«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №5

Вариант 1

Студент:

Алхимовици А.

P3210

Преподаватель:

Наумова Н. А.

Санкт-Петербург, 2025 г.

Цель лабораторной работы:

Решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек.

1 Вычислительная реализация задачи:

1. Выбрать таблицу $y = f(x)$:

	X	y	вариант	X1	X2
Таблица 1.1	0.25	1.2557	1	0.251	0.402
	0.30	2.1764			
	0.35	3.1218			
	0.40	4.0482			
	0.45	5.9875			
	0.50	6.9195			
	0.55	7.8359			

2. Построить таблицу конечных разностей:

№	x	Δ0	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6
0	0.25	1.2557	2.1764	3.1218	4.0482	5.9875	6.9195	7.8359
1	0.30	0.9207	0.9454	0.9264	1.9393	0.9320	0.9164	
2	0.35	0.0247	-0.0190	1.0129	-1.0073	-0.0156		
3	0.40	-0.0437	1.0319	-2.0202	0.9917			
4	0.45	1.0756	-3.0521	3.0119				
5	0.50	-4.1277	6.0640					
6	0.55	10.1917						

3. Вычислить значения функции для аргумента $X1$ используя интерполяционную формулу Ньютона

Воспользуемся формулой Ньютона для интерполирования вперед(первая формула), так как X1 = 0.251 лежит в левой половине отрезка

Для $X_1 = 0.251$: $t = \frac{(x-x_0)}{h} = \frac{(0.251-0.25)}{0.05} = 0.02$

$$N_6(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \frac{t(t-1)(t-2)}{3!}\Delta^3 y_0 + \frac{t(t-1)(t-2)(t-3)}{4!}\Delta^4 y_0 + \frac{t(t-1)(t-2)(t-3)(t-4)}{5!}\Delta^5 y_0 + \frac{t(t-1)(t-2)(t-3)(t-4)(t-5)}{6!}\Delta^6 y_0$$

$$
\begin{aligned}
y(0.251) \approx{} & 1.2557 + 0.02 * 2.1764 + \frac{0.02 \cdot (-0.98)}{2} * 3.1218 \\
& + \frac{0.02(-0.98)(-1.98)}{6} * (4.0482) + \frac{0.02(-0.98)(-1.98)(-2.98)}{24} \\
& * (5.9875) + \frac{0.02(-0.98)(-1.98)(-2.98)(-3.98)}{120} * (6.9195) \\
& + \frac{0.02(-0.98)(-1.98)(-2.98)(-3.98)(-4.98)}{720} * (7.8359) \\
& \approx 1.2676
\end{aligned}
$$

$$y(0.251) \approx 1.2676$$

4. **Вычислить значения функции для аргумента $\boldsymbol{X_2}$**, используя интерполяционную формулу **Гаусса**:

Воспользуемся формулой Гауса для интерполирования вперед(вторая формула), так как X2 = 0.402 лежит в правой половине отрезка

$$\mathrm{t} \;=\; \frac{(x - x_{mid})}{h} = \frac{(0.402 - 0.4)}{0.05} = 0.04$$

$$P(x) = y_{mid} + \sum_{k=1}^{3} \frac{p_k}{k!} \boldsymbol{\Delta^k y_{idxk}}$$

$$\boldsymbol{T1 = \frac{p_1}{1!} \Delta^1 y_3 = 0.04 * 1.0129 = 0.041276}$$

$$\boldsymbol{T2 = \frac{p_2}{2!} \Delta^2 y_1 = -\frac{0.019}{2} * 0.9320 = -0.008523}$$

$$\boldsymbol{T3 = \frac{p_3}{3!} \Delta^3 y_2 = \frac{0.01766}{6} * (-1.0073) = -0.002966}$$

