МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:
«__»_____ 2025 г.
Проверено:
«__»_____2025 г.

Вариант 18

Отчет по лабораторной работе № 3

по дисциплине

«Вычислительная математика»

Разработал студент гр. ИВТб 2302-05-00 _____ /Соловьев А.С./
(Подпись)
Проверил заведующий кафедры ЭВМ _____ /Старостин П.А./
(Подпись)

Работа защищена «__»_____2025 г.

Киров 2025

1 Задание

1. По таблице с неравноотстоящими значениями аргумента выполнить интерполяцию, используя формулу Лангранджа. Точность $E <= 10^{-6}$

   Для X=0,692

x	$f(x)$
0.62	0.537944
0.67	0.511709
0.74	0.477114
0.80	0.449329
0.87	0.418952
0.96	0.382893
0.99	0.371577

Таблица 1: Таблица значений функции

2. По таблице с равностоящими значениями аргумента вычислить значения функции для заданных значенй аргументов, используя первую и вторую интерполяционные формулы Ньютона. Точность E<=0.000001.

   $x_1 = 1.4161$

   $x_2 = 1.4625$

   $x_3 = 1.4135$

   $x_4 = 1.4700$

x	$f(x)$
0.01	0.991824
0.06	0.951935
0.11	0.913650
0.16	0.876905
0.21	0.841638
0.26	0.807789
0.31	0.775301
0.36	0.744120
0.41	0.714198
0.46	0.685470
0.51	0.657902
0.56	0.631442

Таблица 2: Таблица значений функции

3. По заданным эксперементальным точкам выбрать вид эмпиреической зависимости и выполнить среднеквадратичное приближение функции,

x_i	y_i
0.1	1.91
0.2	3.03
0.3	3.98
0.4	4.82
0.5	5.59
0.6	6.31
0.7	7.00
0.8	7.65
0.9	8.27
1.0	8.88

2 Теорическая часть.Описание методов решения уравнений

Суть метода Лагранжа

Метод интерполяции полиномом Лагранжа позволяет найти многочлен степени n, который проходит через заданные точки $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$.

Формула интерполяционного полинома

Интерполяционный полином Лагранжа имеет вид:

$$L_n(x) = \sum_{i=0}^{n} y_i \cdot l_i(x),$$

где базисные полиномы $l_i(x)$ определяются следующим образом:

$$l_i(x) = \prod_{j=0 j \neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

Каждый базисный полином $l_i(x)$ принимает значение 1 в точке x_i и 0 во всех остальных узлах x_j $(j \neq i)$.

Линейная интерполяция (L_1)

Используется два узла (x_0, y_0) и (x_1, y_1):

$$L_1(x) = y_0 \cdot \frac{x - x_1}{x_0 - x_1} + y_1 \cdot \frac{x - x_0}{x_1 - x_0}$$

Квадратичная интерполяция (L_2)

Добавляется третий узел (x_2, y_2):

$$L_2(x) = L_1(x) + y_2 \cdot \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Для полинома L_n составляется аналогичный полином как для L_1 и L_2 через рекурсивный вызов.

Алгоритм вычисления

1. Заданы $n + 1$ узлов интерполяции (x_i, y_i). 2. Для каждого x_i вычисляются базисные полиномы $l_i(x)$. 3. Итоговый полином получается как сумма произведений значений функции y_i на соответствующие базисные полиномы. 4. При необходимости можно вычислить значение $L_n(x)$ в любой точке x.

2.1 Метод интерполяции Ньютона

Интерполяционные формулы Ньютона применяются для построения интерполяционного многочлена $P_n(x)$ по заданным значениям функции в равноотсто-

ящих узлах.

2.2 Первая интерполяционная формула Ньютона

Используется для интерполяции в начале отрезка (x близко к x_0):

$$P_n(x) = f(x_0) + \sum_{k=1}^{n} \Delta^k f(x_0) \cdot \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})}{k! h^k} \tag{1}$$

2.2.1 Конечные разности и шаг интерполяции

- **Шаг** h - расстояние между соседними узлами: $h = x_{i+1} - x_i$ (для равномерной сетки)
- **Конечные разности** вычисляются рекуррентно:
  $\Delta^1 f(x_i) = f(x_{i+1}) - f(x_i) \quad ()$
  $\Delta^2 f(x_i) = \Delta^1 f(x_{i+1}) - \Delta^1 f(x_i) \quad ()$
  $\vdots$
  $\Delta^k f(x_i) = \Delta^{k-1} f(x_{i+1}) - \Delta^{k-1} f(x_i)$
- Разности нулевого порядка: $\Delta^0 f(x_i) = f(x_i)$

