МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра вычислительной математики

Отчёт

Лабораторная работа №1-4

Вариант № 12

Снежко Льва Владимировича
студента 3 курса, 3 группы
специальности «Информатика»
дисциплина «Численные методы»
Преподаватель: Будник А.М.

Минск, 2025

Постановка задачи

Рассмотрим набор различный точек на отрезке [a, b]:

$$x_0 < x_1 < \cdots < x_n \text{ , } x_i \in [a, b], \text{ i} = \overline{0, n}.$$

$$y_i = \mathrm{f}(x_i), \text{ i} = \overline{0, n}$$

Требуется восстановить значение функции $\mathrm{f}(x) = \alpha_j e^x + (1 - \alpha_j)\mathrm{sinx}$ в других точках $x^*, x^{**}, x^{***} \in [a, b]$:

$$[\mathrm{a, b}] = [\alpha_j,\ 1\ +\ \alpha_j], \text{ где}$$

$\alpha_j\ =\ 0.1\ +\ 0.05\ \cdot$ j = [j = 12 – номер варианта] = $0.1\ +\ 0.05\ \cdot\ 12\ =\ 0.7$.

Таким образом имеем:

$$[\mathrm{a, b}] = [0.7, 1.7]$$

$$\mathrm{f}(x) = 0.7e^x + 0.3\mathrm{sinx}$$

Необходимо интерполировать эту функцию:

1. Многочленом Ньютона $P_{10}(x)$ на равномерной сетке;
2. Многочленом Ньютона $P_{10}(x)$ на Чебышёвской сетке;
3. Кубическим сплайном;
4. Методом наименьших квадратов (полином 5 степени).

Для каждого из методов необходимо:

- Вычислить значения интерполяционного многочлена в точках x^*, x^{**}, x^{***};
- Вычислить или оценить остаток интерполирования в точках x^*, x^{**}, x^{***};
- Вычислить истинную погрешность $r_{n,\text{ист.}}$;
- Сравнить и проанализировать полученные результаты.

Алгоритм решения

1. Интерполирование многочленом Ньютона на равномерной сетке

Пусть x_i заданы равномерно на отрезке [0.7,1.7]:

$$x_i = \alpha_i + \mathrm{ih},\ где$$

$$\mathrm{h} = \frac{1}{n},\ \mathrm{i} = \overline{0,\ n},\ \mathrm{n} = 10,\ т.е.$$

$\mathrm{h} = 0.1$

$x_i = 0.7 + 0.1 \cdot i,\ ,\ \mathrm{i} = \overline{0,\ 10}$

3 точки восстановления:

$$x^{*} = x_0 + \frac{2}{3}\mathrm{h} = 0.5 + \frac{2}{3} \cdot 0.1 = 0.766667,$$

$$x^{**} = x_{\frac{n}{2}} + \frac{1}{2}\mathrm{h} = x_5 + \frac{1}{2}\mathrm{h} = 1 + 0.5 \cdot 0.1 = 1.25,$$

$$x^{***} = x_n - \frac{1}{3}\mathrm{h} = 1.5 - \frac{1}{3} \cdot 0.1 = 1.666667$$

Интерполяционный многочлен в форме Ньютона будем искать в следующем виде:

$$P_n(x) = \sum_{i=0}^{n} \alpha_i \omega_i(x) = \alpha_0 + \alpha_1 (x - x_0) + \alpha_2 (x - x_0)(x - x_1) \dots \alpha_n (x - x_0) \dots (x - x_{n-1}),$$

где $\omega_i(x) = \prod_{j=0}^{i-1}(x - x_j),\ i = \overline{0,10}$

Коэффициенты ИМ удобно вычислять по определению разделённых разностей:

$$\alpha_i = \mathrm{f}[x_0, \dots, x_i],\ i = \overline{0,10}$$

$$\mathrm{f}[\mathrm{x}_0, \dots, \mathrm{x}_\mathrm{i}] = \frac{\mathrm{f}[\mathrm{x}_1, \dots, \mathrm{x}_\mathrm{i}] - \mathrm{f}[\mathrm{x}_0, \dots, \mathrm{x}_{\mathrm{i}-1}]}{\mathrm{x}_\mathrm{i} - \mathrm{x}_0}, \quad \mathrm{i} = \overline{1, 10};$$

$$\mathrm{f}[\mathrm{x}_\mathrm{j}] = \mathrm{f}(\mathrm{x}_\mathrm{j}), \quad \mathrm{j} = \overline{0, 10}$$

