Министерство науки и высшего образования РФ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Казанский (Приволжский) федеральный университет»
Институт вычислительной математики и информационных технологий

ОТЧЕТ

На тему: «Интерполирование трансцендентных функций алгебраическими многочленами»

Выполнил:
студент группы 09-222 Шпак В.С.
Проверил:
ассистент Глазырина О.В.

Казань, 2023 год

СОДЕРЖАНИЕ

Оглавление

ЦЕЛЬ РАБОТЫ

Изучить поведение погрешности при интерполяции трансцендентных функций алгебраическими многочленами на примере интерполирования интегрального синуса интерполяционными полиномами Лагранжа и Ньютона при равномерном и неравномерном распределениях

ИСХОДНЫЕ ДАННЫЕ И ЗАДАНИЕ

1) Найти узлы при равномерном распределении
2) Найти узлы Чебышева
3) Вычислить значения функции в узлах при различных распределениях
4) Вычислить интерполяционный полином Лагранжа при равно распределенных узлах
5) Вычислить погрешность для полинома Лагранжа при равно распределенных узлах
6) Вычислить интерполяционный полином Ньютона при равно распределенных узлах
7) Вычислить погрешность для полинома Ньютона при неравно распределенных узлах
8) Вычислить интерполяционный полином Лагранжа при неравно распределенных узлах
9) Вычислить погрешность для полинома Лагранжа при неравно распределенных узлах
10) Вычислить интерполяционный полином Ньютона при неравно распределенных узлах
11) Вычислить погрешность для полинома Ньютона при неравно распределенных узлах
12) Сравнить погрешности при разных интерполяционных полиномах и различных распределениях
13) Исследовать зависимость максимальной погрешности от количества узлов при использовании различных интерполяционных полиномах и при равно распределенных узлах
14) Исследовать зависимость максимальной погрешности от количества узлов при использовании различных интерполяционных полиномах и при неравно распределенных узлах
15) Сравнить зависимости максимальной погрешности при разных типах распределений

ХОД РАБОТЫ

ТАБУЛЯЦИЯ ФУНКЦИИ

Для того чтобы про табулировать функцию на отрезке [a, b] с шагом h использовался предварительно, вычисленный ряд Тейлора формула 1.

$$Si(x) = 1)\int_0^x \frac{sint}{t}dt = \sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!(2n+1)}x^{2n+1} \quad (1)$$

В ряде Тейлора каждый последующий член a_{n+1} ряда получается умножением предыдущего a_n на величину $q_n = \frac{a_{n+1}}{a_n}$. Для ряда Тейлора интегрального синуса она находиться по формуле 2.

$$q_n = -\frac{x^2(2n+1)}{(2n+3)^2(2n+2)} \quad (2)$$

Такое вычисление членов ряда позволяет избежать переполнения памяти при вычислении функции факториала. Результаты табулирования функции на отрезке [0,4] с шагом 0,8 и точностью $\varepsilon = 10^{-6}$ представлены в таблице 1. Рис. 1 изображен график, построенный на основе полученных значений.

