Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

По Вычислительной математике

Вариант №10

Выполнил:

Таджеддинов Рамиль Эмильевич

Группа № P3108

Поток № 1.3

Преподаватель:

Санкт-Петербург 2025

Содержание

1 Цель работы

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

2 Порядок выполнения работы

2.1 Вычислительная реализация задачи

Функция в соответствии с вариантом

$$y = \frac{18x}{x^4 + 10} \quad x \in [0, 4] \quad h = 0.4$$

Сформируем таблицу табулирования функции

i	1	2	3	4	5	6	7	8	9	10	11
x_i	0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
y_i	0.000	0.718	1.383	1.789	1.740	1.385	1.001	0.705	0.501	0.364	0.271

2.1.1 Линейная аппроксимация

Найдём линейное приближение данной функции

$$\phi(x) = ax + b$$

Сумма квадратов отклонений запишется следующим образом:

$$S = S(a,b) = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} [\varphi(x_i) - y_i]^2 = \sum_{i=1}^{n} (ax_i + b - y_i)^2 \to \min$$

Для нахождения a и b необходимо найти минимум функции $S(a,b)$.

Необходимое условие существования минимума для функции S:

$$\begin{cases} \dfrac{\partial S}{\partial a} = 0 \\ \\ \dfrac{\partial S}{\partial b} = 0 \end{cases} \quad \text{или} \quad \begin{cases} 2\sum\limits_{i=1}^{n}(ax_i + b - y_i)x_i = 0 \\ 2\sum\limits_{i=1}^{n}(ax_i + b - y_i) = 0 \end{cases}$$

Упростим полученную систему:

$$\begin{cases} a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i \\ a\sum_{i=1}^{n} x_i + bn = \sum_{i=1}^{n} y_i \end{cases}$$

Введем обозначения:

$$SX = \sum_{i=1}^{n} x_i, \quad SXX = \sum_{i=1}^{n} x_i^2, \quad SY = \sum_{i=1}^{n} y_i, \quad SXY = \sum_{i=1}^{n} x_i y_i$$

Получим систему уравнений для нахождения параметров a и b:

$$\begin{cases} aSXX + bSX = SXY \\ aSX + bn = SY \end{cases}$$

из которой по правилу Крамера получаем:

$$\begin{cases} \Delta = SXX \cdot n - SX \cdot SX \\ \Delta_1 = SXY \cdot n - SX \cdot SY \\ \Delta_2 = SXX \cdot SY - SX \cdot SXY \end{cases} \Rightarrow \begin{cases} a = \dfrac{\Delta_1}{\Delta} \\ b = \dfrac{\Delta_2}{\Delta} \end{cases}$$

Поставив соответствующие значения получаем:

$$\begin{cases} a = \dfrac{17.468 \cdot 11 - 22.000 \cdot 9.857}{61.600 \cdot 11 - 22.000 \cdot 22.000} = -0.128 \\ b = \dfrac{61.600 \cdot 9.857 - 22.000 \cdot 17.468}{61.600 \cdot 11 - 22.000 \cdot 22.000} = 1.151 \end{cases}$$

Таким образом аппроксимирующая линейная функция имеет вид:

$$\phi(x) = -0.128x + 1.151$$

i	1	2	3	4	5	6	7	8	9	10	11
x_i	0.0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0

y_i	0.000	0.718	1.383	1.789	1.740	1.385	1.001	0.705	0.501	0.364	0.271
$\varphi(x_i)$	1.151	1.100	1.049	0.998	0.947	0.896	0.845	0.794	0.743	0.692	0.641
$(\varphi(x_i) - y_i)^2$	1.326	0.146	0.112	0.625	0.628	0.239	0.024	0.008	0.058	0.107	0.137

Вычислим среднеквадратичное отклонение для полученной аппроксимации

$$\sigma = \sqrt{\frac{\sum\limits_{i=1}^{n}(\varphi(x_i) - y_i)^2}{n}} = 0.557$$

2.1.2 Квадратичная аппроксимация

Найдём квадратичное приближение данной функции

$$\phi(x) = a_2x^2 + a_1x + a_0$$

Сумма квадратов отклонений запишется следующим образом:

$$S = S(a_0, a_1, a_2) = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} [\varphi(x_i) - y_i]^2 = \sum_{i=1}^{n} (a_2x_i^2 + a_1x_i + a_0 - y_i)^2 \to \min$$

Для нахождения a_0,a_1 и a_2 необходимо найти минимум функции $S(a_0,a_1,a_2)$.