путём построения треугольной таблицы следующего вида:

$$
\begin{array}{ccccc}
x_0, f[x_0] & & & & \\
 & f[x_0, x_1] & & & \\
x_1, f[x_1] & & f[x_0, x_1, x_2] & & \\
 & f[x_1, x_2] & & \ddots & \\
x_2, f[x_2] & & f[x_0, x_1, x_2] & f[x_0, \ldots, x_{n-1}] & \\
\vdots & f[x_2, x_3] & \vdots & & f[x_0, \ldots, x_n] \\
\vdots & \vdots & \vdots & f[x_1, \ldots, x_n] & \\
\vdots & \vdots & \vdots & \iddots & \\
x_{n-1}, f[x_{n-1}] & \vdots & f[x_{n-2}, x_{n-1}, x_n] & & \\
 & f[x_{n-1}, x_n] & & & \\
x_n, f[x_n] & & & &
\end{array}
$$

Остаток интерполирования в точках x^*, x^{**}, x^{***} вычислим по следующей формуле:

$$r_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\,\omega_{n+1}(x), \quad x \in \{x^*,\ x^{**},\ x^{***}\}$$

Истинную погрешность вычислим так:

$$r_{n,\text{ист}}(x^*) = f(x^*) - P_n(x^*)$$

И оценим по формуле

$$||r_n|| \le \frac{\left|\left|f^{(n+1)}\right|\right|}{(n+1)!}\,||\omega_{n+1}||$$

Листинг кода

```
%pip install tabulate
%pip install sympy
import numpy as np
import math
from tabulate import tabulate
import matplotlib.pyplot as plt
import pandas as pd
import sympy as sp

j = 12
n = 10
alpha_j = 0.1 + 0.05*j
h = 1/n
def f(x):
    return alpha_j * math.exp(x) + (1 - alpha_j) * math.sin(x)

# Шаг 1. Построим исходную таблицу
x_vals = np.array([alpha_j + i * h for i in range(n+1)])
f_vals = np.array([f(x_) for x_ in x_vals])
x_star = np.array([x_vals[0] + 2*h/3, x_vals[len(x_vals) // 2] + h/2, x_vals[-1] - h/3])
f_star = np.array([f(x_) for x_ in x_star])
print(tabulate(zip(x_vals, f_vals), headers=['x', 'f(x)']))
print('\nСпециальные точки')
print(tabulate(zip(x_star, f_star), headers=['x*', 'f(x*)']))



# Таблица значений функции
table = pd.DataFrame({"x_i": x_vals, "f(x_i)": f_vals})
table_transposed = table.T

# Точки для проверки интерполяции
x_star1 = x_star[0]
x_star2 = x_star[1]
x_star3 = x_star[2]

f_x_star = f_star[0]
f_x_star2 = f_star[1]
f_x_star3 = f_star[2]

def compute_newton_coefficients(x_vals, y_vals):
    """
```

```
    Возвращает список коэффициентов интерполяционного многочлена Ньютона
    с использованием рекурсивного определения разделённых разностей.
    """
    n = len(x_vals)
    # Создаём таблицу размером n x n
    dd_table = [y_vals.copy()]  # f[x_i]

    for level in range(1, n):
        prev_column = dd_table[-1]
        curr_column = []
        for i in range(n - level):
            numerator = prev_column[i + 1] - prev_column[i]
            denominator = x_vals[i + level] - x_vals[i]
            curr_column.append(numerator / denominator)
        dd_table.append(curr_column)

    # Коэффициенты Ньютона — это верхние элементы каждого столбца
    return dd_table, [dd_table[i][0] for i in range(n)]

dd_table, newton_coeffs = compute_newton_coefficients(x_vals, f_vals)
def newton_interpolation(x_vals, y_vals, x, coef):
    """
    Вычисляет значение интерполяционного многочлена Ньютона в точке x
    с использованием рекурсивной формулы:
    P_{n+1}(x) = P_n(x) + alpha_{n+1} * omega_{n+1}(x)
    """
    result = coef[0]
    omega = 1.0
    for i in range(1, len(coef)):
        omega *= (x - x_vals[i - 1])
        result += coef[i] * omega
    return result

omegas = [np.prod([abs(x_point - x_val) for x_val in x_vals]) for x_point in x_star]

P_x_star = newton_interpolation(x_vals, f_vals, x_star1, newton_coeffs)
P_x_star2 = newton_interpolation(x_vals, f_vals, x_star2, newton_coeffs)
P_x_star3 = newton_interpolation(x_vals, f_vals, x_star3, newton_coeffs)

omega_frame = pd.DataFrame(omegas, index=[f'omega_{i}' for i in range(len(omegas))])
# Результаты интерполяции
data = {
    "Точка": ["x*", "x**", "x***"],
    "Значение x": [x_star1, x_star2, x_star3],
    "f(x)": [f_x_star, f_x_star2, f_x_star3],
    "P(x) (полином)": [P_x_star, P_x_star2, P_x_star3]
}

n = len(newton_coeffs)
dd_frame = pd.DataFrame(dd_table, columns=[f"f[x0..x{i}]" for i in range(len(newton_coeffs))])
```

```python
dd_frame.insert(0, "x_i", x_vals)
coeff_frame = pd.DataFrame(newton_coeffs, index=[f"a_{i}" for i in range(n)]).T
df = pd.DataFrame(data)

# Производная (n+1)-го порядка
x = sp.Symbol('x')
f_sym = alpha_j * sp.exp(x) + (1 - alpha_j) * sp.sin(x)
f_derivative = sp.diff(f_sym, x, n + 1)

# Максимум абсолютного значения производной на отрезке [0.7, 1.7]
f_derivative_abs = sp.lambdify(x, sp.Abs(f_derivative), 'numpy')
x_test = np.linspace(0.7, 1.7, 1000)
M_max = np.max(f_derivative_abs(x_test))

# Истинная погрешность
r_x_star = f_x_star - P_x_star
r_x_star2 = f_x_star2 - P_x_star2
r_x_star3 = f_x_star3 - P_x_star3

# Оценка погрешности по неравенству
factorial = math.factorial(n + 1)
x_stars = [x_star1, x_star2, x_star3]
r_x_stars = [r_x_star, r_x_star2, r_x_star3]
error_bound_stars = []

for i in range(len(x_star)):
    # prod_term = np.prod([abs(x_val - xi) for xi in x_vals])
    error_bound = M_max / factorial * abs(omegas[i])
    error_bound_stars.append(error_bound)

# Проверка выполнения неравенства
is_error_bound_stars_valid = [
    abs(r_x_stars[i]) <= error_bound_stars[i] for i in range(3)
]
# Таблица ошибок
error_table = pd.DataFrame({
    "Точка": ["x*", "x**", "x***"],
    "Значение x": [x_star1, x_star2, x_star3],
    "r истинная": [abs(r) for r in r_x_stars],
    "оценка погрешности": error_bound_stars,
    "M = max|f^(n+1)(x)|": [M_max] * 3,
    "Неравенство выполняется?": is_error_bound_stars_valid
})
# Вывод таблиц
display(table_transposed)
display(df)
display(error_table)
display(dd_frame)
display(coeff_frame)
display(omega_frame)
```