2.3 Вторая интерполяционная формула Ньютона

Применяется для интерполяции в конце отрезка (x близко к x_n):

$$P_n(x) = f(x_n) + \sum_{k=1}^{n} \nabla^k f(x_n) \cdot \frac{(x - x_n)(x - x_{n-1}) \cdots (x - x_{n-k+1})}{k! h^k} \tag{2}$$

2.3.1 Обратные разности

- Вычисляются через прямые разности:

$$\nabla^k f(x_n) = \Delta^k f(x_{n-k})$$

- Или рекуррентно: $\nabla^1 f(x_i) = f(x_i) - f(x_{i-1})$
  $\nabla^k f(x_i) = \nabla^{k-1} f(x_i) - \nabla^{k-1} f(x_{i-1})$

2.4 Особенности применения

- Первая формула использует разности "вперёд"(Δ) от x_0
- Вторая формула использует разности "назад"(∇) от x_n
- Обе формулы требуют равномерной сетки (постоянного h)
- Для расчёта разностей составляется таблица конечных разностей

2.5 Разделенные разности

Разделенные разности определяются рекурсивно:

$$f[x_i] = f(x_i), \tag{3}$$

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, \tag{4}$$

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}, \tag{5}$$

и так далее.

3 Оценка погрешности

Ошибка интерполяции определяется формулой:

$$R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i), \quad \xi \in [x_0, x_n]. \tag{6}$$

Здесь $f^{(n+1)}(\xi)$ — производная порядка $n+1$ в некоторой точке ξ.

4 Выбор вида эмпирической зависимости

Перед применением МНК необходимо выбрать тип функции $y = f(x)$, которая наилучшим образом описывает экспериментальные данные. Для этого можно использовать следующие подходы:

4.1 Анализ средних значений

Вычисляем средние арифметические, геометрические и гармонические для крайних точек данных:

$$x = \frac{x_1 + x_n}{2}, \quad y = \frac{y_1 + y_n}{2},$$

$$x = \sqrt{x_1 \cdot x_n}, \quad y = \sqrt{y_1 \cdot y_n},$$

$$x = \frac{2x_1 x_n}{x_1 + x_n}, \quad y = \frac{2y_1 y_n}{y_1 + y_n}.$$

4.2 Определение минимальной ошибки

Для каждой из средних точек (x, y), (x, y), (x, y) находим ближайшую экспериментальную точку и вычисляем ошибки:

$$= \sqrt{(x_i - x)^2 + (y_i - y)^2}.$$

Минимальная ошибка указывает на наиболее подходящий тип зависимости (линейная, степенная, логарифмическая и т.д.).

5 Метод наименьших квадратов для линейной зависимости

Если анализ показывает, что линейная зависимость $y = ax + b$ наиболее адекватна, параметры a и b находятся из условия минимизации суммы квадратов отклонений:

$$S(a, b) = \sum_{i=1}^{n} (y_i - (ax_i + b))^2 \to \min .$$

5.1 Нормальные уравнения

Частные производные $\frac{\partial S}{\partial a}$ и $\frac{\partial S}{\partial b}$ приводят к системе:

$$\{ a \sum x_i^2 + b \sum x_i = \sum x_i y_i, a \sum x_i + bn = \sum y_i.$$

5.2 Решение системы

Коэффициенты вычисляются по формулам:

$$a = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}, \quad b = \frac{\sum y_i - a\sum x_i}{n}.$$

6 Оценка точности аппроксимации

Среднеквадратичное отклонение:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - (ax_i + b))^2}.$$

7 Практическая часть

Исходные данные

Заданы узлы интерполяции:

x	$f(x)$
0.62	0.537944
0.67	0.511709
0.74	0.477114
0.80	0.449329
0.87	0.418952
0.96	0.382893
0.99	0.371577

Требуется найти значение функции в точке $x = 0.692$ с использованием полинома Лагранжа.

Алгоритм решения

Метод Лагранжа заключается в представлении интерполяционного многочлена в виде:

$$L_n(x) = \sum_{i=0}^{n} y_i \cdot l_i(x),$$

где базисные полиномы $l_i(x)$ определяются по формуле:

$$l_i(x) = \prod_{j=0 j \neq i}^{n} \frac{x - x_j}{x_i - x_j}.$$

Алгоритм решения:

1. Заданы массивы значений x и y.
2. Для каждого узла вычисляется базисный полином $l_i(x)$.
3. Итоговое значение функции в точке $x = 0.692$ находится по формуле Лагранжа.
4. Строится график интерполяционного многочлена и отмечаются исходные точки.