Необходимое условие существования минимума для функции S:

$$\begin{cases} \dfrac{\partial S}{\partial a_0} = 0 \\ \\ \dfrac{\partial S}{\partial a_1} = 0 \\ \\ \dfrac{\partial S}{\partial a_2} = 0 \end{cases} \Rightarrow \begin{cases} \dfrac{\partial S}{\partial a_0} = 2\sum\limits_{i=1}^{n}(a_2x_i^2 + a_1x_i + a_0 - y_i) = 0 \\ \\ \dfrac{\partial S}{\partial a_1} = 2\sum\limits_{i=1}^{n}(a_2x_i^2 + a_1x_i + a_0 - y_i)x_i = 0 \\ \\ \dfrac{\partial S}{\partial a_2} = 2\sum\limits_{i=1}^{n}(a_2x_i^2 + a_1x_i + a_0 - y_i)x_i^2 = 0 \end{cases}$$

Упростим полученную систему:

$$
\begin{cases}
a_0 n + a_1 \sum\limits_{i=1}^{n} x_i + a_2 \sum\limits_{i=1}^{n} x_i^2 = \sum\limits_{i=1}^{n} y_i \\
a_0 \sum\limits_{i=1}^{n} x_i + a_1 \sum\limits_{i=1}^{n} x_i^2 + a_2 \sum\limits_{i=1}^{n} x_i^3 = \sum\limits_{i=1}^{n} x_i y_i \\
a_0 \sum\limits_{i=1}^{n} x_i^2 + a_1 \sum\limits_{i=1}^{n} x_i^3 + a_2 \sum\limits_{i=1}^{n} x_i^4 = \sum\limits_{i=1}^{n} x_i^2 y_i
\end{cases}
$$

Поставив соответствующие значения получаем систему:

$$
\begin{cases}
a_0 \cdot 11 + a_1 \cdot 22 + a_2 \cdot 61.6 = 9.857 \\
a_0 \cdot 22 + a_1 \cdot 61.6 + a_2 \cdot 193.6 = 17.468 \\
a_0 \cdot 61.6 + a_1 \cdot 193.6 + a_2 \cdot 648.525 = 39.046
\end{cases}
$$

из которой по правилу Крамера получаем:

$$
\begin{cases}
\Delta = 4252.424 \\
\Delta_0 = 1564.796 \\
\Delta_1 = 5009,161 \\
\Delta_2 = -1387,957
\end{cases}
\Rightarrow
\begin{cases}
a_0 = \dfrac{\Delta_0}{\Delta} = \dfrac{1564.796}{4252.424} = 0.368 \\
a_1 = \dfrac{\Delta_1}{\Delta} = \dfrac{5009,161}{4252.424} = 1.178 \\
a_2 = \dfrac{\Delta_2}{\Delta} = \dfrac{-1387,957}{4252.424} = -0.326
\end{cases}
$$

Таким образом аппроксимирующая квадратичная функция имеет вид:

$$\phi(x) = -0.326x^2 + 1.178x + 0.368$$

i	1	2	3	4	5	6	7	8	9	10	11

x_i	0.0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
y_i	0.000	0.718	1.383	1.789	1.740	1.385	1.001	0.705	0.501	0.364	0.271
$\phi(x_i)$	0.368	0.787	1.101	1.312	1.417	1.418	1.315	1.107	0.795	0.379	-0.142
$(\phi(x_i) - y_i)^2$	0.135	0.005	0.079	0.228	0.104	0.001	0.099	0.162	0.086	0.000	0.171