Результаты

Таблица значений:

	0	1	2	3	4	5	6	7	8	9	10
x_i	0.700000	0.800000	0.90000	1.000000	1.100000	1.200000	1.300000	1.400000	1.500000	1.600000	1.700000
f(x_i)	1.602892	1.773085	1.95672	2.155239	2.370278	2.603694	2.857575	3.134275	3.436431	3.766995	4.129263

Точка	Значение x	f(x)	P(x) (полином)
x*	0.766667	1.714927	1.714927
x**	1.250000	2.727935	2.727935
x***	1.666667	4.004765	4.004765

	Точка	Значение x	r истинная	оценка погрешности	M = max|f^(n+1)(x)|
0	x*	0.766667	1.021405e-13	1.658546e-13	4.129263
1	x**	1.250000	3.108624e-15	4.134870e-15	4.129263
2	x***	1.666667	2.433609e-13	3.535009e-13	4.129263

Коэффициенты интерполяционного многочлена (разделенные разности):

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_10
0	1.602892	1.701933	0.672075	0.240342	0.081633	0.016566	0.0023	0.000369	0.000059	0.000006	5.661216e-07

Таблица разделённых разностей

	x_i	f[x0..x0]	f[x0..x1]	f[x0..x2]	f[x0..x3]	f[x0..x4]	f[x0..x5]	f[x0..x6]	f[x0..x7]	f[x0..x8]	f[x0..x9]	f[x0..x10]
0	0.7	1.602892e+00	1.773085	1.956720	2.155239	2.370278	2.603694	2.857575	3.134275	3.436431	3.766995	4.129263
1	0.8	1.701933e+00	1.836348	1.985183	2.150398	2.334151	2.538816	2.766998	3.021559	3.305639	3.622678	NaN
2	0.9	6.720749e-01	0.744178	0.826076	0.918765	1.023320	1.140911	1.272809	1.420399	1.585195	NaN	NaN
3	1.0	2.403423e-01	0.272996	0.308962	0.348518	0.391970	0.439659	0.491968	0.549321	NaN	NaN	NaN
4	1.1	8.163336e-02	0.089916	0.098889	0.108629	0.119224	0.130772	0.143382	NaN	NaN	NaN	NaN
5	1.2	1.656594e-02	0.017946	0.019480	0.021189	0.023095	0.025221	NaN	NaN	NaN	NaN	NaN
6	1.3	2.299651e-03	0.002558	0.002849	0.003176	0.003543	NaN	NaN	NaN	NaN	NaN	NaN
7	1.4	3.685241e-04	0.000416	0.000467	0.000524	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	1.5	5.893869e-05	0.000065	0.000071	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	1.6	6.451632e-06	0.000007	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	1.7	5.661216e-07	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Анализ

Порядок $r_{ист}$ не превышает погрешности интерполяции в контрольных точках. Видно, что погрешность возрастает в точках, близких к краям отрезка

(x^*, x^{***}) и имеет наиболее точное значение в точке находящейся близко к середине отрезка (x^{**}). Разница погрешностей в зависимости от точки восстановления связана с возрастанием многочлена ω(x) и расположением контрольной точки относительно ближайшего узла.

