Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Лабораторная работа 4: «Аппроксимация функции методом наименьших квадратов»

«Вычислительная математика»

вариант 17

Выполнила:
Сенина Мария Михайловна
группа P3112
Преподователь:
Малышева Татьяна Алексеевна

ITMO UNIVERSITY

2022г.
г.Санкт-Петербург

1 Цель работы

Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

2 Порядок выполнения работы

2.1 Для исследования использовать

- линейную функцию
- полиномиальную функцию 2-й степени
- полиномиальную функцию 3-й степени
- экспоненциальную функцию
- логарифмическую функцию
- степенную функцию

2.2 Методика проведения исследования

1. Вычислить меру отклонения $S = \sum_{i=1}^{n}(\phi(x_i) - y_i)^2$ для всех исследуемых функций.
2. Уточнить значения коэффициентов эмпирических функций, минимизируя функцию S.
3. Сформировать массивы предполагаемых эмпирических зависимостей $(\phi(x_i), \epsilon)$.
4. Определить среднеквадратичное отклонение для каждой аппроксимирующей функции. Выбрать наименьшее значение и, следовательно, наилучшее приближение.
5. Построить графики полученных эмпирических функций.

2.3 Вычислительная реализация задачи

1. Для заданной функции (см. таблицу 1) построить наилучшие линейное и квадратичное приближения по 11 точкам указанного интервала.
2. Найти среднеквадратические отклонения. Ответы дать с тремя знаками после запятой.
3. Построить графики линейного и квадратичного приближений и заданной функции.
4. Привести в отчете подробные вычисления.

2.4 Программная реализация задачи:

1. Предусмотреть ввод исходных данных из файла/консоли (таблица $y = f(x)$ должна содержать 10 - 12 точек).
2. Реализовать метод наименьших квадратов, исследуя все функции п.1.
3. Предусмотреть вывод результатов в файл/консоль.
4. Для линейной зависимости вычислить коэффициент корреляции Пирсона.
5. Программа должна отображать наилучшую аппроксимирующую функцию.
6. Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом).

2.5 Анализ результатов работы

Апробация и тестирование.

3 Рабочие формулы

3.1 Вычислительная часть

По варианту нужно найти апроксимацию для точек из интервала $x \in [0; 2]$ с шагом $h = 0.2$, заданных функцией: $x = \frac{2x}{x^4+7}$.

Получается таблица и исходными данными такая: Точки оказываются уже отсортированные, так что сортировать их нам не надо.

№	x	$f(x)$
0	0.0	0.0
1	0.2	0.057
2	0.4	0.114
3	0.6	0.168
4	0.8	0.216
5	1.0	0.25
6	1.2	0.265
7	1.4	0.258
8	1.6	0.236
9	1.8	0.206
10	2.0	0.174

Таблица 1: Исходные данные

Линейная апроксимация

Чтобы посчитать коэффициенты для апроксимации данных линейной функцией вида $f(x) = ax + b$ нужно найти такую прямую, чтобы сумма квадратов расстояний от неё до данных была наименьшей. Коэфиициенты, при которых достигается минимумы суммы квадратов расстояний можно посчитать решив систему уравнений вида:

$$\begin{cases} \sum_{i=0}^{n} x_i^2 a + \sum_{i=0}^{n} x_i b = \sum_{i=0}^{n} x_i y_i, \\ \sum_{i=0}^{n} x_i a + nb + \sum_{i=0}^{n} y_i \end{cases}$$

Посчитаем коэффициенты в матрицу:

$\sum_{i=0}^{n} x_i$	$\sum_{i=0}^{n} x_i^2$	$\sum_{i=0}^{n} y_i$	$\sum_{i=0}^{n} x_i y_i$	n
11.0	15.4	1.944	2.356	11

Таблица 2: Коэффициенты в системе уравнений

Итого матрица: $\begin{pmatrix} 15.4 & 11.0 & 2.356 \\ 11.0 & 11 & 1.944 \end{pmatrix}$