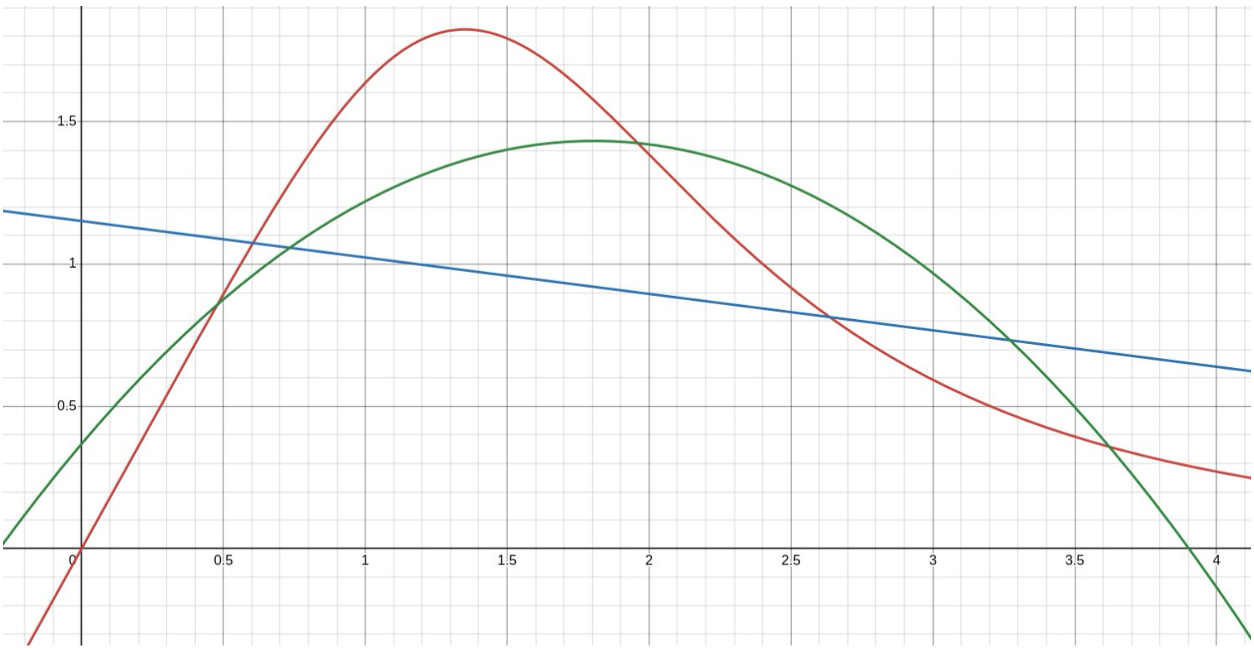
Вычислим среднеквадратичное отклонение для полученной аппроксимации

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(\varphi(x_i) - y_i)^2}{n}} = 0.312$$

2.1.3 Выбор лучшего приближения

Сравнив среднеквадратичные отклонения для линейного и квадратичного приближений становится понятно, что для квадратичной аппроксимации оно меньше, а значит эта аппроксимация является более точной.