$\omega_{n+1}(x^*)$ = -0.000002;

$\omega_{n+1}(x^{**})$ = 0.000064;

$\omega_{n+1}(x^{***})$ = 0.000123.

2. Интерполирование многочленом Ньютона на Чебышёвской сетке

Пусть теперь x_i заданы на отрезке [0.7,1.7] следующим образом :

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} cos\left(\frac{\pi(2i+1)}{2n+1}\right) \text{ , } где$$

$i = \overline{0, n}$,

a = 0,7,

b = 1,7,

n = 10, т. е.

$$x_i = 1 + \frac{1}{2} cos\left(\frac{\pi(2i+1)}{21}\right) \text{ , } i = \overline{0, \ 10},$$

Точки восстановления те же:

- $x^* = 0.766667$,

- $x^{**} = 1.25$,
- $x^{***} = 1.66667$

Остаток интерполирования в точках x^*, x^{**}, x^{***} оценим по следующим формулам:

$$|r_n(x)| \le \frac{M}{(n+1)!}|\omega_{n+1}(x)| \le \frac{M}{(n+1)!} \cdot 2\left(\frac{b-a}{4}\right)^{n+1}, \quad (\text{v}.1),$$

$$|r_n(x)| \le |f[x_0, \dots, x_n, x]| \cdot |\omega_{n+1}(x)| \le$$

$$\le |f[x_0, \dots, x_n, x]| \cdot 2\left(\frac{b-a}{4}\right)^{n+1}, (\text{v}.2), \text{где}$$

$x \in \{x^*, x^{**}, x^{***}\}$

$M = \|f^{(n+1)}\|_{C[a,b]} = \max_{a \le x \le b}|f^{(n+1)}(x)|$,

n = 10,

a = 0.7, b = 1.7.