Решение системы: $a = 0.083$; $b = 0.094$

Т.е. уравнение апроксимирующей функции: $f(x) = 0.083x + 0.094$

Квадратичная апроксимация

Тут уже нужная другая система уравнений:

$$\begin{cases} \text{n a}_0 + \sum_{i=0}^{n} x_i a_1 + \sum_{i=0}^{n} x_i^2 a_2 = \sum_{i=0}^{n} y_i, \\ \sum_{i=0}^{n} x_i a_0 + \sum_{i=0}^{n} x_i^2 a_1 + \sum_{i=0}^{n} x_i^3 a_2 = \sum_{i=0}^{n} x_i y_i, \\ \sum_{i=0}^{n} x_i^2 a_0 + \sum_{i=0}^{n} x_i^3 a_1 + \sum_{i=0}^{n} x_i^4 a_2 = \sum_{i=0}^{n} x_i^2 y_i, \end{cases}$$

Соответственно коэффициенты к этой системе:

$\sum_{i=0}^{n} x_i$	$\sum_{i=0}^{n} x_i^2$	$\sum_{i=0}^{n} x_i^3$	$\sum_{i=0}^{n} x_i^4$	$\sum_{i=0}^{n} y_i$	$\sum_{i=0}^{n} x_i y_i$	$\sum_{i=0}^{n} x_i^2 y_i$	n
11.0	15.4	24.2	40.533	1.944	2.356	3.323	11

Таблица 3: Коэффициенты в системе уравнений

Итого матрица: $\begin{pmatrix} 11 & 11.0 & 15.4 & 1.944 \\ 11.0 & 15.4 & 24.2 & 2.356 \\ 15.4 & 24.2 & 40.533 & 3.323 \end{pmatrix}$

Решение системы: $a_0 = -0.014$; $a_1 = 0.417 a_2 = -0.162$

Т.е. уравнение апроксимирующей функции: $f(x) = -0.014x^2 + 0.417x + -0.162$

3.2 Программная часть

Полиномиальная апроксимация Для линейной и квадратичной функций формулы и принципы вычисления коэффициентов апроксимирующей функции остаются те же.

В общем случае для полинома степени k необходимо решить систему вида:

$$\begin{cases} \text{na}_0 + \sum_{i=0}^n x_i a_1 + \sum_{i=0}^n x_i^2 a_2 + \dots + \sum_{i=0}^n x_i^2 a_k = \sum_{i=0}^n y_i, \\ \sum_{i=0}^n x_i a_0 + \sum_{i=0}^n x_i^2 a_1 + \sum_{i=0}^n x_i^3 a_2 + \dots + \sum_{i=0}^n x_i^2 a_{k+1} = \sum_{i=0}^n x_i y_i, \\ \dots \\ \sum_{i=0}^n x_i^k a_0 + \sum_{i=0}^n x_i^{k+1} a_1 + \sum_{i=0}^n x_i^4 a_2 + \dots + \sum_{i=0}^n x_i^{k^2} a_k = \sum_{i=0}^n x_i^2 y_i, \end{cases}$$

И получить коэффициенты $a_k, a_{k-1}, \dots, a_1, a_0$ функции: $f(x) = a_k x^k + \dots + a_1 x + a_0$

Степенная аппроксимация

Аппроксимирующая функция задана степенной функцией вида: $\phi = a^{xb}$

Для применения метода наименьших квадратов степенная функция линеаризуется: $ln(\phi(x)) = ln(a^{xb}) = ln(a) + bln(x)$

Введем обозначения: $Y = ln(\phi(x))$; $A = ln(a)$ $B = b$; $X = ln(x)$ Получаем линейную зависимость: $Y = A + B$. После определения коэффициентов А и В вернемся к принятым ранее обозначениям: $a = e^x$, $B = b$

Экспоненциальная аппроксимация

Аппроксимирующая функция задана степенной функцией вида: $\phi(x) = ae^{(bx)}$

Для применения метода наименьших квадратов степенная функция линеаризуется: $ln(\phi(x)) = ln(ae^{(bx)}) = ln(a) + ln(bx)$

Введем обозначения: $Y = ln(\phi(x))$; $A = ln(a)$; $B = b$

Получаем линейную зависимость: $Y = A + B$.