2.1.4 Графики функций



2.2 Программная реализация задачи

2.2.1 Листинг программы

```
def linear_approximation(x, y, n): if n < 2:
        raise Exception('Должно быть минимум 2 точки')

    sx = sum(x)
    sxx = sum(xi ** 2 for xi in x) sy = sum(y) sxy = sum(xi
    * yi for xi, yi in zip(x, y))

    try:
        a, b = np.linalg.solve( [
                [sxx, sx],
                [sx, n]
            ],
            [sxy, sy]
        ) except np.linalg.LinAlgError: raise Exception('Не удалось
    подобрать коэффициенты')

    phi = lambda x_: a * x_ + b return phi, (a,

    b)


def square_approximation(x, y, n):
```

```
    if n < 3:
            raise Exception('Должно быть минимум 3 точки')

    sx = sum(x) sxx = sum(xi ** 2 for xi in x)
    sxxx = sum(xi ** 3 for xi in x) sxxxx =
    sum(xi ** 4 for xi in x) sy = sum(y)
    sxy = sum(xi * yi for xi, yi in zip(x, y)) sxxy = sum(xi * xi * yi
    for xi, yi in zip(x, y))

    try:
        a0, a1, a2 = np.linalg.solve( [
                [n, sx, sxx],
                [sx, sxx, sxxx],
                [sxx, sxxx, sxxxx]
            ],
            [sy, sxy, sxxy]
        ) except np.linalg.LinAlgError: raise Exception('Не удалось
    подобрать коэффициенты')

    phi = lambda x_: a2 * x_ ** 2 + a1 * x_ + a0

    return phi, (a0, a1, a2)


def cubic_approximation(xs, ys, n): if n < 4:
            raise Exception('Должно быть минимум 4 точки')

    sx = sum(xs) sxx = sum(xi ** 2 for xi in xs)
    sxxx = sum(xi ** 3 for xi in xs) sxxxx = sum(xi
    ** 4 for xi in xs) sxxxxx = sum(xi ** 5 for xi in
    xs) sxxxxxx = sum(xi ** 6 for xi in xs) sy =
    sum(ys)
    sxy = sum(xi * yi for xi, yi in zip(xs, ys)) sxxy = sum(xi * xi * yi for xi, yi in
    zip(xs, ys)) sxxxy = sum(xi * xi * xi * yi for xi, yi in zip(xs, ys))

    try:
        a0, a1, a2, a3 = np.linalg.solve(
            [
                [n, sx, sxx, sxxx],
                [sx, sxx, sxxx, sxxxx],
                    [sxx, sxxx, sxxxx, sxxxxx],
                    [sxxx, sxxxx, sxxxxx, sxxxxxx]
```

```
            ],
            [sy, sxy, sxxy, sxxxy]
        ) except np.linalg.LinAlgError: raise Exception('Не удалось
    подобрать коэффициенты')

    phi = lambda x_: a3 * x_ ** 3 + a2 * x_ ** 2 + a1 * x_ + a0

    return phi, (a0, a1, a2, a3)


def exponential_approximation(x, y, n): if n < 2:
        raise Exception('Должно быть минимум 2 точки')
    if min(y) <= 0:
                raise ValueError('Аппроксимация возможна только для наборов точек
        ↪у которых y > 0')

    _, (a_, b_) = linear_approximation(x, np.log(y), n)

    a  = a_
    b  = np.exp(b_) phi = lambda x_: b * np.exp(a * x_) return phi,

      (a, b)


def logarithmic_approximation(x, y, n): if n < 2:
        raise Exception('Должно быть минимум 2 точки')
    if min(x) <= 0:
                raise ValueError('Аппроксимация возможна только для наборов точек
        ↪у которых x > 0')

    _, (a_, b_) = linear_approximation(np.log(x), y, n)

    a = a_ b
    = b_

    phi = lambda x_: a * np.log(np.clip(x_, 1e-10, None)) + b return phi, (a, b)


def power_approximation(x, y, n): if n < 2:
        raise Exception('Должно быть минимум 2 точки') if min(x)
    <= 0 or min(y) <= 0:
```

```
            raise ValueError('Аппроксимация возможна только для наборов точек
             ↪у которых x > 0 и y > 0')

        _, (b_, a_) = linear_approximation(np.log(x), np.log(y), n)

        a = np.exp(a_) b =
        b_

        def phi(x_): if b <
            0:
                x_ = np.where(x_ != 0, x_, 1e-10)
            if abs(b) < 1:
                x_ = np.clip(x_, 1e-10, None) return
        a * np.power(x_, b) return phi, (a, b)
```

2.2.2 Результаты выполнения программы

```
Выберите способ ввода:
1 -> Консоль
2 -> Файл
1
Вводите точки, по одной в строке. По окончании ввода введите q
1.2 7.4
2.9 9.5
4.1 11.1
5.5 12.9
6.7 14.6
7.8 17.3
9.2 18.2
10.3 20.7 q
Выберите способ вывода ответа:
1 -> Консоль
2 -> Файл
1

====================================================

Аппроксимирующая функция: Линейная
Функция: phi(x) = 1.454x + 5.291
Среднеквадратичное отклонение: sigma = 0.410
Коэффициент детерминации: R² = 0.991
Мера отклонения: S = 1.346
Коэффициент кореляции Пирсона: r = 0.995

====================================================
```

Аппроксимирующая функция: Полиноминальная 2-й степени
Функция: phi(x) = 0.026x^2 + 1.153x + 5.943
Среднеквадратичное отклонение: sigma = 0.356
Коэффициент детерминации: R^2 = 0.993
Мера отклонения: S = 1.016

==

Аппроксимирующая функция: Полиноминальная 3-й степени
Функция: phi(x) = -0.002x^3 + 0.067x^2 + 0.955x + 6.178
Среднеквадратичное отклонение: sigma = 0.353
Коэффициент детерминации: R^2 = 0.993
Мера отклонения: S = 1.000

==

Аппроксимирующая функция: Экспоненциальная
Функция: phi(x) = 6.840 * e^0.111x
Среднеквадратичное отклонение: sigma = 0.583
Коэффициент детерминации: R^2 = 0.982
Мера отклонения: S = 2.719

