МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра вычислительной математики

Отчёт

Лабораторная работа 2

“Интерполирование многочленом Лагранжа на Чебышёвской сетке”

Вариант 5

Благодарного Артёма Андреевича
студента 3 курса, 3 группы
специальности «Информатика»
дисциплина «Численные методы»
Преподаватель: Будник А.М

Минск, 2025

Постановка Задачи

Для минимизации остатка интерполирования на отрезке [a, b] выбрать узлы интерполирования оптимальным образом. По значениям функции в этих узлах построить интерполяционный многочлен Лагранжа, вычислить приближенные значения в x*, x**, x***, которые были определены ранее на равномерной сетке узлов и оценить погрешность.

Алгоритм решения

Это можно сделать при помощи Чебышёвских узлов, которые определяются по формуле:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\cos\frac{\pi(2i+1)}{2n+2}, \quad i = \overline{0,n}.$$

где $i = \overline{0,n}$, $a = 0.5$, $b = 1.5$, $n = 10$, x_i являются корнями многочлена Чебышева, который, в свою очередь, является наименее отклоняющимся от нуля среди всех многочленов (то есть его значение по норме всегда будет меньше нормы значения любого другого

$$\|\omega_{n+1}\| = 2\left(\frac{b-a}{4}\right)^{n+1}.$$

полинома той же степени). Кроме этого, при выборе Чебышёвских узлов интерполирования имеем:

Это равенство используется при оценке погрешности интерполирования

$\|f^{n+1}\| = M$ (считали ранее в лабораторной работе №1)

$$\|r_n\| \leqslant 2\left(\frac{b-a}{4}\right)^{n+1} \frac{\|f^{(n+1)}\|}{(n+1)!}.$$

Листинг программы

```
import numpy as np
import math
import pandas as pd
import sympy as sp
from IPython.display import display

j = 5
alpha_j = 0.1 + 0.05 * j  # alpha_j = 0.35
n = 10  # степень интерполяционного многочлена
a, b = 0.35, 1.35  # интервал интерполирования

# Определение функции f(x)
def f(x):
    return alpha_j * np.exp(x) + (1 - alpha_j) * np.cos(x)

# Генерация узлов Чебышёва на отрезке [a, b]
def chebyshev_nodes(a, b, n):
    return np.array([
        0.5 * (a + b) + 0.5 * (b - a) * np.cos(np.pi * (2 * k + 1) / (2 * n + 1))
        for k in range(n + 1)
    ])

# Узлы и значения функции
x_vals = chebyshev_nodes(a, b, n)
f_vals = f(x_vals)

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

# Интерполяционный многочлен Лагранжа
def lagrange_interpolation(x_vals, y_vals, x):
    n = len(x_vals)
    result = 0.0
    for i in range(n):
        term = y_vals[i]
        for j in range(n):
            if i != j:
                term *= (x - x_vals[j]) / (x_vals[i] - x_vals[j])
        result += term
    return result

# Точки для оценки интерполяции
x_star = 0.41667
x_star2 = 0.9
x_star3 = 1.31667

# Значения функции в точках
f_x_star = f(x_star)
```

```
f_x_star2 = f(x_star2)
f_x_star3 = f(x_star3)

# Значения интерполяционного многочлена в этих точках
P_x_star = lagrange_interpolation(x_vals, f_vals, x_star)
P_x_star2 = lagrange_interpolation(x_vals, f_vals, x_star2)
P_x_star3 = lagrange_interpolation(x_vals, f_vals, x_star3)

# Таблица интерполяционных значений
interp_data = {
    "Точка": ["x*", "x**", "x***"],
    "Значение x": [x_star, 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]
}
df = pd.DataFrame(interp_data)

# Оценка максимальной погрешности
x = sp.Symbol('x')
f_sym = alpha_j * sp.exp(x) + (1 - alpha_j) * sp.sin(x)

# Производная (n+1)-го порядка
f_derivative = sp.diff(f_sym, x, n + 1)
f_derivative_abs = sp.lambdify(x, sp.Abs(f_derivative), 'numpy')

# Максимум на [a, b]
x_test = np.linspace(a, b, 1000)
M_max = np.max(f_derivative_abs(x_test))

# Истинные остатки
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
factorial = math.factorial(n + 1)
x_stars = [x_star, x_star2, x_star3]
r_x_stars = [r_x_star, r_x_star2, r_x_star3]
error_bound_stars = []
is_error_bound_stars_valid = []

for r in r_x_stars:
    # Формула оценки максимальной погрешности
    error_bound = (M_max / factorial) * (2 * ((1 / 4) ** (n + 1)))
    error_bound_stars.append(error_bound)
    is_error_bound_stars_valid.append(abs(r) <= error_bound)

# Таблица ошибок
error_table = pd.DataFrame({
    "Точка": ["x*", "x**", "x***"],
```

```
    "Значение x": x_stars,
    "r(истинная ошибка)": [abs(r_x_star), abs(r_x_star2), abs(r_x_star3)],
    "Оценка погрешности": error_bound_stars,
    "M = max|f^(n+1)(x)|": [M_max]*3,
    "Неравенство выполняется?": is_error_bound_stars_valid
})

# Вывод результатов

print("Таблица значений функции в узлах Чебышёва:")
display(table_transposed)

print("Значения функции и полинома в контрольных точках:")
display(df)

print("Сравнение истинных и оценённых ошибок:")
display(error_table)
```