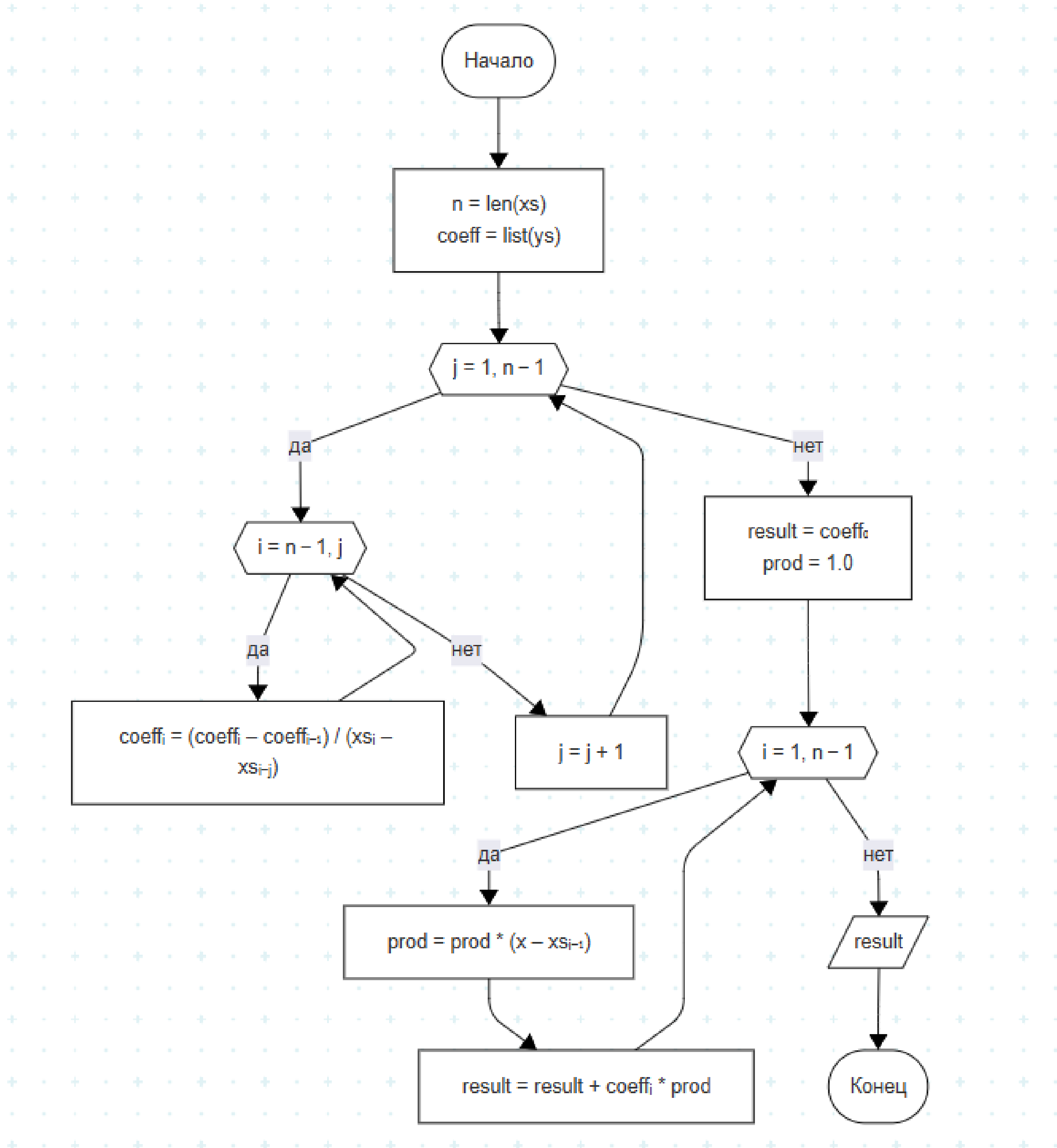
P(0.402)=ymid+T1+T2+T3=4.111195+0.041276−0.008523−0.002966≈4.140982