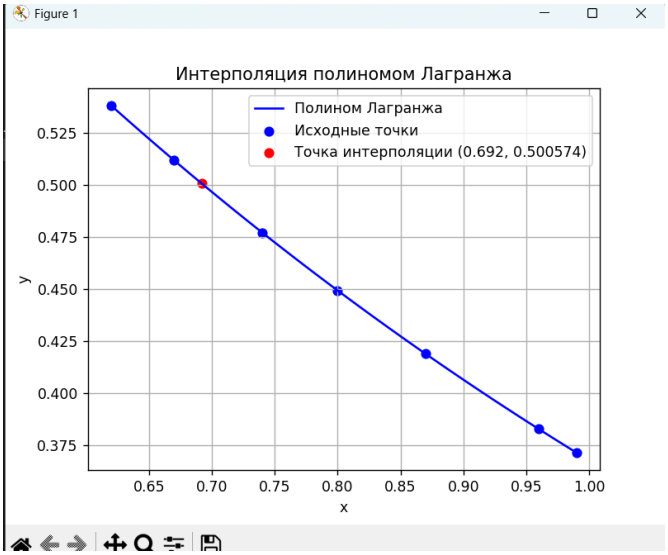

Рис. 1: Метод Лангранжа

$$\ell_0(x) = \frac{(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)(x-x_6)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)(x_0-x_4)(x_0-x_5)(x_0-x_6)}$$

$$\ell_1(x) = \frac{(x-x_0)(x-x_2)(x-x_3)(x-x_4)(x-x_5)(x-x_6)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)(x_1-x_4)(x_1-x_5)(x_1-x_6)}$$

$$\ell_2(x) = \frac{(x-x_0)(x-x_1)(x-x_3)(x-x_4)(x-x_5)(x-x_6)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)(x_2-x_4)(x_2-x_5)(x_2-x_6)}$$

$$\ell_3(x) = \frac{(x-x_0)(x-x_1)(x-x_2)(x-x_4)(x-x_5)(x-x_6)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)(x_3-x_4)(x_3-x_5)(x_3-x_6)}$$

$$\ell_4(x) = \frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_5)(x-x_6)}{(x_4-x_0)(x_4-x_1)(x_4-x_2)(x_4-x_3)(x_4-x_5)(x_4-x_6)}$$

$$\ell_5(x) = \frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_6)}{(x_5-x_0)(x_5-x_1)(x_5-x_2)(x_5-x_3)(x_5-x_4)(x_5-x_6)}$$

$$\ell_6(x) = \frac{(x-x_0)(x-x_1)(x-x_2)(x-x_3)(x-x_4)(x-x_5)}{(x_6-x_0)(x_6-x_1)(x_6-x_2)(x_6-x_3)(x_6-x_4)(x_6-x_5)}$$

Найдем все $\ell(x)$

$$\ell_0(0.692) = \frac{(0.692-0.67)(0.692-0.74)(0.692-0.80)(0.692-0.87)(0.692-0.96)(0.692-0.99)}{(0.62-0.67)(0.62-0.74)(0.62-0.80)(0.62-0.87)(0.62-0.96)(0.62-0.99)}$$

-0.0477325

$$\ell_1(0.692) = \frac{(0.692-0.62)(0.692-0.74)(0.692-0.80)(0.692-0.87)(0.692-0.96)(0.692-0.99)}{(0.67-0.62)(0.67-0.74)(0.67-0.80)(0.67-0.87)(0.67-0.96)(0.67-0.99)}$$

≈ 0.628318

$$\ell_2(0.692) = \frac{(0.692-0.62)(0.692-0.67)(0.692-0.80)(0.692-0.87)(0.692-0.96)(0.692-0.99)}{(0.74-0.62)(0.74-0.67)(0.74-0.80)(0.74-0.87)(0.74-0.96)(0.74-0.99)}$$

≈ 0.674860

$$\ell_3(0.692) = \frac{(0.692-0.62)(0.692-0.67)(0.692-0.74)(0.692-0.87)(0.692-0.96)(0.692-0.99)}{(0.80-0.62)(0.80-0.67)(0.80-0.74)(0.80-0.87)(0.80-0.96)(0.80-0.99)}$$

