**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет прикладной математики и информатики

Лабораторная работа №1

“Интерполяция функции многочленом Ньютона”

Снежко Лев,   
студент 3 курса 3 группы,  
дисциплина «Численные методы»  
Преподаватель: Будник А.М.

2025

## Постановка задачи

В этой лабораторной работе требуется восстановить значения функции *f(x)* в точках *x∊[0.7, 1.7]* с помощью интерполяционного многочлена Лагранжа.

**Алгоритм решения**

1. **Формулировка задачи интерполяции**

Пусть на отрезке *[0.7, 1.7]* заданы 11 узлов интерполяции согласно формуле: . Заметим, что .

Необходимо интерполировать многочленом Лагранжа *P10(x)* функцию:

Причём .

|  |  |
| --- | --- |
|  |  |
| 0.15 | 0.30129755 |
| 0.25 | 0.40289718 |
| 0.35 | 0.50432327 |
| 0.45 | 0.60496753 |
| 0.55 | 0.7042721 |
| 0.65 | 0.80173957 |
| 0.75 | 0.89694295 |
| 0.85 | 0.98953537 |
| 0.95 | 1.07925963 |
| 1.05 | 1.16595741 |
| 1.15 | 1.24957829 |

1. **Интерполяция многочленом Ньютона**

Будем интерполировать функцию с помощью многочлена Ньютона *P10(x)*. Отметим, что узлы интерполяции попарно различны, поэтому интерполяционный многочлен существует и единственен. Будем вычислять значения, принимаемые ИМ Ньютона, по рекуррентной формуле:

где , *xi*– узлы интерполяции, *yi = f(xi)* – значения функции.

1. **Анализ результатов**

Будем сравнивать значения функции и ИМ Ньютона в точках восстановления *x\* =* 0.766667*, x\*\* =* 0.65*, x\*\*\* =* 1.666667: оценим сверху остаток интерполяции и вычислим истинный остаток интерполяции . Воспользуемся следующими формулами:

,

– многочлен 11-й степени вида

**Листинг программы**

%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] + 2/3\*h, x\_vals[n // 2] / 2 + 0.5 \* h, x\_vals[-1] - 1/3\*h])

f\_star = np.array([f(x\_) for x\_ in x\_star])

print(tabulate(zip(x\_vals, y\_vals), headers=['x', 'f(x)']))

print('\nСпециальные точки')

print(tabulate(zip(x\_star, f\_star), headers=['x\*', 'f(x\*)']))

# Строим интерполяционный многочлен

# 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 newton\_interpolation(x\_vals, y\_vals, x):

    n = len(x\_vals)

    # Строим таблицу разделённых разностей

    coef = [0.0 for \_ in range(n)]

    for i in range(n):

        coef[i] = y\_vals[i]

    for j in range(1, n):

        for i in range(n - 1, j - 1, -1):

            coef[i] = (coef[i] - coef[i - 1]) / (x\_vals[i] - x\_vals[i - j])

    # Вычисляем значение многочлена в точке x

    result = coef[0]

    product\_term = 1.0

    for i in range(1, n):

        product\_term \*= (x - x\_vals[i - 1])

        result += coef[i] \* product\_term

    return result

P\_x\_star = newton\_interpolation(x\_vals, f\_vals, x\_star1)

P\_x\_star2 = newton\_interpolation(x\_vals, f\_vals, x\_star2)

P\_x\_star3 = newton\_interpolation(x\_vals, f\_vals, x\_star3)

# 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]

}

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 x\_val in x\_stars:

    prod\_term = np.prod([abs(x\_val - xi) for xi in x\_vals])

    error\_bound = M\_max / factorial \* prod\_term

    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

})

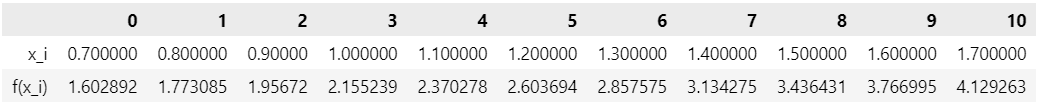
# Вывод таблиц

display(table\_transposed)

display(df)

display(error\_table)

### Результаты и их анализ



|  | **Точка** | **Значение x** | **f(x)** | **P(x) (полином)** |
| --- | --- | --- | --- | --- |
| 0 | x\* | 0.766667 | 1.714927 | 1.714927 |
| 1 | x\*\* | 0.650000 | 1.522435 | 1.522435 |
| 2 | x\*\*\* | 1.666667 | 4.004765 | 4.004765 |

|  | **Точка** | **Значение x** | **r истинная** | **оценка погрешности** | **M = max|f^(n+1)(x)|** | **Неравенство выполняется?** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | x\* | 0.766667 | 1.021405e-13 | 1.343614e-13 | 2.78765 | True |
| 1 | x\*\* | 0.650000 | 3.516742e-12 | 4.688495e-12 | 2.78765 | True |
| 2 | x\*\*\* | 1.666667 | 2.433609e-13 | 2.863766e-13 | 2.78765 | True |

Видно, что значение истинного остатка не превзошло оценки сверху, что свидетельствует о действенности данной оценки.