$$P(0.402) \approx 4.141$$

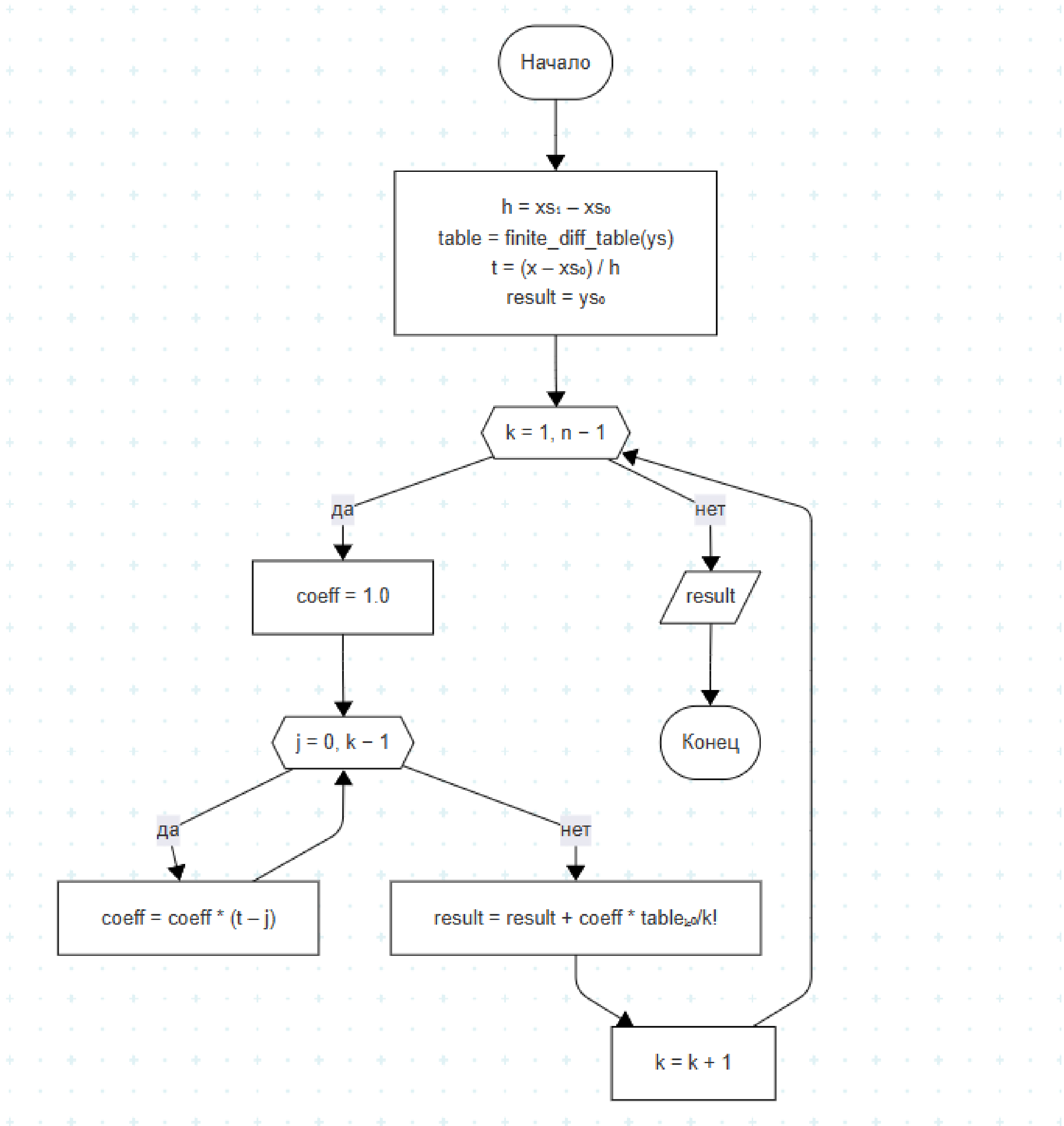
2. Программная реализация