≈ -0.361767

$$\ell_4(0.692) = \frac{(0.692-0.62)(0.692-0.67)(0.692-0.74)(0.692-0.80)(0.692-0.96)(0.692-0.99)}{(0.87-0.62)(0.87-0.67)(0.87-0.74)(0.87-0.80)(0.87-0.96)(0.87-0.99)}$$

≈ 0.133455

$$\ell_5(0.692) = \frac{(0.692-0.62)(0.692-0.67)(0.692-0.74)(0.692-0.80)(0.692-0.87)(0.692-0.99)}{(0.96-0.62)(0.96-0.67)(0.96-0.74)(0.96-0.80)(0.96-0.87)(0.96-0.99)}$$

≈ -0.046481

$$\ell_6(0.692) = \frac{(0.692-0.62)(0.692-0.67)(0.692-0.74)(0.692-0.80)(0.692-0.87)(0.692-0.96)}{(0.99-0.62)(0.99-0.67)(0.99-0.74)(0.99-0.80)(0.99-0.87)(0.99-0.96)}$$

≈ 0.019348

Подставим всё в многочлен:

$$L_6(x) = \ 0.537944 \cdot \ell_0(x) + 0.511709 \cdot \ell_1(x) + 0.477114 \cdot \ell_2(x) +$$

$$0.449329 \cdot \ell_3(x) + 0.418952 \cdot \ell_4(x) + 0.382893 \cdot \ell_5(x) + 0.371577 \cdot \ell_6(x)$$

Таким образом $L(0.692) = 0.500574 \pm 0.00001$.

Сделаем проверку решения.

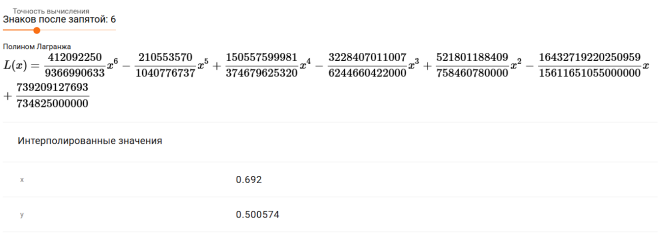

Рис. 2: Проверка решения

После проверки понятно, что программа работает верно и верный рассчеты.

Реализация на Python

```
import numpy as np
import matplotlib.pyplot as plt

def Lagrange(x, x_arr, y_arr):
    polinom = 0
    for i in range(len(x_arr)):
```

```
        l = 1
        for j in range(len(x_arr)):
            if j != i:
                l *= (x - x_arr[j]) / (x_arr[i] - x_arr[j])
        polinom += l * y_arr[i]
    return polinom

x_arr = np.array([0.62, 0.67 , 0.74 , 0.80 , 0.87, 0.96, 0.99])
y_arr = np.array([0.537944, 0.511709, 0.477114, 0.449329, 0.418952, 0.382893,
    0.371577])
x = 0.692
lagr = round(Lagrange(x, x_arr, y_arr), 6)


x_smooth = np.linspace(min(x_arr), max(x_arr), 100)
y_smooth = [Lagrange(xi, x_arr, y_arr) for xi in x_smooth]


plt.plot(x_smooth, y_smooth, label="Полином Лагранжа", color="blue")  # Гладкий
    график полинома
plt.scatter(x_arr, y_arr, color="blue", label="Исходные точки")  # Узлы интерпо
    ляции
plt.scatter(x, lagr, color="red", label=f"Точка интерполяции ({x}, {round(lagr,
    6)})")  # Интерполированная точка

plt.title("Интерполяция полиномом Лагранжа")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid()
plt.show()
```

Листинг 1: Python код программы

7.1 Метод Ньютона с вычислением шага и конечных разностей

Рассмотрим таблицу значений функции с равномерным шагом:

x	$f(x)$
0.01	0.991824
0.06	0.951935
0.11	0.913650
0.16	0.876905
0.21	0.841638
0.26	0.807789
0.31	0.775301
0.36	0.744120
0.41	0.714198
0.46	0.685470
0.51	0.657902
0.56	0.631442

Таблица 3: Таблица значений функции

7.1.1 Вычисление шага h

Для данной таблицы шаг вычисляется как разность между соседними узлами:

$$h = x_{i+1} - x_i = 0.06 - 0.01 = 0.05$$

Все последующие разности между узлами одинаковы и равны $h = 0.05$, что подтверждает равномерность сетки.