==

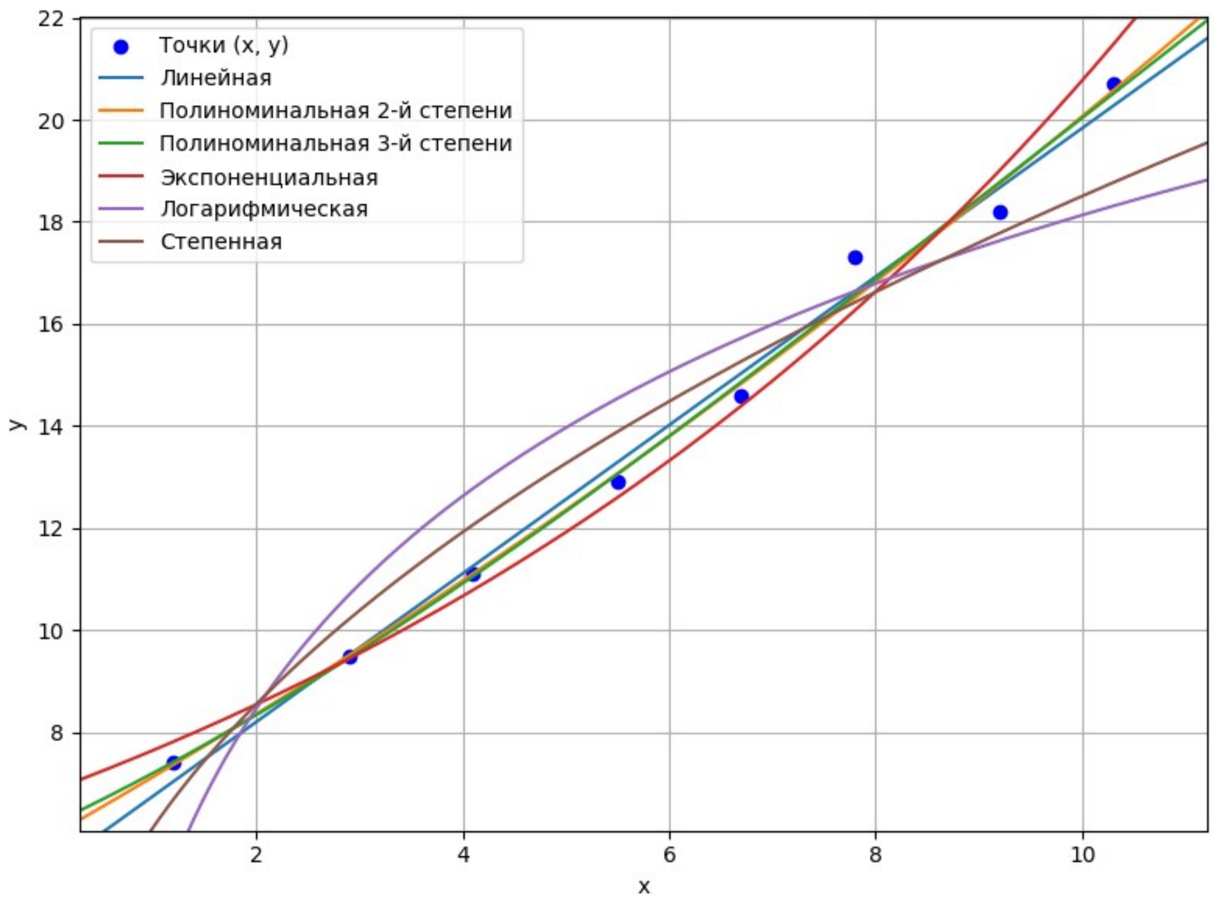
Аппроксимирующая функция: Логарифмическая
Функция: phi(x) = 6.009 * ln(x) + 4.296
Среднеквадратичное отклонение: sigma = 1.529
Коэффициент детерминации: R^2 = 0.873
Мера отклонения: S = 18.693

==

Аппроксимирующая функция: Степенная
Функция: phi(x) = 6.129 * x^0.480
Среднеквадратичное отклонение: sigma = 1.003
Коэффициент детерминации: R^2 = 0.945
Мера отклонения: S = 8.046

==

Лучшая аппроксимирующая функция: Полиноминальная 3-й степени



3 Вывод

В ходе выполнения данной лабораторной работы я научился выполнять аппроксимацию таблично заданных функций, используй линейное, квадратичное, кубическое, логарифмическое, экспоненциальное и показательное приближения. Используя эти знания, я написал программу на языке python, который определяет коэффициенты приближений из описанного выше списка функций. Также для каждой из функций аппроксимации вычисляются среднеквадратичное отклонение, коэффициент детерминации и мера отклонения. Для линейной зависимости также вычисляется коэффициент корреляции Пирсона. Когда все функции определены программа строит график, на котором изображены введённые пользователем точки и графики аппроксимирующих функций.