Блок схемы

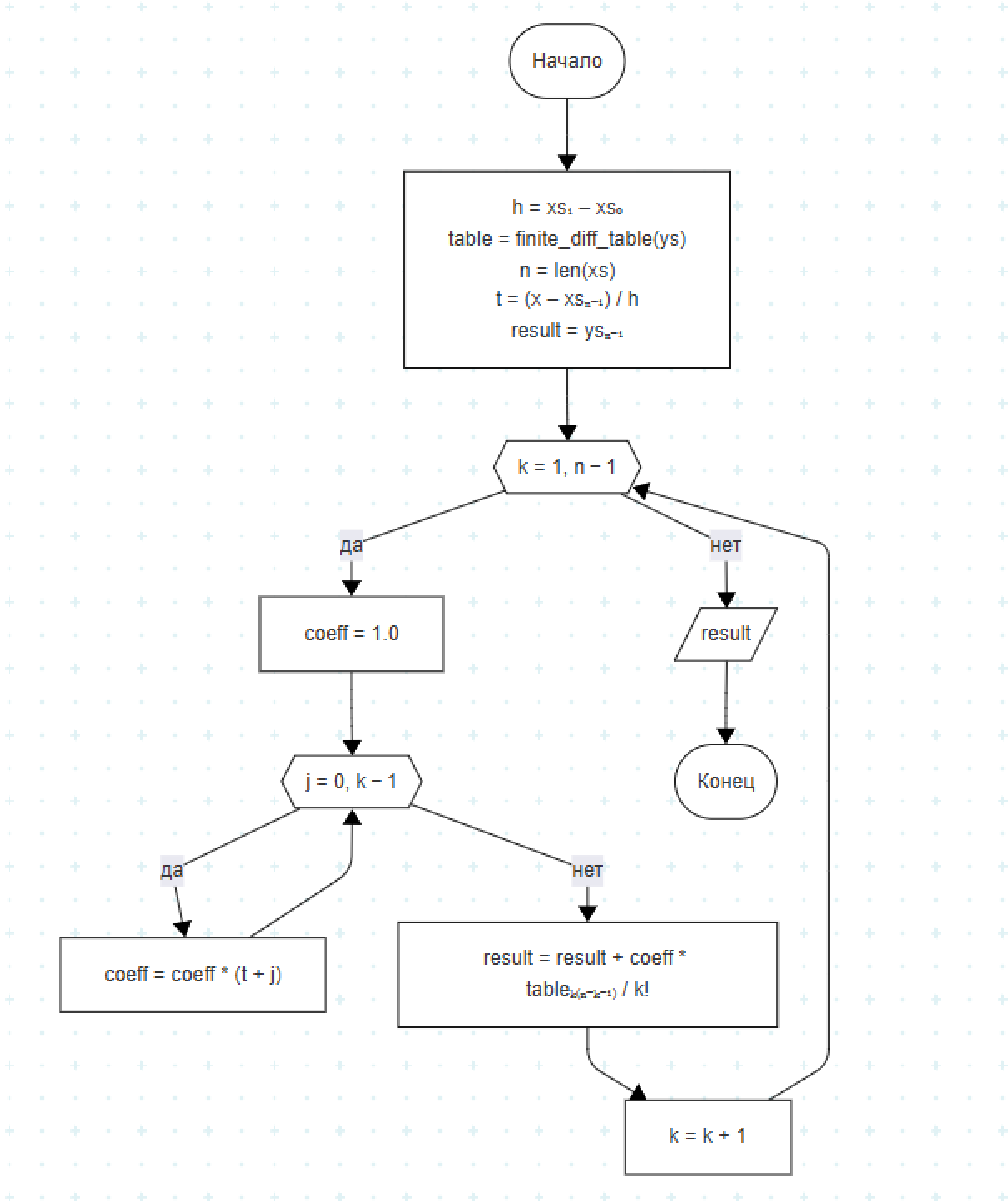
Многочлен Ньютона с разделенными разностями