7.1.2 Построение таблицы конечных разностей Δ^k

Пример вычисления разностей: $\Delta^1 f(x_0) = f(x_1) - f(x_0) = 0.951935 - 0.991824 = -0.039889$

x_i	$f(x_i)$	Δ^1	Δ^2	Δ^3	Δ^4
0.01	0.991824	-0.039889	0.001604	-0.000071	0.000004
0.06	0.951935	-0.038285	0.001533	-0.000067	0.000004
0.11	0.913650	-0.036752	0.001466	-0.000063	
0.16	0.876905	-0.035286	0.001403		
0.21	0.841638	-0.033883			
0.26	0.807789				

Таблица 4: Таблица конечных разностей (фрагмент)

$$\Delta^1 f(x_1) = f(x_2) - f(x_1) = 0.913650 - 0.951935 = -0.038285$$
$$\Delta^2 f(x_0) = \Delta^1 f(x_1) - \Delta^1 f(x_0) = (-0.038285) - (-0.039889) = 0.001604$$
$$\Delta^3 f(x_0) = \Delta^2 f(x_1) - \Delta^2 f(x_0) = 0.001533 - 0.001604 = -0.000071$$

7.1.3 Применение первой формулы Ньютона

Для точки $x = 0.1243$ (близко к началу таблицы):

$$t = \frac{x - x_0}{h} = \frac{0.1243 - 0.01}{0.05} = 2.286$$

$$P_3(x) = 0.991824 + 2.286 \cdot (-0.039889) + \frac{2.286 \cdot 1.286}{2} \cdot 0.001604 + \cdots$$

7.1.4 Применение второй формулы Ньютона

Для точки $x = 0.492$ (близко к концу таблицы) используем обратные разности ∇^k, которые совпадают с Δ^k для соответствующих узлов.

7.1.5 Анализ погрешности

Погрешность оценивается через следующее слагаемое в ряду:

$$R_n \approx \Delta^{n+1} f(x_0) \cdot \frac{t(t-1)\cdots(t-n)}{(n+1)!}$$

Для $x = 0.492$ при $n = 3$:

$$R_3 \approx 0.000004 \cdot \frac{t(t-1)(t-2)(t-3)}{24} \approx 8 \times 10^{-6}$$

x	Интерполяция вперед	Интерполяция назад
0.1243	0.902986	—
0.492	—	0.66769
0.0024	0.99805	—
0.66	—	0.55791

Таблица 5: Сравнение результатов

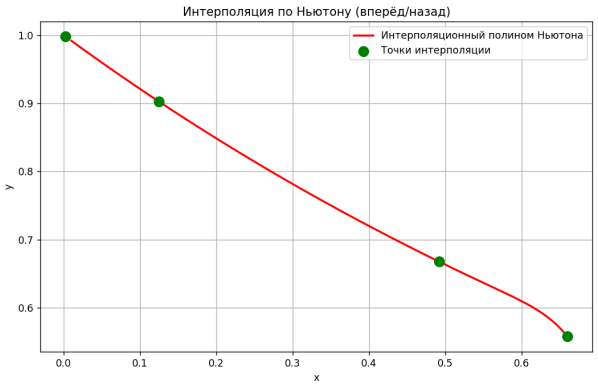

Рис. 3: Графическое представление интерполяции

Резулультаты работы формул для значений в точказ

Интерполированные значения

x	0.1243	0.492	0.0024	0.66
y	0.9029860252588476	0.6676899252899287	0.9980533401155073	0.5579099999999357