После определения коэффициентов А и В вернемся к принятым ранее обозначениям: $a = e^x$, $B = b$

А сама функция будет такой: $\phi(x) = aln(x) + b$

Логарифмическая апроксимация

Аппроксимирующая функция задана логарифмической функцией вида: $\phi(x) = aln(x)+b$ Введем обозначения: $Y = ln(\phi(x))$; $X = x$; $A = a$; $B = b$

Получаем линейную зависимость: $Y = A + B$.

Оценка

Оценка полученной функции может происходить через сравнение функций по какой-то одинаковой метрике. В нашем случае на роль хорошей метрики подойдёт дисперсия или среднеквадратическое отклонение:

Дисперсия

$$\sigma = \frac{\sum_{i=0}^n (\phi(x_i) - y_i}{n}$$

Среднеквадратическое отклонение

$$\sigma = \sqrt{\frac{\sum_{i=0}^n (\phi(x_i) - y_i}{n}}$$

Или же можно считать через R^2 статистику:

$$R^2 = \frac{\sum_{i=0}^n (y_i - \phi_i}{\sum_{i=0}^n \phi_i^2 - \frac{1}{n}(\sum_{i=0}^n \phi_i)^2}$$

Для линейной апроксимации так же можно оценить коэффициент Пирсона:

$$r = \frac{\sum_{i=0}^n (x_i - x_m)(y_i - y_m)}{\sum_{i=0}^n (x_i - x_m)^2 \sum_{i=0}^n (y_i - y_m)^2}$$

, где x_m и y_m – средние значения для x и y

4 Листинг программы

Ссылка репозиторий с кодом всей программы

4.1 Линейная аппроксимация

```
def line_approximation(points):

    n = len(points)
    x = [point[0] for point in points]
    y = [point[1] for point in points]

    sum_x = sum(x)
    sum_x2 = sum([xi ** 2 for xi in x])
    sum_y = sum(y)
    sum_xy = sum([x[i] * y[i] for i in range(n)])

    matrix =    [[sum_x2, sum_x, sum_xy],
                [sum_x, n, sum_y]]
    r = solve_matrix(matrix)
    a = r[1]
    b = r[0]

    result = {}
    result['a'] = a
    result['b'] = b
    f = lambda x: a * x + b

    result['function'] = f
    result['string_function'] = f"{round(a, 3)}x + {round(b, 3)}"
    result['variance'] = variance(points, f)
    result['standard_deviation'] = standard_deviation(points, f)

    mean_x = sum_x/n
    mean_y = sum_y/n
    sum_xy = sum([(point[0] - mean_x)*(point[1] - mean_y) for point in points])
    sum_x = sum([(xi - mean_y)**2 for xi in x])
    sum_y = sum([(yi - mean_y)**2 for yi in y])
    result['pirson'] = sum_xy/sqrt(sum_x*sum_y)

    return result
```

4.2 Квадратичная аппроксимация

```
def square_approximation(points):

    n = len(points)
    x = [dot[0] for dot in points]
    y = [dot[1] for dot in points]

    sum_x = sum(x)
    sum_x2 = sum([xi ** 2 for xi in x])
    sum_x3 = sum([xi ** 3 for xi in x])
    sum_x4 = sum([xi ** 4 for xi in x])
    sum_y = sum(y)
    sum_xy = sum([x[i] * y[i] for i in range(n)])
    sum_x2y = sum([(x[i] ** 2) * y[i] for i in range(n)])

    matrix =[[n, sum_x, sum_x2, sum_y],
            [sum_x, sum_x2, sum_x3, sum_xy],
            [sum_x2, sum_x3, sum_x4, sum_x2y]]
```

```
    r = solve_matrix(matrix)

    result = {}
    result['a_0'] = r[0]
    result['a_1'] = r[1]
    result['a_2'] = r[2]

    f = lambda i: r[2] * (i ** 2) + r[1] * i + r[0]
    result['function'] = f
    result['string_function'] = f"{round(r[2], 3)}x^2 + {round(r[1], 3)}*x + {round(r[0], 3)}"
    result['variance'] = variance(points, f)
    result['standard_deviation'] = standard_deviation(points, f)

    return result
```