Формула прямых конечных разностей



Формула обратных конечных разностей



Интерполяция Лагранжа

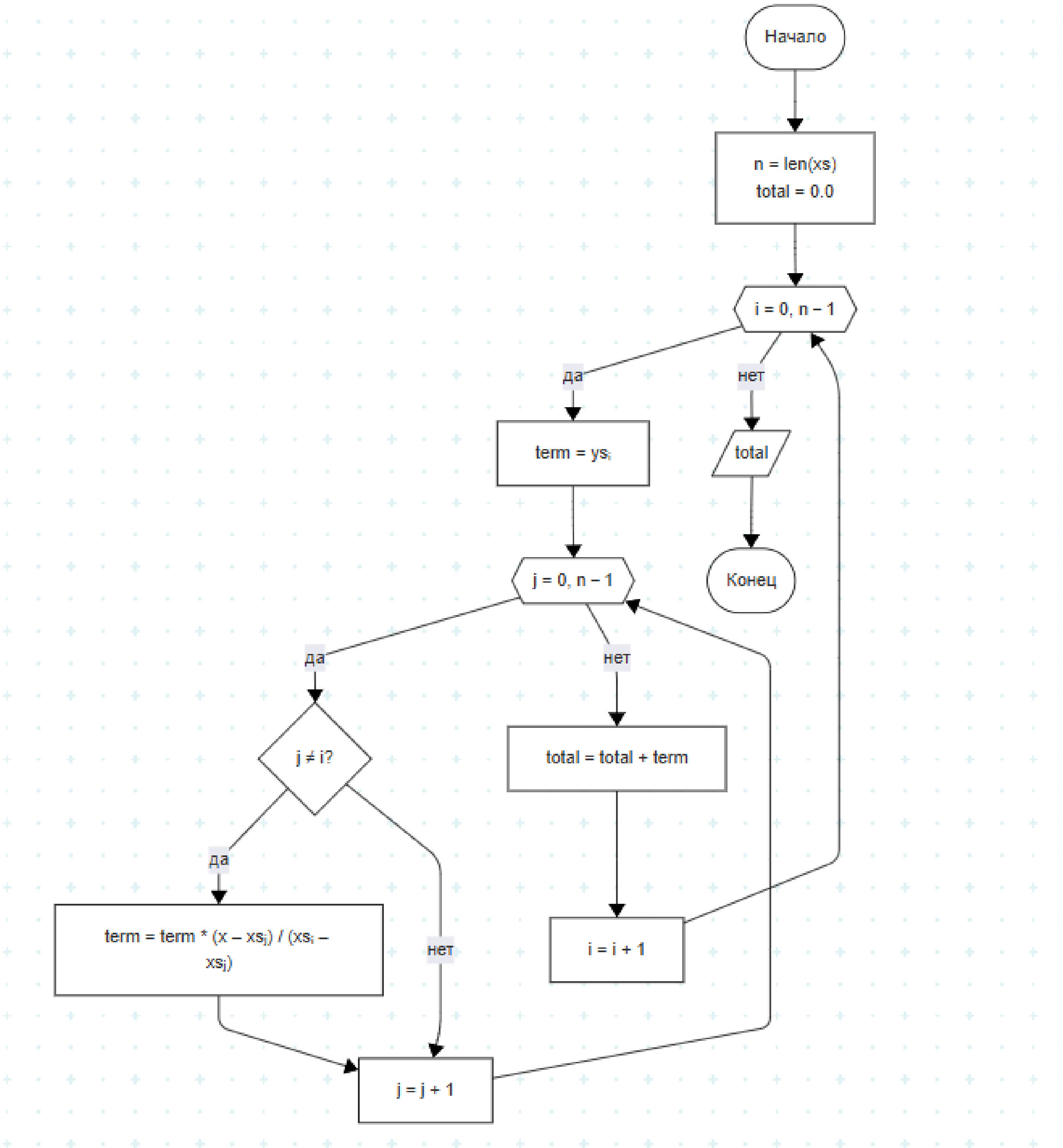