Листинг кода

```
import numpy as np
import math
from tabulate import tabulate
import matplotlib.pyplot as plt
import pandas as pd
import sympy as sp
j = 12
n = 10
a = 0.7
b = 1.7
alpha_j = 0.1 + 0.05*j
h = 1/n
def f(x):
    return alpha_j * math.exp(x) + (1 - alpha_j) * math.sin(x)
x_vals = np.array(sorted([(a+b) / 2 + (b-a) / 2 * math.cos(math.pi * (2*i+1) / (2*n+2)) for i
in range(n+1)]))
f_vals = np.array([f(x_) for x_ in x_vals])
x_star = np.array([x_vals[0] + 2*h/3, x_vals[n // 2] + h / 2, x_vals[-1] - h / 3])
```

```
f_star = np.array([f(x_) for x_ in x_star])
delta_x_star = [min([abs(x - x_s) for x in x_vals]) for x_s in x_star]
print(tabulate(zip(x_vals, f_vals), headers=['x', 'f(x)']))
print('\nСпециальные точки')
print(tabulate(zip(x_star, f_star), headers=['x*', 'f(x*)']))
print('\nМинимальное расстояние от контрольных точек до узлов интерполяции:\n')
print(tabulate(zip(x_star, delta_x_star), headers=['x*_i', '\delta_i']))
# Таблица значений функции
table = pd.DataFrame({"x_i": x_vals, "f(x_i)": f_vals})
table_transposed = table.T
# Точки для проверки интерполяции
x_star1 = x_star[0]
x_star2 = x_star[1]
x_star3 = x_star[2]
f_x_star = f_star[0]
f_x_star2 = f_star[1]
f_x_star3 = f_star[2]
def compute_newton_coefficients(x_vals, y_vals):
    n = len(x_vals)
    # Создаём таблицу размером n x n
    dd_table = [y_vals.copy()]  # f[x_i]
    for level in range(1, n):
        prev_column = dd_table[-1]
        curr_column = []
        for i in range(n - level):
            numerator = prev_column[i + 1] - prev_column[i]
            denominator = x_vals[i + level] - x_vals[i]
            curr_column.append(numerator / denominator)
        dd_table.append(curr_column)
    # Коэффициенты Ньютона — это верхние элементы каждого столбца
    return dd_table, [dd_table[i][0] for i in range(n)]
dd_table, newton_coeffs = compute_newton_coefficients(x_vals, f_vals)
def newton_interpolation(x_vals, y_vals, x, coef):
    """
    Вычисляет значение интерполяционного многочлена Ньютона в точке x
    с использованием рекурсивной формулы:
    P_{n+1}(x) = P_n(x) + alpha_{n+1} * omega_{n+1}(x)
    """
    result = coef[0]
    omega = 1.0
    for i in range(1, len(coef)):
        omega *= (x - x_vals[i - 1])
        result += coef[i] * omega
    return result, omega
omegas = np.array([1.0,1.0,1.0])
P_x_star, omegas[0] = newton_interpolation(x_vals, f_vals, x_star1, newton_coeffs)
P_x_star2, omegas[1] = newton_interpolation(x_vals, f_vals, x_star2, newton_coeffs)
P_x_star3, omegas[2] = newton_interpolation(x_vals, f_vals, x_star3, newton_coeffs)
omega_frame = pd.DataFrame(omegas, index=[f'omega_{i}' for i in range(len(omegas))])
# Результаты интерполяции
```

```python
data = {
    "Точка": ["x*", "x**", "x***"],
    "Значение x": [x_star1, x_star2, x_star3],
    "f(x)": [f_x_star, f_x_star2, f_x_star3],
    "P(x) (полином)": [P_x_star, P_x_star2, P_x_star3]
}

#n = len(newton_coeffs)
dd_frame = pd.DataFrame(dd_table, columns=[f"f[x0..x{i}]" for i in range(len(newton_coeffs))])
dd_frame.insert(0, "x_i", x_vals)
coeff_frame = pd.DataFrame(newton_coeffs, index=[f"a_{i}" for i in
range(len(newton_coeffs))]).T
df = pd.DataFrame(data)
# Производная (n+1)-го порядка
x = sp.Symbol('x')
f_sym = alpha_j * sp.exp(x) + (1 - alpha_j) * sp.sin(x)
f_derivative = sp.diff(f_sym, x, n + 1)
# Максимум абсолютного значения производной на отрезке [0.7, 1.7]
f_derivative_abs = sp.lambdify(x, sp.Abs(f_derivative), 'numpy')
x_test = np.linspace(a, b, 1000)
M_max = np.max(f_derivative_abs(x_test))
# Истинная погрешность
r_x_star = f_x_star - P_x_star
r_x_star2 = f_x_star2 - P_x_star2
r_x_star3 = f_x_star3 - P_x_star3
# Оценка погрешности по неравенству
factorial = math.factorial(n + 1)
x_stars = [x_star1, x_star2, x_star3]
r_x_stars = [r_x_star, r_x_star2, r_x_star3]
error_bound_stars_v1 = [M_max / factorial * 2 * ((b-a)/4)**(n+1) for i in range(3)]
error_bound_stars_v2 = [dd_table[0][n-1] * 2 * ((b-a)/4)**(n+1) for i in range(3)]
# Проверка выполнения неравенства
is_error_bound_stars_valid_v1 = [
    abs(r_x_stars[i]) <= error_bound_stars_v1[i] for i in range(3)
]
is_error_bound_stars_valid_v2 = [
    abs(r_x_stars[i]) <= error_bound_stars_v2[i] for i in range(3)
]
# Таблица ошибок
error_table = pd.DataFrame({
    "Точка": ["x*", "x**", "x***"],
    "Значение x": [x_star1, x_star2, x_star3],
    "r истинная": [abs(r) for r in r_x_stars],
    "оценка погрешности v1": error_bound_stars_v1,
    "оценка погрешности v2": error_bound_stars_v2,
    "M = max|f^(n+1)(x)|": [M_max] * 3,
    "Неравенство оценки v1 выполняется?": is_error_bound_stars_valid_v1,
    "Неравенство оценки v2 выполняется?": is_error_bound_stars_valid_v2

})
```

```python
# Вывод таблиц
display(table_transposed)
display(df)
display(error_table)
display(dd_frame)
display(coeff_frame)
display(omega_frame)
```

Результаты

Таблица значений:

	0	1	2	3	4	5	6	7	8	9	10
x_i	0.705089	0.745184	0.822125	0.929680	1.059134	1.200000	1.340866	1.470320	1.577875	1.654816	1.694911
f(x_i)	1.611250	1.678212	1.812509	2.014017	2.280290	2.603694	2.967752	3.343927	3.691247	3.961424	4.110004

Коэффициенты интерполяционного многочлена (разделенные разности):

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_10
1.61125	1.670114	0.643691	0.225181	0.077884	0.016057	0.002234	0.000362	0.000059	0.000006	5.671354e-07

Интерполяция в контрольных точках:

Точка	Значение x	f(x)	P(x) (полином)
x*	0.771756	1.723712	1.723712
x**	1.250000	2.727935	2.727935
x***	1.661577	3.986094	3.986094

Точка	Значение x	r истинная	оценка погрешности v1	оценка погрешности v2	M = max|f^(n+1)(x)|
x*	0.771756	2.442491e-14	4.623513e-14	0.000002	3.870417
x**	1.250000	2.442491e-14	4.623513e-14	0.000002	3.870417
x***	1.661577	1.065814e-14	4.623513e-14	0.000002	3.870417

Анализ

Нетрудно видеть, что для x^*, x^{**}, x^{***} $r_{\text{ист}}$ не превосходит оценки сверху. Также можно заметить, что истинная погрешность в контрольных точках на чебышевской сетке улучшилась по сравнению с равномерной сеткой.

Многочлен $\omega_{n+1}(x)$ в контрольных точках принимает значения:

omega_0	4.907741e-07
omega_1	9.560534e-07
omega_2	5.198465e-06

Давайте проанализируем расположение точек восстановления относительно ближайших узлов:

```
Минимальное расстояние от контрольных точек до узлов интерполяции:

    x*_i     \delta_i
--------   ----------
0.771756   0.0265719
1.25       0.05
1.66158    0.00676139
```

Наименьшая погрешность наблюдается в точке x^{***}, для которой контрольная точка находится ближе остальных к ближайшему узлу интерполирования, что снижает интерполяционную ошибку.

Наибольшая погрешность соответствует точкам x^{*}, x^{**} , Заметим, что погрешность интерполяции в этих точках – самая высокая. Можно сделать следующий вывод: чем дальше контрольная точка от узла, тем выше ошибка интерполяции.

3. Интерполирование с помощью кубического сплайна

Будем строить кубический сплайн для равномерной сетки.

Каждый «кусок» сплайна будем искать в виде:

$$s_i(x) = \alpha_i + \beta_i(x - x_i) + \frac{\gamma_i}{2}(x - x_i)^2 + \frac{\delta_i}{6}(x - x_i)^3, \quad x \in \Delta_i = [x_{i-1}, x_i].$$

Коэффициенты вычисляем следующим образом:

$$\alpha_i = y_i, \quad i = \overline{1, n},$$

$$\beta_i = \frac{y_i - y_{i-1}}{h_i} + \frac{2\gamma_i + \gamma_{i-1}}{6} h_i.$$

$$\delta_i = \frac{\gamma_i - \gamma_{i-1}}{h_i}, \qquad i = \overline{2, n}.$$

Для нахождения γ_i необходимо решить СЛАУ (i = $\overline{1, n-1}$):

$$h_i \gamma_{i-1} + 2(h_i + h_{i+1})\gamma_i + h_{i+1}\gamma_{i+1} = 6\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i}\right)$$

методом прогонки.

В качестве граничных условий берём:

$$\gamma_0 = f''(a), \gamma_n = f''(b)$$

В нашей задаче мы имеем фиксированное h = 0.1.

Погрешность интерполирования естественным кубическим сплайном на всем отрезке интерполирования может быть оценена следующей константой:

$$|r(x)| \leqslant h^4 \max_{x \in [a,b]} |f^{(4)}(x)|,$$

Листинг кода

```
import numpy as np
import math
from tabulate import tabulate
import matplotlib.pyplot as plt
import pandas as pd
import sympy as sp

j = 12
n = 10
alpha_j = 0.1 + 0.05*j
a = alpha_j
b = 1+alpha_j
h = 1/n
```

```
def f(x):
    return alpha_j * math.exp(x) + (1 - alpha_j) * math.sin(x)

x_vals = np.array([alpha_j + i * h for i in range(n+1)])
f_vals = np.array([f(x_) for x_ in x_vals])
x_star = np.array([x_vals[0] + 2/3*h, x_vals[n // 2] + 0.5 * h, x_vals[-1] - h/3])
f_star = np.array([f(x_) for x_ in x_star])
print(tabulate(zip(x_vals, f_vals), headers=['x', 'f(x)']))
print('\nСпециальные точки')
print(tabulate(zip(x_star, f_star), headers=['x*', 'f(x*)']))
x_sym = sp.Symbol('x')
f_sym = alpha_j * sp.exp(x_sym) + (1 - alpha_j) * sp.sin(x_sym)
f_diff_2 = sp.lambdify(x_sym, sp.diff(f_sym, x_sym, 2), 'numpy')
f_diff_4_abs = sp.lambdify(x_sym, sp.Abs(sp.diff(f_sym, x_sym, 4)),'numpy')

def solve_spline_system(x, y):
    a_coef = [0] * n
    b_coef = [0] * n
    for i in range(1, n):
        temp1 = 6 * ((f_vals[i + 1] - f_vals[i]) - (f_vals[i] - f_vals[i - 1])) / h
        temp2 = 4 * h
        a_coef[i] = -h / temp2
        b_coef[i] = (temp1 - h * b_coef[i - 1]) / temp2
    for i in range(n - 1, 0, -1):
        gamma[i] = a_coef[i] * gamma[i + 1] + b_coef[i]
    gamma[0] = f_diff_2(x_vals[0])
    gamma[n] = f_diff_2(x_vals[-1])
    return gamma

gamma = solve_spline_system(x_vals, f_vals)
print(tabulate(enumerate(gamma), headers=['gamma_i']))
def build_cubic_spline(x, y, gamma):
    # Коэффициенты для каждого интервала [x_i, x_{i+1}]
    a = y
    b = np.zeros(n+1)
    c = [g / 2 for g in gamma]
    d = np.zeros(n+1)
    for i in range(1, n+1):
        b[i] = (y[i] - y[i-1]) / h + (2 * gamma[i] + gamma[i-1]) * h / 6
    for i in range(1, n+1):
        d[i] = (gamma[i] - gamma[i-1]) / (6 * h)

    def spline_func(xq):
        # Найти соответствующий интервал
        i = np.searchsorted(x, xq)
        dx = xq - x[i]
        return a[i] + b[i]*dx + c[i]*dx**2 + d[i]*dx**3
    return spline_func

spline = build_cubic_spline(x_vals, f_vals, gamma)
print(spline)
```

```python
x_segment = np.linspace(a,b, 1000)
error_bound = h**4 * np.max(f_diff_4_abs(x_segment))
f_interpolated = [spline(x_) for x_ in x_star]
true_error = abs(f_star - f_interpolated)
data = pd.DataFrame({
    'Точка' : x_star,
    'Значение функции' : f_star,
    'Интерполированное значение' : f_interpolated,
    'Истинная погрешность' : true_error,
    'Оценка погрешности' : error_bound
})
display(data)
```

Результаты

Точка	Значение функции	Интерполированное значение	Истинная погрешность	Оценка погрешности
0.766667	1.714927	1.714714	0.000213	0.000413
1.250000	2.727935	2.728042	0.000106	0.000413
1.666667	4.004765	4.004843	0.000077	0.000413

Анализ

Различия в погрешностях объясняются поведением исходной функции и свойствами кубического сплайна. Истинная погрешность зависит от локальных особенностей функции. Вблизи быстрорастущей экспоненты (правая часть интервала) погрешность стремится к верхней оценке, тогда как в середине и начале интервала она значительно меньше.

4. Интерполирование с помощью метода наименьших квадратов

Будем строить многочлен МНК 5 степени, он имеет следующий вид:

$$\varphi(x) = \sum_{i=0}^{n} \alpha_i \varphi_i(x),$$

где α - является решением системы линейных уравнений:

$$\sum_{j=0}^{n} \gamma_{lj} \alpha_j = \beta_l, \quad l = \overline{0, n},$$

которая в матричном виде записывается просто как:

$$\Gamma\alpha = \beta,$$

$$\Gamma = (\gamma_{ij})_{i,j=0}^{n}, \qquad \gamma_{ij} = (\phi_i, \phi_j), \qquad \beta_i = (y, \phi_i).$$

Положим $\varphi_i = x^i, i = \overline{0,6}$.

Скалярное произведение будем вычислять по формуле:

$$(\mathrm{u}, \mathrm{v}) = \sum_{i=0}^{N} u(x) \cdot v(x)$$

Т.е. имеем:

$$\gamma_{ij} = \sum_{k=0}^{N} \varphi_i(x_k)\varphi_j(x_k), \quad \beta_i = \sum_{k=0}^{N} y_k \varphi_i(x_k).$$

N = 10.

Погрешность будем вычислять по следующей формуле:

$$\Delta = \left(\sum_{k=0}^{N} \left(y_k - \sum_{i=0}^{n} \alpha_i \varphi_i(x_k) \right)^2 \right)^{1/2}$$