4.3 Кубическая аппроксимация

```
def power_approximation(points):

    n = len(points)
    if not all(point[0] >= 0 and point[1] >= 0 for point in points): return None

    x = [point[0] for point in points]
    y = [point[1] for point in points]

    lin_x = [log(x[i]) for i in range(n)]
    lin_y = [log(y[i]) for i in range(n)]
    result_values = line_approximation([(lin_x[i], lin_y[i]) for i in range(n)])

    a = exp(result_values['b'])
    b = result_values['a']

    result = {}
    result['a'] = a
    result['b'] = b

    f = lambda i: a * (i ** b)
    result['function'] = f

    result['string_function'] = f"{round(a, 3)}*x^{round(b, 3)}"

    result['variance'] = variance(points, f)

    result['standard_deviation'] = standard_deviation(points, f)

    return result
```

4.4 Степенная аппроксимация

```
def power_approximation(points):

    n = len(points)
    if not all(point[0] >= 0 and point[1] >= 0 for point in points): return None
```

```
    x = [point[0] for point in points]
    y = [point[1] for point in points]

    lin_x = [log(x[i]) for i in range(n)]
    lin_y = [log(y[i]) for i in range(n)]
    result_values = line_approximation([(lin_x[i], lin_y[i]) for i in range(n)])

    a = exp(result_values['b'])
    b = result_values['a']

    result = {}
    result['a'] = a
    result['b'] = b

    f = lambda i: a * (i ** b)
    result['function'] = f

    result['string_function'] = f"{round(a, 3)}*x^{round(b, 3)}"

    result['variance'] = variance(points, f)

    result['standard_deviation'] = standard_deviation(points, f)

    return result
```

4.5 Экспоненциальная аппроксимация

```
def exp_approximation(points):

    n = len(points)
    x = [point[0] for point in points]
    y = [point[1] for point in points]
    if not all(point[1] >= 0 for point in points): return None

    lin_y = [log(y[i]) for i in range(n)]
    r = line_approximation([(x[i], lin_y[i]) for i in range(n)])

    a = exp(r['b'])
    b = r['a']

    result = {}
    result['a'] = a
    result['b'] = b

    f = lambda i: a * exp(b * i)
    result['function'] = f
    result['string_function'] = f"{round(a, 3)}*e^({round(b, 3)}*x)"
    result['variance'] = variance(points, f)
    result['standard_deviation'] = standard_deviation(points, f)

    return result
```

4.6 Логарифмическая аппроксимация

```
def log_appproximation(points):
```

```
    n = len(points)
    x = [point[0] for point in points]
    if not all(point[0] >= 0 for point in points): return None
    y = [point[1] for point in points]

    lin_x = [log(xi) for xi in x]
    result_values = line_approximation([(lin_x[i], y[i]) for i in range(n)])

    a = result_values['a']
    b = result_values['b']

    result = {}
    result['a'] = a
    result['b'] = b

    f = lambda i: a * log(i) + b
    result['function'] = f
    result['string_function'] = f"{round(a, 3)}*ln(x) + {round(b, 3)}"
    result['variance'] = variance(points, f)
    result['standard_deviation'] = standard_deviation(points, f)

    return result
```

4.7 Вычисление среднеквадратического отклонения и дисперсии

```
def variance(points, f):
    n = len(points)
    x = [dot[0] for dot in points]
    y = [dot[1] for dot in points]

    return sum([(f(x[i]) - y[i]) ** 2 for i in range(n)])


def standard_deviation(points, f):
    n = len(points)
    return sqrt(variance(points, f) / n)
```