Схема Стирлинга

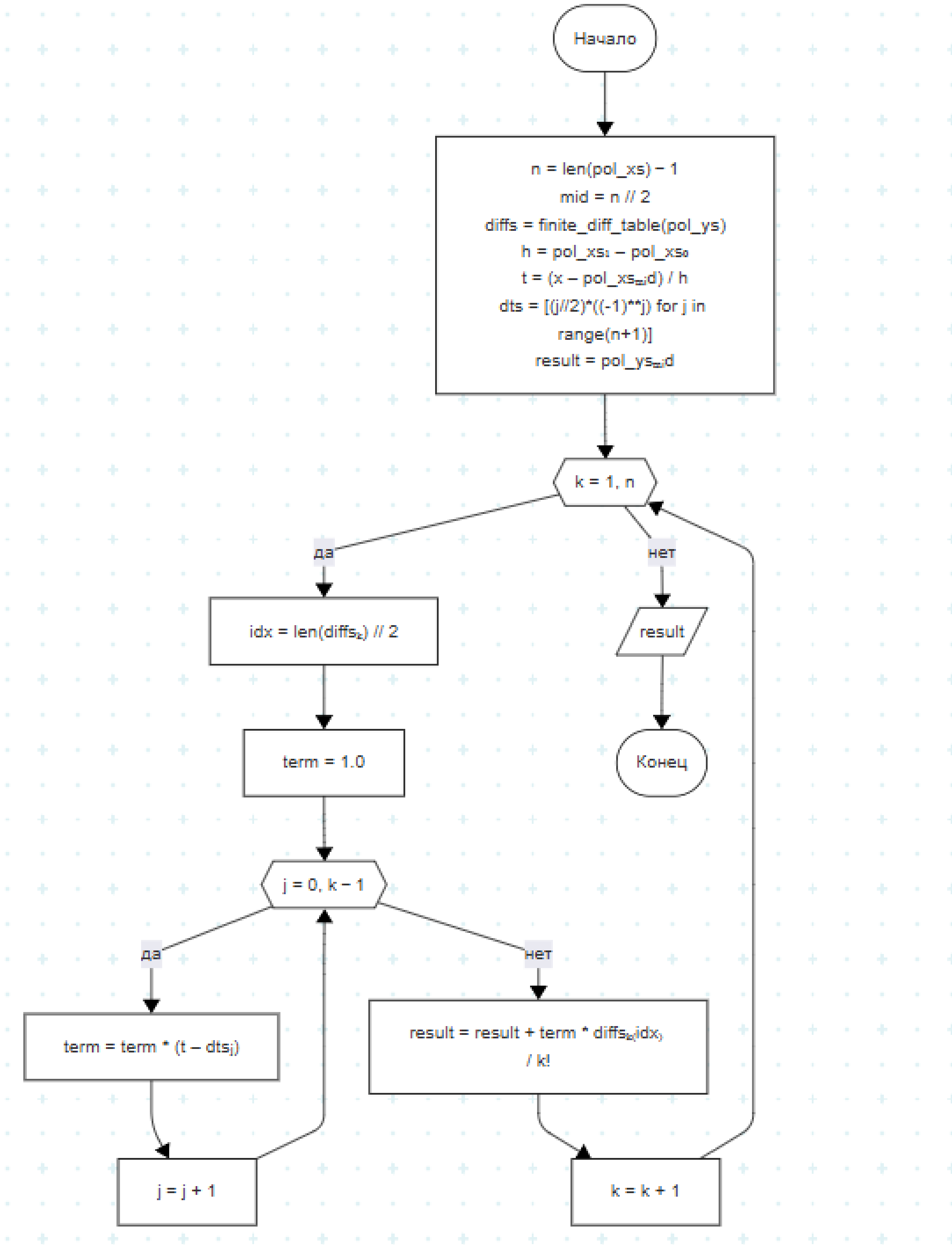
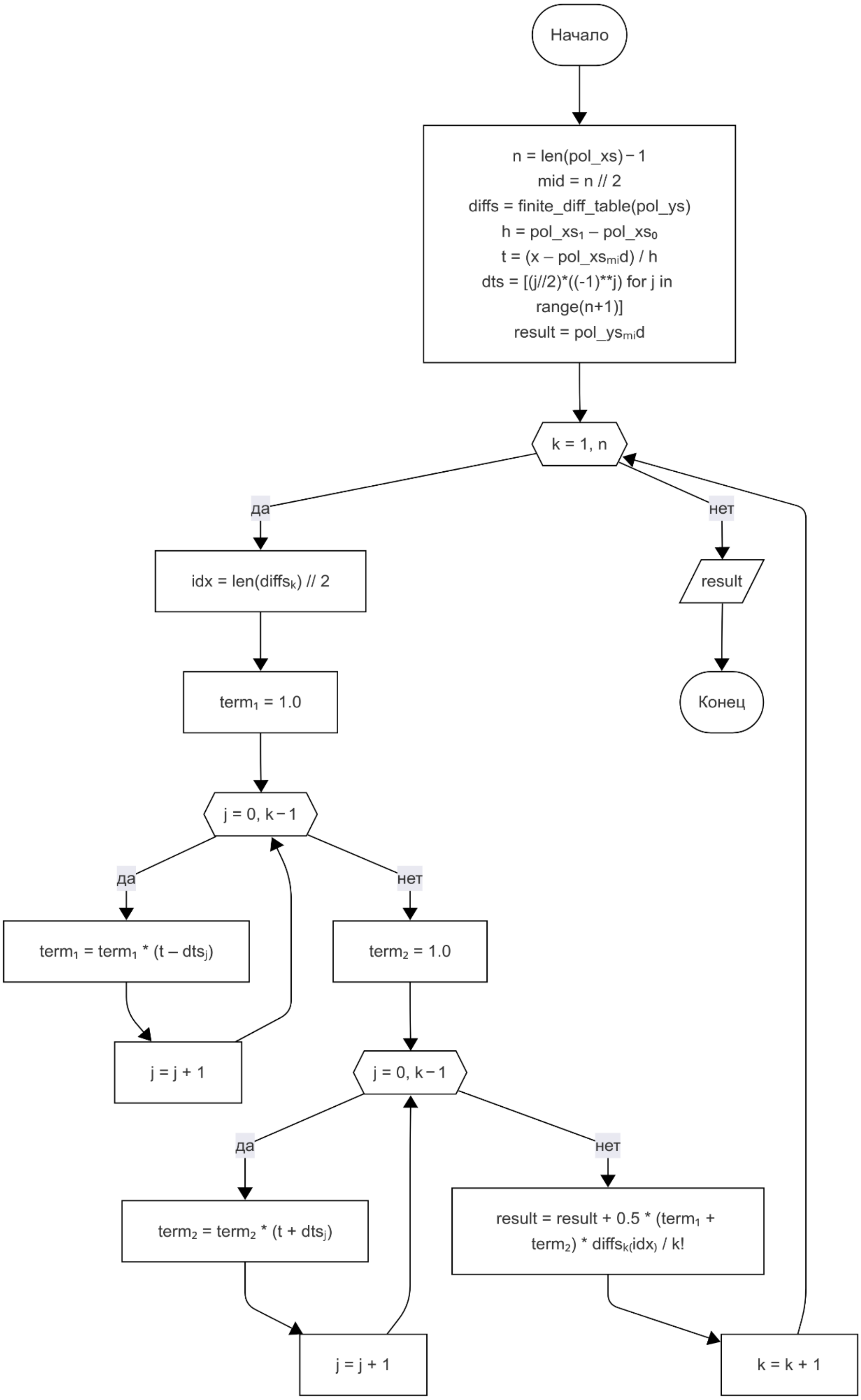