Таблица 1 — Протабулированные значения x в 6 точках

x_n	$f(x_n)$
0	0
0.8	0.772096
1.6	1.389181
2.4	1.752486
3.2	1.851401
4	1.758203

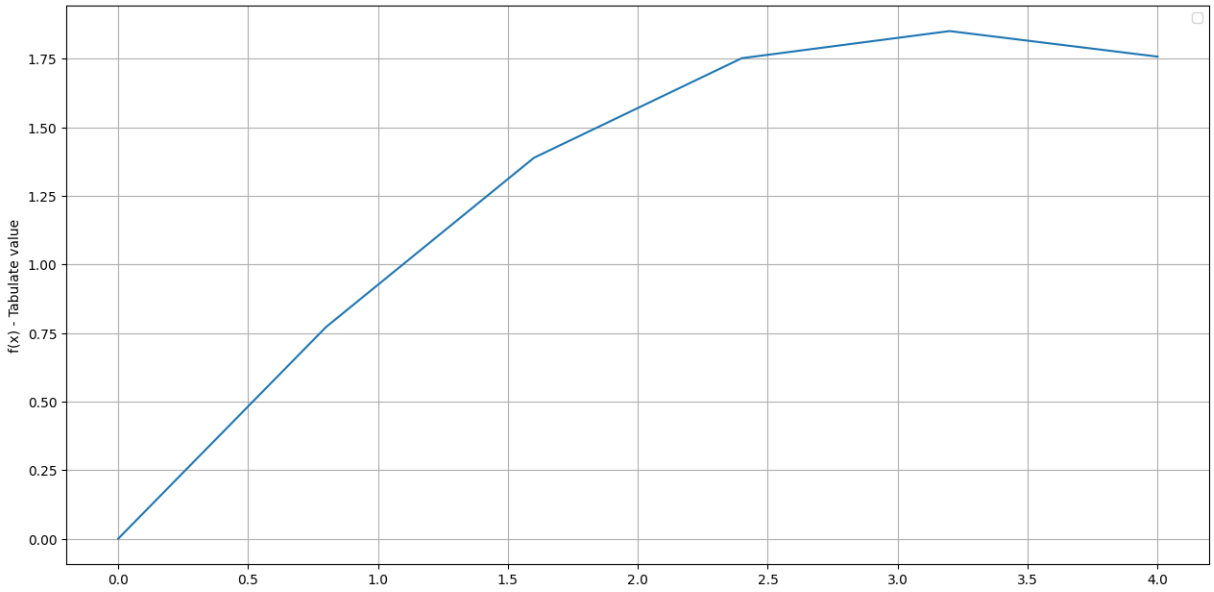

Рисунок 1 График табуляции в 6 узлах

РАВНОМЕРНОЕ РАСПРЕДЕЛЕНИЕ

ВЫЧИСЛЕНИЕ ПОЛИНОМА В ФОРМЕ ЛАГРАНЖА

Для построения интерполяционного полинома Лагранжа на равномерно распределенных узлах была использована формула 3.

$$P_n = \sum_{i=1}^{N} f_i \prod_{j=1}^{N} \frac{x - x_i}{x_i - x_j} \quad (3)$$

Результаты построения полинома в 11 точках с использованием 6 узлов, протабулированные значения 11 узлов и их погрешность представлены в таблице 2. На рис. 2 график, построенный на основе полученных данных. На рис. 3 график построенный по погрешности интерполирования полиномом Лагранжа

Таблица 2 — Интерполяция полиномом Лагранжа и ее погрешность

x_i	$L(x_i)$	$f(x_i)$	$\lvert L(x_i) - f(x_i) \rvert$
0.0	0.000000	0.000000	0.000000
0.4	0.395663	0.396461	0.000798
0.8	0.772096	0.772096	0.000000
1.2	1.108316	1.108047	0.000268
1.6	1.389181	1.389181	0.000000
2.0	1.605223	1.605413	0.000190
2.4	1.752485	1.752486	0.000000
2.8	1.832355	1.832097	0.000259
3.2	1.851401	1.851401	0.000000
3.6	1.821205	1.821948	0.000743
4.0	1.758203	1.758203	0.000000