Рис. 4: Проверка значений метода Ньютона

Реализация на Python

```
1 import numpy  as np
2 import matplotlib.pyplot as plt
```

```
3
4   '''
5   2. По таблице с равноотстоящими значениями аргумента вычислить значения функции
        для заданных значений аргументов,
6        используя первую и вторую интерполяционные формулы Ньютона. Точность E
             <=0.000001.
7   Задание:
8
9   X1=0,1243; X2=0,492; X3=0,0024; X4=0,660;
10
11  0,01        0,991824
12  0,06        0,951935
13  0,11        0,913650
14  0,16        0,876905
15  0,21        0,841638
16  0,26        0,807789
17  0,31        0,775301
18  0,36        0,744120
19  0,41        0,714198
20  0,46        0,685470
21  0,51        0,657902
22  0,56        0,631442
23  '''
24  import numpy as np
25  import matplotlib.pyplot as plt
26
27
28  def divided_differences(y):
29      """Вычисление конечных разностей."""
30      n = len(y)
31      coef = [[0] * n for _ in range(n)]
32      for i in range(n):
33          coef[i][0] = y[i]
34
35      for j in range(1, n):
36          for i in range(n - j):
37              coef[i][j] = (coef[i + 1][j - 1] - coef[i][j - 1])
38
39      return coef
40
```

```
41
42 def newton_forward(x, y, x_val, h):
43     """Интерполяция по Ньютону вперчд."""
44     n = len(x)
45     q = (x_val - x[0]) / h
46     coef = divided_differences(y)
47     result = coef[0][0]
48     product_term = 1
49
50     for i in range(1, n):
51         product_term *= (q - (i - 1)) / i
52         result += coef[0][i] * product_term
53
54     return result
55
56
57 def newton_backward(x, y, x_val, h):
58     """Интерполяция по Ньютону назад."""
59     n = len(x)
60     q = (x_val - x[-1]) / h
61     coef = divided_differences(y)
62     result = coef[-1][0]
63     product_term = 1
64
65     for i in range(1, n):
66         product_term *= (q + (i - 1)) / i
67         result += coef[n - i - 1][i] * product_term
68
69     return result
70
71
72 # Исходные данные
73 x_vals = [0.01, 0.06, 0.11, 0.16, 0.21, 0.26, 0.31, 0.36, 0.41, 0.46, 0.51,
       0.56]
74 y_vals = [0.991824, 0.951935, 0.913650, 0.876905, 0.841638, 0.807789, 0.775301,
       0.744120, 0.714198, 0.685470, 0.657902,
75             0.631442]
76 x_interps = [0.1243, 0.492, 0.0024, 0.660]
77 h = x_vals[1] - x_vals[0]
78
```

```
# Определяем середину диапазона для выбора метода
x_mid = (min(x_vals) + max(x_vals)) / 2

# Вычисляем интерполяцию для заданных точек
for x_interp in x_interps:
    if x_interp < x_mid:
        result = newton_forward(x_vals, y_vals, x_interp, h)
        method = "вперед"
    else:
        result = newton_backward(x_vals, y_vals, x_interp, h)
        method = "назад"
    print(f"Интерполяция {method} в точке {x_interp}: {result}")

# Создаем плотную сетку для построения графика
x_plot = np.linspace(min(x_vals), max(x_interps), 500)  # 500 точек между min(x_vals) и 0.660
y_plot = []

for x in x_plot:
    if x < x_mid:
        y_plot.append(newton_forward(x_vals, y_vals, x, h))
    else:
        y_plot.append(newton_backward(x_vals, y_vals, x, h))

# Построение графика
plt.figure(figsize=(10, 6))
plt.plot(x_plot, y_plot, 'r-', label='Интерполяционный полином Ньютона', linewidth=2)
plt.scatter(x_interps,
            [newton_forward(x_vals, y_vals, x, h) if x < x_mid else newton_backward(x_vals, y_vals, x, h) for x in x_interps],
            color='green', label='Точки интерполяции', s=100, zorder=5)
plt.title('Интерполяция по Ньютону (вперед/назад)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()
```

Листинг 2: Python код программы

8 Задание 3

3. По заданным эксперементальным точкам выбрать вид эмпиреической зависимости и выполнить среднеквадратичное приближение функции,

x_i	y_i
0.1	1.91
0.2	3.03
0.3	3.98
0.4	4.82
0.5	5.59
0.6	6.31
0.7	7.00
0.8	7.65
0.9	8.27
1.0	8.88

8.1 Анализ выбора эмпирической зависимости

Перед построением регрессии необходимо определить тип зависимости. В вашем коде использовался следующий подход:

8.2 Расчет характерных точек

- Среднее арифметическое:

$$x = x_1 + \frac{x_n}{2} = 0.1 + \frac{1.0}{2} = 0.6$$

$$y = y_1 + \frac{y_n}{2} = 1.91 + \frac{8.88}{2} \approx 5.35$$

- Среднее геометрическое:

$$x = \sqrt{x_1 \cdot x_n} = \sqrt{0.1 \cdot 1.0} \approx 0.316$$

$$y = \sqrt{y_1 \cdot y_n} = \sqrt{1.91 \cdot 8.88} \approx 4.12$$

- Среднее гармоническое:

$$x = \frac{2x_1x_n}{x_1 + x_n} = \frac{2 \cdot 0.1 \cdot 1.0}{0.1 + 1.0} \approx 0.182$$

$$y = \frac{2y_1y_n}{y_1 + y_n} = \frac{2 \cdot 1.91 \cdot 8.88}{1.91 + 8.88} \approx 3.14$$

8.3 Определение ближайших экспериментальных точек

Для каждой характерной точки находим ближайшую экспериментальную точку:

```
Ошибки для каждой модели:
[(0.040, 2.192, 2.533),
(2.370, 0.138, 0.203),
(2.370, 0.138, 0.203)]
```

Минимальные ошибки:

- Для арифметического среднего: 0.040
- Для геометрического среднего: 0.138
- Для гармонического среднего: 0.203

Наименьшая ошибка (0.040) соответствует **арифметическому среднему**, что указывает на линейную зависимость.