Схема Бесселя



Листинг программы

https://github.com/senya-2011/Vu4Math/tree/main/lab5

```
def stirling(pol_xs, pol_ys, x):
    """Схема Стирлинга для равноотстоящих узлов"""
    n = len(pol_xs) - 1
    mid = n // 2
    diffs = finite_diff_table(pol_ys)
    h = pol_xs[1] - pol_xs[0]
    t = (x - pol_xs[mid]) / h
    # dts: последовательность шагов для центральной схемы
    dts = [(j // 2) * ((-1) ** (j)) for j in range(n + 1)]
    result = pol_ys[mid]
    for k in range(1, n + 1):
        # выбор центрального элемента конечных разностей
        idx = len(diffs[k]) // 2
        term = 1.0
        for j in range(k):
            term *= (t - dts[j])
        result += term * diffs[k][idx] / factorial(k)
    return result


def bessel(pol_xs, pol_ys, x):
    """Схема Бесселя для равноотстоящих узлов"""
    n = len(pol_xs) - 1
    mid = n // 2
    diffs = finite_diff_table(pol_ys)
    h = pol_xs[1] - pol_xs[0]
    t = (x - pol_xs[mid]) / h
    # dts: последовательность шагов для центральной схемы
    dts = [(j // 2) * ((-1) ** (j)) for j in range(n + 1)]
    result = pol_ys[mid]
    for k in range(1, n + 1):
        idx = len(diffs[k]) // 2
        # первая часть
        term1 = 1.0
        for j in range(k):
            term1 *= (t - dts[j])
        # вторая часть (сдвиг на 0.5)
        term2 = 1.0
        for j in range(k):
            term2 *= (t + dts[j])
        result += 0.5 * (term1 + term2) * diffs[k][idx] / factorial(k)
    return result

def gauss_forward(xs, ys, x):
    h = xs[1] - xs[0]
    n = len(xs)
    mid = n // 2
    table = finite_diff_table(ys)
    t = (x - xs[mid]) / h
    result = ys[mid]
    p = 1.0

    for k in range(1, mid + 1):
        if k % 2 == 1:
            idx = mid - (k // 2)
        else:
            idx = mid - (k // 2) - 1

        if not (0 <= idx < len(table[k])):
            raise IndexError(f"gauss_forward: k={k}, idx={idx},
len(table[{k}])={len(table[k])}")
```

```
        p *= (t - (k - 1) / 2)
        result += p * table[k][idx] / factorial(k)

    return result


def gauss_backward(xs, ys, x):
    h = xs[1] - xs[0]
    n = len(xs)
    mid = n // 2
    table = finite_diff_table(ys)
    t = (x - xs[mid]) / h
    result = ys[mid]
    p = 1.0

    for k in range(1, mid + 1):
        if k % 2 == 1:
            idx = mid - (k // 2)
        else:
            idx = mid - (k // 2) - 1

        if not (0 <= idx < len(table[k])):
            raise IndexError(f"gauss_backward: k={k}, idx={idx},
len(table[{k}])={len(table[k])}")

        p *= (t + (k - 1) / 2)
        result += p * table[k][idx] / factorial(k)
    return result

def lagrange(xs, ys, x):
    """
    Интерполяция Лагранжа.
    """
    n = len(xs)
    total = 0.0
    for i in range(n):
        term = ys[i]
        for j in range(n):
            if j != i:
                term *= (x - xs[j]) / (xs[i] - xs[j])
        total += term
    return total

# Многочлен Ньютона с разделёнными разностями
def newton_divided(xs, ys, x):
    """
    Интерполяция Ньютона (разделённые разности).
    """
    n = len(xs)
    coeff = list(ys)
    for j in range(1, n):
        for i in range(n - 1, j - 1, -1):
            coeff[i] = (coeff[i] - coeff[i - 1]) / (xs[i] - xs[i - j])
    result = coeff[0]
    prod = 1.0
    for i in range(1, n):
        prod *= (x - xs[i - 1])
        result += coeff[i] * prod
    return result

# Многочлен Ньютона с конечными разностями (прямая формула)
def newton_forward(xs, ys, x):
    """Формула прямых конечных разностей. Только для равноотстоящих
узлов."""
    h = xs[1] - xs[0]
    table = finite_diff_table(ys)
```

```
    t = (x - xs[0]) / h
    result = ys[0]
    for k in range(1, len(xs)):
        coeff = 1.0
        for j in range(k):
            coeff *= (t - j)
        result += coeff * table[k][0] / factorial(k)
    return result

# Многочлен Ньютона с конечными разностями (обратная формула)
def newton_backward(xs, ys, x):
    """Формула обратных конечных разностей. Только для равноотстоящих
узлов."""
    h = xs[1] - xs[0]
    table = finite_diff_table(ys)
    n = len(xs)
    t = (x - xs[-1]) / h
    result = ys[-1]
    for k in range(1, n):
        coeff = 1.0
        for j in range(k):
            coeff *= (t + j)
        result += coeff * table[k][n - k - 1] / factorial(k)
    return result
```

