбросы (как 3, 5 примеры) или не соответствуют линейному закону (как 2 пример), код реализованного метода устойчив к таким ситуациям.
Однако он может быть менее точным, чем метод наименьших квадратов, если данные хорошо соответствуют линейной зависимости. Чтобы сравнить эти два метода можем написать еще функцию, которая реализует метод наименьших квадратов и высчитывает наибольшее отклонение от линейной аппроксимации:

$$E_{OLS}(a,b) = \sum_i (ax_i + b - y_i)^2 \to min$$

```python
def approximate_linear_least_squares(x_axis, y_axis):
    n = len(x_axis)
    sum_x = sum(x_axis)
    sum_y = sum(y_axis)
    sum_x_squared = sum([x**2 for x in x_axis])
    sum_xy = sum([x*y for x, y in zip(x_axis, y_axis)])

    a = (n*sum_xy - sum_x*sum_y) / (n*sum_x_squared - sum_x**2)
    b = (sum_y - a*sum_x) / n

    deviations = [(y - (a*x + b))**2 for x, y in zip(x_axis, y_axis)]
    max_deviation = max(deviations)

    return max_deviation
def linear_least_absolute_approximation(x_axis, y_axis):
    n = len(x_axis)
    a = 0
    b = 0
    for i in range(n):
        a += (x_axis[i] - x_axis[0]) * (y_axis[i] - y_axis[0])
        b += (x_axis[i] - x_axis[0]) ** 2

    if b != 0:
        a /= b
    else:
        a = 0
```

```
    b = y_axis[0] - a * x_axis[0]

    return a, b


def approximate_linear_least_modules(x_axis, y_axis):
    a, b = linear_least_absolute_approximation(x_axis, y_axis)

    deviations = [abs(y - (a*x + b)) for x, y in zip(x_axis, y_axis)]
    max_deviation = max(deviations)

    return max_deviation

if __name__ == '__main__':
    axis_count = int(input().strip())

    x_axis = list(map(float, input().rstrip().split()))
    y_axis = list(map(float, input().rstrip().split()))

    result_absolute = approximate_linear_least_modules(x_axis, y_axis)
    result_squares = approximate_linear_least_squares(x_axis, y_axis)

    print("Max deviation for least modules method: " + str(result_absolute))
    print("Max deviation for least squares method: " + str(result_squares))
```

Такой вывод мы получим, если используем значения с большими выбросами:

```
>? -1 2 -3 4 -5
>? 2 -3 5 -7 11
Max deviation for least modules method: 1.4444444444444446
Max deviation for least squares method: 1.3581887048448185
```

А здесь маленькие выбросы:

```
>? 1 2 3 4 5
>? 2 4 5 8 9
Max deviation for least modules method: 0.6000000000000005
Max deviation for least squares method: 0.36000000000000065
```

Метод наименьших квадратов стремится минимизировать сумму квадратов отклонений между предсказанными значениями и реальными данными. При этом, если данные хорошо соответствуют линейной зависимости, то минимизация суммы квадратов отклонений даст наилучшую аппроксимацию этой зависимости. Метод наименьших квадратов будет более точным.

С другой стороны, метод наименьших модулей более устойчив к выбросам и большим отклонениям, но при этом он может быть менее точным в случае, когда данные хорошо соответствуют линейной зависимости. Это происходит потому, что метод наименьших модулей не учитывает веса отклонений так же эффективно, как метод наименьших квадратов.

Алгоритмическая сложность метода:

- Вычисление a и b в цикле длиной n: O(n)

- Вычисление отклонений и поиск максимального отклонения: O(n)

Алгоритмическая сложность метода составляет O(n), где n - количество точек данных. Этот метод линейно зависит от размера входных данных.

Анализ численной ошибки в самом численном методе:

В методе наименьших модулей используется модуль отклонения, что позволяет уменьшить влияние выбросов аппроксимации. Однако вычисление модуля может привести к потере точности из-за операции взятия модуля.

Поиск значения модуля наибольшего отклонения также может быть подвержен ошибкам округления при вычислении разности между значениями точек.