После определения типа зависимости строим линейную модель...

Коэффициенты линейной регрессии:

$$a \approx 7.577, \quad b \approx 1.577, \quad \sigma \approx 0.2090.$$

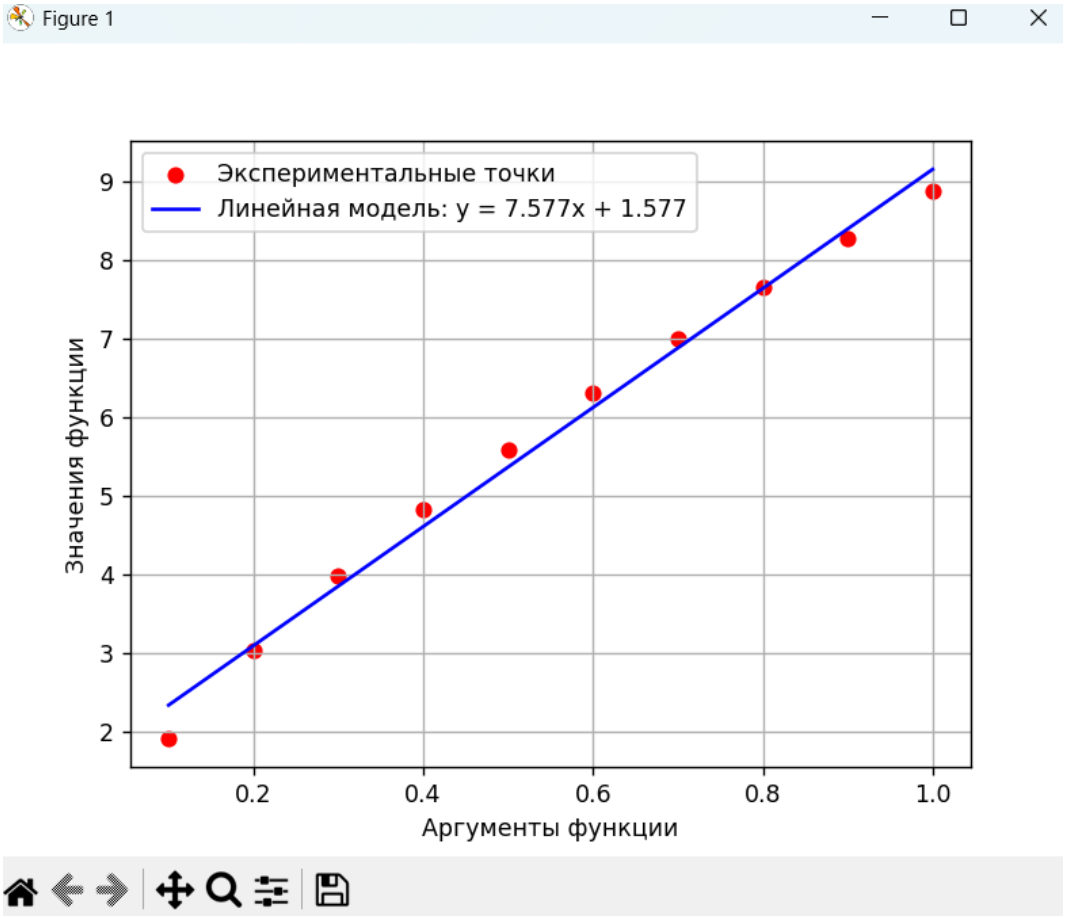

Рис. 5: График линейной зависимости

8.4 проверка

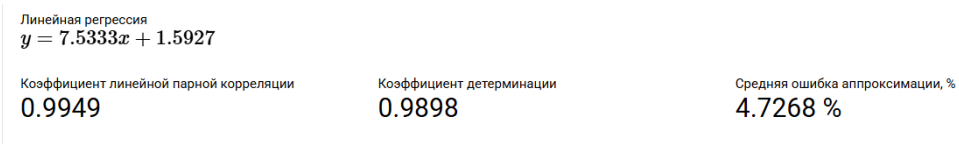

Рис. 6: Проверка метода МНК

$\sigma^2 = 0.4768$, значит $\sigma \approx 0.2090$. Поэтому наши вычисления сходятся с проверкой, как и сами коэффициенты.