Листинг кода

```
import numpy as np
```

```python
import sympy as sp
import pandas as pd
import matplotlib.pyplot as plt

# Параметры
j = 12
N = 10  # количество точек - 1
h = 1 / N
alpha_j = 0.1 + 0.05 * j
n = 6   # количество базисных функций: ϕ_0 до ϕ_5 => многочлен 5-й степени
# Функция f(x)
def f(x):
    return alpha_j * np.exp(x) + (1 - alpha_j) * np.sin(x)
# Узлы интерполяции
x_vals = np.array([alpha_j + 0.1 * i for i in range(N + 1)])
y_vals = f(x_vals)
# Проверочные точки
x_star = x_vals[0] + (2 / 3) * h
x_star2 = x_vals[n // 2] + (1 / 2) * h
x_star3 = x_vals[-1] - (1 / 3) * h
TEST_POINTS = [x_star, x_star2, x_star3]
def phi(i, x):
    return x ** i
def calculate_gram_matrix(n, x_vals):
    A = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            A[i, j] = sum(phi(i, xk) * phi(j, xk) for xk in x_vals)
    return A
def calculate_beta(n, x_vals, y_vals):
    b = np.zeros(n)
    for i in range(n):
        b[i] = sum(fx * phi(i, xk) for fx, xk in zip(y_vals, x_vals))
    return b
def solve_system(A, b):
    return np.linalg.solve(A, b)
def approximate(alpha, x):
    return sum(alpha[i] * phi(i, x) for i in range(len(alpha)))
def calculate_error(alpha, x_vals):
    return np.sqrt(sum((f(xk) - approximate(alpha, xk)) ** 2 for xk in x_vals))
# Основная логика
A = calculate_gram_matrix(n, x_vals)
b = calculate_beta(n, x_vals, y_vals)
alpha = solve_system(A, b)
df_alpha = pd.DataFrame([ [round(a, 6) for a in alpha] ],
                        columns=[f"α{i}" for i in range(len(alpha))])
app_x_star = approximate(alpha, x_star)
app_x_star2 = approximate(alpha, x_star2)
app_x_star3 = approximate(alpha, x_star3)
df = pd.DataFrame({
```

```
    "Точка": ["x*", "x**", "x***"],
    "Значение x": [x_star, x_star2, x_star3],
    "f(x)": [f(x_star), f(x_star2), f(x_star3)],
    "ϕ(x)": [app_x_star, app_x_star2, app_x_star3]
})
# # Истинная ошибка
r_x_star = round(abs(f(x_star) - app_x_star),10)
r_x_star2 = round(abs(f(x_star2) - app_x_star2), 10)
r_x_star3 = round(abs(f(x_star3) - app_x_star3), 10)
error_bound = calculate_error(alpha, x_vals)
is_error_bound_valid = [
    r_x_star <= error_bound,
    r_x_star2 <= error_bound,
    r_x_star3 <= error_bound
]
error_table = pd.DataFrame({
    "Точка": ["x*", "x**", "x***"],
    "Значение x": [x_star, x_star2, x_star3],
    "r(ист)": [r_x_star, r_x_star2, r_x_star3],
    "Оценка погрешности Δ": [error_bound] * 3,
    "Неравенство выполняется?": is_error_bound_valid
})
# Отображение таблиц
display(df)
display(df_alpha)
display(error_table)
```

Результаты

	Точка	Значение x	f(x)	φ(x)
0	x*	0.766667	1.714927	1.714929
1	x**	1.050000	2.260583	2.260582
2	x***	1.666667	4.004765	4.004767

	α0	α1	α2	α3	α4	α5
0	0.694511	1.02948	0.286032	0.137965	-0.013215	0.020463

Точка	Значение x	r(ист)	Оценка погрешности Δ
x*	0.766667	1.910800e-06	0.000003
x**	1.050000	4.747000e-07	0.000003
x***	1.666667	1.665900e-06	0.000003

Анализ

В ходе интерполяции с применением метода наименьших квадратов была проведена оценка точности аппроксимации на ряде контрольных точек. Вычисленные значения ошибок оказались порядка 10^{-7}, что свидетельствует о высокой степени точности, достигаемой данным методом. Полученные результаты оказались лучше теоретических оценок, что указывает на эффективность выбранного подхода и точное воспроизведение исходной функции.

Замечено, что значения погрешности для всех тестовых точек имеют сопоставимый масштаб, что говорит о равномерном распределении ошибки по всему интервалу. Это подчеркивает не только точность, но и стабильность метода наименьших квадратов при аппроксимации.

Дополнительно устойчивость метода подтверждается отсутствием значительных расхождений между реальными и аппроксимированными значениями: ошибки во всех точках остаются устойчиво малы, что свидетельствует о надежности численных вычислений.

Выводы

Полином пятой степени, использованный для аппроксимации, продемонстрировал высокую эффективность при приближении исходной функции на отрезке [0,7;1,7]. Он не только обеспечивает точное, но и равномерное приближение по всему интервалу, что указывает на достаточность выбранной степени. Повышение степени полинома, скорее всего, не приведет к значимому приросту точности.

Таким образом, в рамках проведённого эксперимента метод наименьших квадратов проявил себя как надёжный и точный инструмент для аппроксимации. Использование полинома пятой степени обеспечило устойчивое и высокоточное приближение, превзошедшее теоретически ожидаемую точность, что подтверждает практическую ценность данного подхода в задачах численного анализа.