5 Примеры и результаты работы программы

5.1 Пример

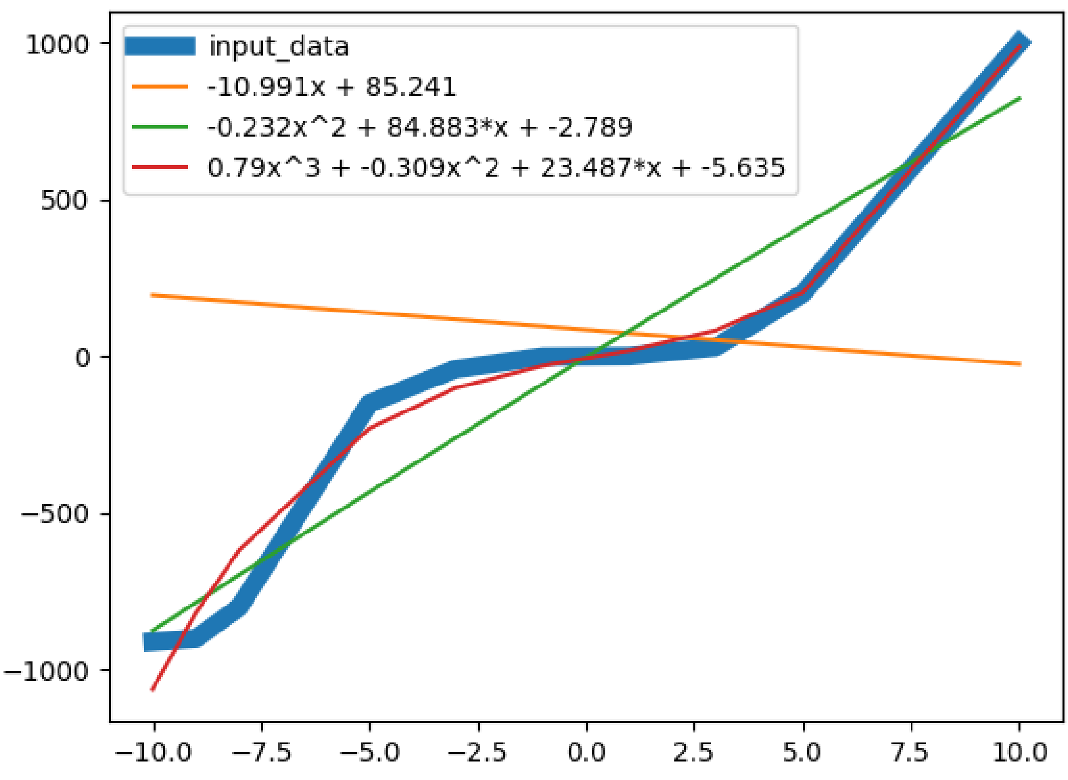
Ввод:

```
Input file content:
-10 -910
-9 -900
-8 -800
-5 -150
-3 -40
-1 -1
0 0
1 1
3 30
5 200
10 1000
```

Вывод:

```
Read data from file? (y/n)y
[(—10.0, —910.0), (—9.0, —900.0), (—8.0, —800.0), (—5.0, —150.0), (—3.0, —40.0), (—1.0, —1.0), (0.0, 0.0), (1.0, 1.0), (3.0, 30.0
[—1061.405, —817.9570000000001, —617.787, —229.54499999999996, —100.20700000000001, —30.220999999999997, —5.635, 18.333, 83.375,
Results for whole methods
line = {'a': —10.991, 'b': 85.241, 'function': <function line_approximation.<locals>.<lambda> at 0x7f2a3209b5b0>, 'string_functio
square = {'a_0': —2.789, 'a_1': 84.883, 'a_2': —0.232, 'function': <function square_approximation.<locals>.<lambda> at 0x7f2a3209b
cube = {'a_0': —5.635, 'a_1': 23.487, 'a_2': —0.309, 'a_3': 0.79, 'function': <function cube_approximation.<locals>.<lambda> at 0
standard_deviation(line) = 643.4562813968723
standard_deviation(square) = 163.39016493156385
standard_deviation(cube) = 83.65915238035821
Best method is cube with standard_deviation 83.65915238035821

a_0 = —5.635
a_1 = 23.487
a_2 = —0.309
a_3 = 0.79
function = <function cube_approximation.<locals>.<lambda> at 0x7f2a3209b6d0>
string_function = 0.79x^3 + —0.309x^2 + 23.487*x + —5.635
variance = 76987.39154699993
standard_deviation = 83.65915238035821
```



6 Выводы

В этой лабораторной работе я научилась апроксимировать входную табличную функцию линейной, полиномиальной 2ой и 3ей степени, экспоненциальной, логарифмической и степенной функциями по методу наименьших квадратов.