Примеры и результаты работы программы

Интерполяция для x = 1.0

Скачать отчет Новая задача

Все методы сразу

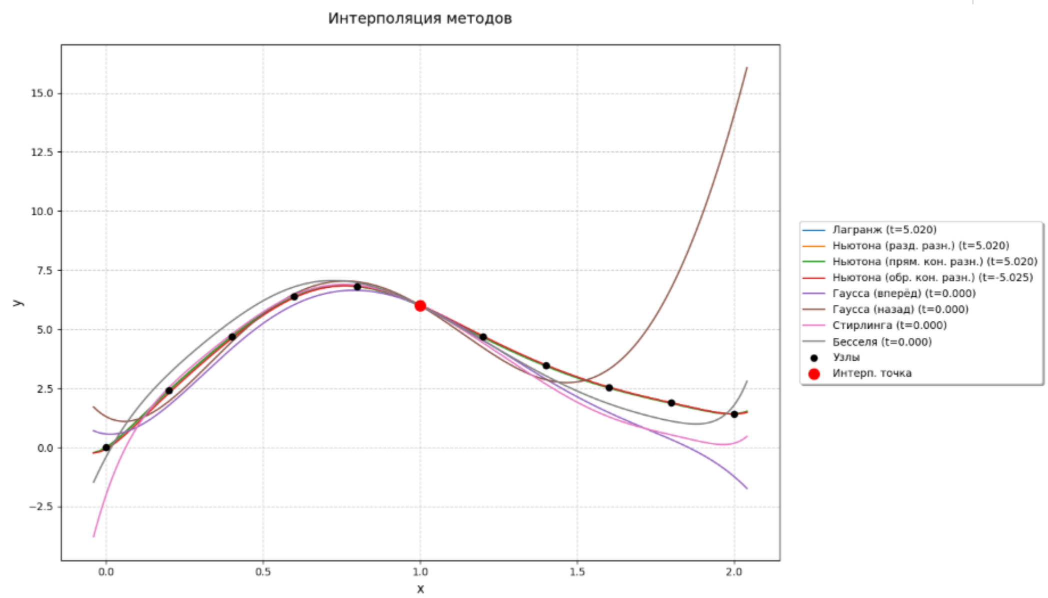
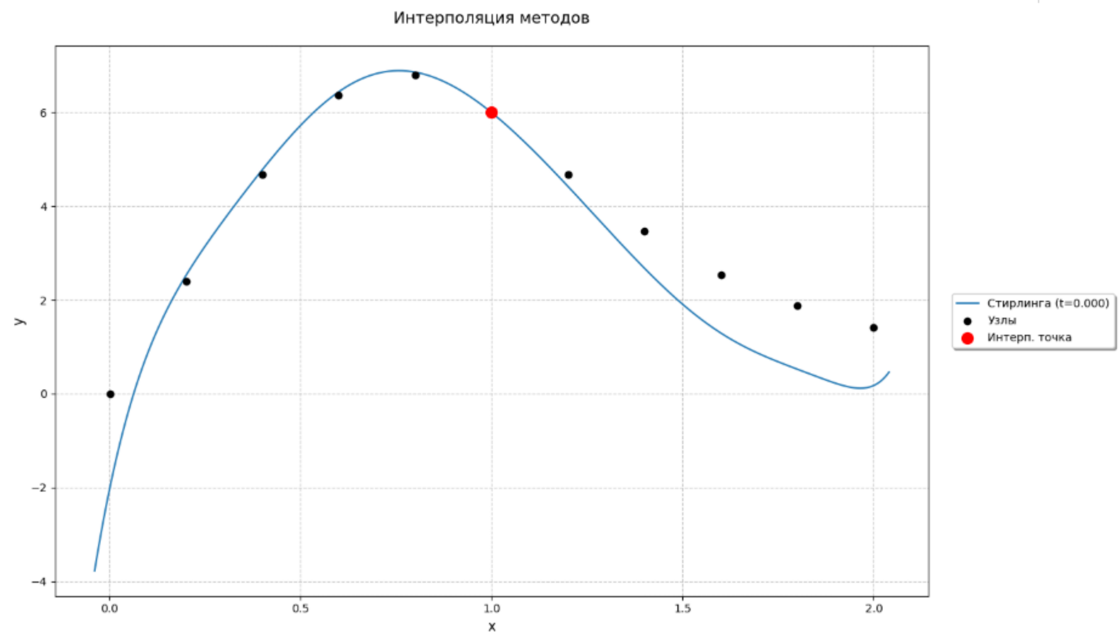


Таблица конечных разностей

Δ0	Δ1	Δ2	Δ3	Δ4	Δ5	Δ6	Δ7	Δ8	Δ9	Δ10
0.0010	2.3960	4.6800	6.3740	6.8100	6.0000	4.6850	3.4700	2.5420	1.8790	1.4120
2.3950	2.2840	1.6940	0.4360	-0.8100	-1.3150	-1.2150	-0.9280	-0.6630	-0.4670	
-0.1110	-0.5900	-1.2580	-1.2460	-0.5050	0.1000	0.2870	0.2650	0.1960		
-0.4790	-0.6680	0.0120	0.7410	0.6050	0.1870	-0.0220	-0.0690			
-0.1890	0.6800	0.7290	-0.1360	-0.4180	-0.2090	-0.0470				

Лагранж Ньютона (разд. разн.) Ньютона (прям. кон. разн.) Ньютона (обр. кон. разн.) Гаусса (вперёд) Гаусса (назад) Стирлинга Бесселя

Стирлинга



P(x) = 6.000000

t = 0.000

Вывод:

В ходе выполнения лабораторной работы я изучил и практично применил интерполяционные методы Ньютона и Гаусса для обработки табличных данных. Эти методы позволили аппроксимировать значения функции в точках, отсутствующих в исходном наборе.

Разработанная программа обеспечила вычисление значений функции в заданных точках с использованием обоих методов. Проведённый сравнительный анализ показал совпадение результатов, что свидетельствует о правильной реализации и работоспособности выбранных алгоритмов.