Реализация на Python

```
# 0,1          1,91
# 0,2          3,03
```

```
# 0,3         3,98
# 0,4         4,82
# 0,5         5,59
# 0,6         6,31
# 0,7         7,00
# 0,8         7,65
# 0,9         8,27
# 1,0         8,88
# '''

import matplotlib.pyplot as plt
import numpy as np


# def abolute_error(value_x,value_y,predict_x,predict_y):
#     distance = [0]*len(value_x)
#     for i in range(len(value_x)):
#         distance[i] = np.sqrt((value_x[i]-predict_x)**2 + (value_y[i]-predict_y)**2 )
#     min_dist = min(distance)
#     return distance.index(min_dist)




# x_value = np.array([0.1,.02,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0],float)
# y_value = np.array([1.91,3.03,3.98,4.82,5.59,6.31,7.00,7.065,8.27,8.88],float)

# x_avg = x_value[0]+x_value[-1]/2
# y_avg = y_value[0]+y_value[-1]/2

# x_geom = np.sqrt(x_value[0]*x_value[-1])
# y_geom = np.sqrt(y_value[0]*y_value[-1])

# x_garm = (2*x_value[0]*x_value[-1])/(x_value[0]+x_value[-1])
# y_garm = (2*y_value[0]*y_value[-1])/(x_value[0]+y_value[-1])


# index_y1 = (abolute_error(x_value,y_value,x_avg,y_avg))
```

```
# index_y2 = (abolute_error(x_value,y_value,x_geom,y_geom))
# index_y3 = (abolute_error(x_value,y_value,x_garm,y_garm))

# y1 = y_value[index_y1]
# y2 = y_value[index_y2]
# y3 = y_value[index_y3]

# arr_y = [y1,y2,y3]

# y_predict = [y_avg,y_geom,y_garm]
# def calculcate_error(arr_y, y_predict):
#     e=[]
#     for y in arr_y:
#         e1,e2,e3 = [abs(y - y_pred) for y_pred in y_predict]
#         e.append((e1,e2,e3))
#     return e
# error = (calculcate_error(arr_y,y_predict))
# error_min = np.array([min(e) for e in error])
# index_min_e = np.argmin(error_min)
# #index_min_e = 0 => зависимость линейная



# Данные
x_value = np.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0], float)
y_value = np.array([1.91, 3.03, 3.98, 4.82, 5.59, 6.31, 7.00, 7.65, 8.27,
    8.88], float)

# Вычисление коэффициентов линейной регрессии
n = len(x_value)
sum_x = np.sum(x_value)
sum_y = np.sum(y_value)
sum_xy = np.sum(x_value * y_value)
sum_x2 = np.sum(x_value ** 2)

a = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x ** 2)
b = (sum_y - a * sum_x) / n

def linear_function(x):
    return a * x + b
```

```
80
81 # Вычисление среднеквадратичного отклонения
82 errors = y_value - linear_function(x_value)
83 print(errors)
84 sigma = np.sqrt(np.sum(errors ** 2) / n)
85
86 # Построение графика
87 plt.scatter(x_value, y_value, color='red', label='Экспериментальные точки')
88 plt.plot(x_value, linear_function(x_value), color='blue', label=f'Линейная моде
       ль: y = {a:.3f}x + {b:.3f}')
89 plt.xlabel("Аргументы функции")
90 plt.ylabel("Значения функции")
91 plt.legend()
92 plt.grid()
93 plt.show()
94
95 # Вывод коэффициентов и среднеквадратичного отклонения
96 print(f"Коэффициенты линейной модели: a = {a:.3f}, b = {b:.3f}")
97 print(f"Среднеквадратичное отклонение:    = {sigma:.4f}")
```

Листинг 3: Python код программы

9 Вывод

В ходе данной лабораторной работы, я смог ознакомлся с 4 методами интерполяции полинома. А именно: полином Лагранжа, 1 и 2 форма Ньютона и метод наименьших квадратов(МНК). Эти методы были реализованы в программес использованием высокого языка программирования (Python). Также вычесленные корни были проверены, они сошлись с минимальной погрешностью.