Результаты

- Исходная таблица:

	0	1	2	3	4	5	6	7	8	9	10
x_i	1.344415	1.300484	1.216526	1.100000	0.961260	0.812635	0.667329	0.538255	0.436881	0.372214	0.350000
f(x_i)	1.975996	1.911273	1.791039	1.630743	1.448191	1.260810	1.084438	0.932770	0.816780	0.744221	0.719557

- Значение функции и полинома в точках x*, x**, x***

	Точка	Значение x	f(x)	P(x) (полином)
0	x*	0.416667	0.793979	0.793979
1	x**	0.900000	1.370024	1.370024
2	x***	1.316670	1.934966	1.934966

- Оценка погрешности (Чебышёвская сетка)

	Точка	Значение x	r(истинная ошибка)	Оценка погрешности	M = max|f^(n+1)(x)|	Неравенство выполняется?
0	x*	0.416667	1.776357e-15	1.442745e-14	1.207745	True
1	x**	0.900000	5.551115e-15	1.442745e-14	1.207745	True
2	x***	1.316670	7.993606e-15	1.442745e-14	1.207745	True

- Оценка погрешности (Равномерная сетка)

	Точка	Значение x	r истинная	оценка погрешности	M = max|f^(n+1)(x)|	Неравенство выполняется?
0	x*	0.416667	6.239453e-14	9.928288e-14	2.059862	True
1	x**	0.900000	1.998401e-15	2.475192e-15	2.059862	True
2	x***	1.316667	1.374456e-13	2.116106e-13	2.059862	True

Анализ результатов

- Как мы видим из таблицы оценки погрешности, для всех точек x*, x**, x*** истинная погрешность меньше или равна теоретической погрешности (оценки сверху)
- Также можно заметить, что истинная погрешность в точках x*, x*** на Чебышевской сетке улучшилась по сравнению с равномерной сеткой, в то время как в точке x** ухудшилась.

Покажем почему это происходит. Для этого напишем вспомогательную функцию вычисления значения полинома w_{n+1} в точках x*, x**, x***

```
# ω_{n+1}(x) — произведение (x - x_i) по всем узлам x_i
def w_nplus1(x, x_vals):
    return np.prod([(x - xi) for xi in x_vals])

# Вычисляем ω(x) для контрольных точек
w_x_star = w_nplus1(x_star, x_vals)
w_x_star2 = w_nplus1(x_star2, x_vals)
w_x_star3 = w_nplus1(x_star3, x_vals)

# Выводим результаты
print(f"ω(x*)   = {w_x_star:.5e}")
print(f"ω(x**)  = {w_x_star2:.5e}")
print(f"ω(x***) = {w_x_star3:.5e}")
```

Результат:

```
ω(x*)   = -1.76446e-07
ω(x**)  = -6.82322e-07
ω(x***) = -7.08689e-07
```

Если взять эти значения по модулю, то оказывается: чем меньше значение полинома w_{n+1}, тем меньше остаток интерполирования.

Вывод

Чем ближе точка восстановления находится к узлу интерполирования, тем меньше будет значение w_{n+1}, так как узлы являются корнями многочлена w_{n+1}, то есть чем ближе точки x*, x**, x*** к узлу интерполирования, тем меньше значение полинома.