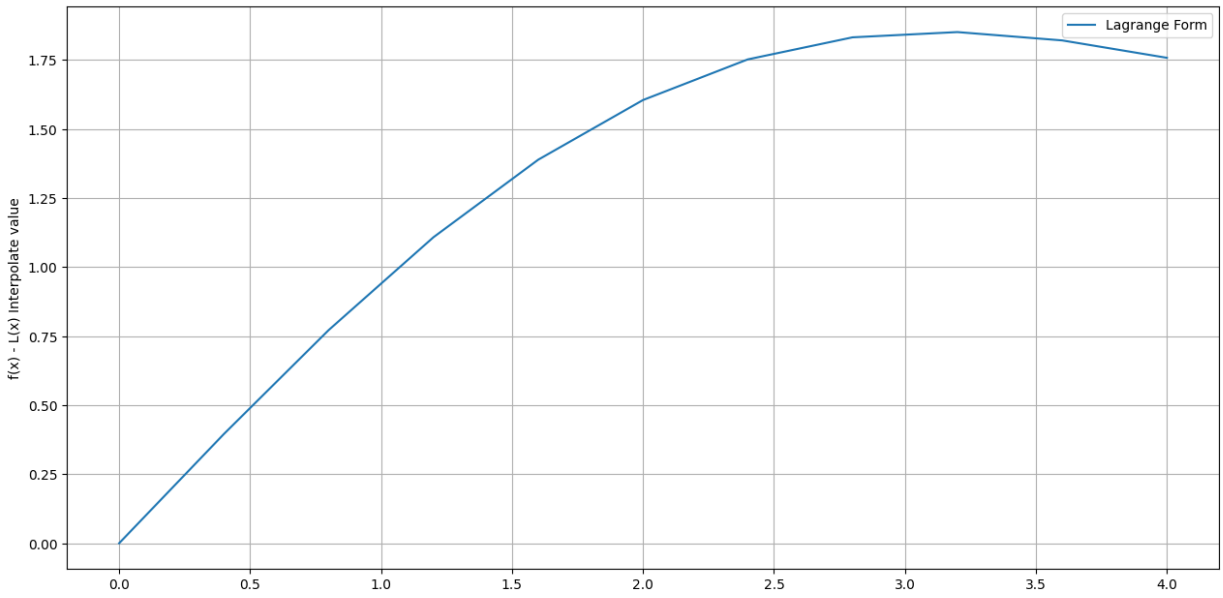

Рисунок 2 График функции приближенной полиномом Лагранжа

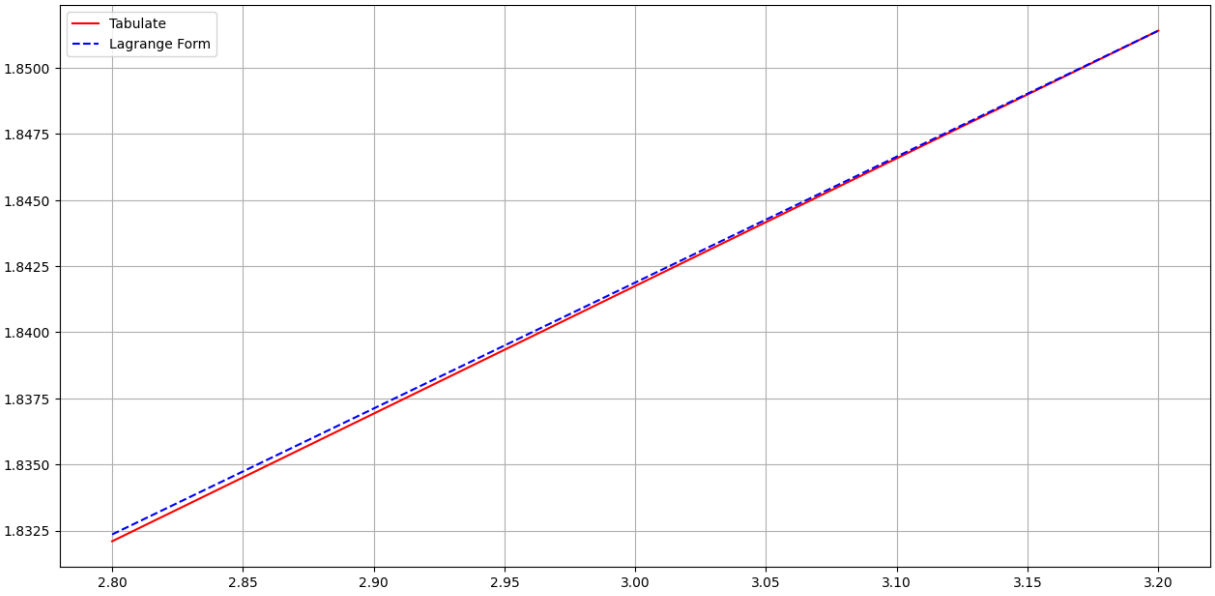

Рисунок 3 График погрешности в узлах 2.8 и 3.2

ВЫЧИСЛЕНИЕ ПОЛИНОМА В ФОРМЕ НЬЮТОНА

Для построения интерполяционного полинома Ньютона на равномерно распределенных узлах была использована формула 4, где коэффициенты a_j – определяются как разделенные разности $[y_0, \dots, y_j]$.

$$N(x) = \sum_{j=0}^{k} a_j \prod_{i=0}^{j-1}(x - x_i) \quad (4)$$

Результаты построения полинома в 11 точках с использованием 6 узлов, протабулированные значения 11 узлов и их погрешность представлены в таблице 3. На рис.4 график, построенный на основе полученных данных. На рис. 5 график построенный по погрешности интерполирования полиномом Ньютона.

Таблица 3 — Интерполяция полиномом Ньютона и ее погрешность

x_i	$N(x_i)$	$f(x_i)$	$\lvert N(x_i) - f(x_i)\rvert$
0.0	0.000000	0.000000	0.000000
0.4	0.395663	0.396461	0.000798
0.8	0.772096	0.772096	0.000000
1.2	1.108316	1.108047	0.000268
1.6	1.389181	1.389181	0.000000
2.0	1.605223	1.605413	0.000190
2.4	1.752486	1.752486	0.000000
2.8	1.832355	1.832097	0.000259
3.2	1.851401	1.851401	0.000000
3.6	1.821205	1.821948	0.000743
4.0	1.758203	1.758203	0.000000

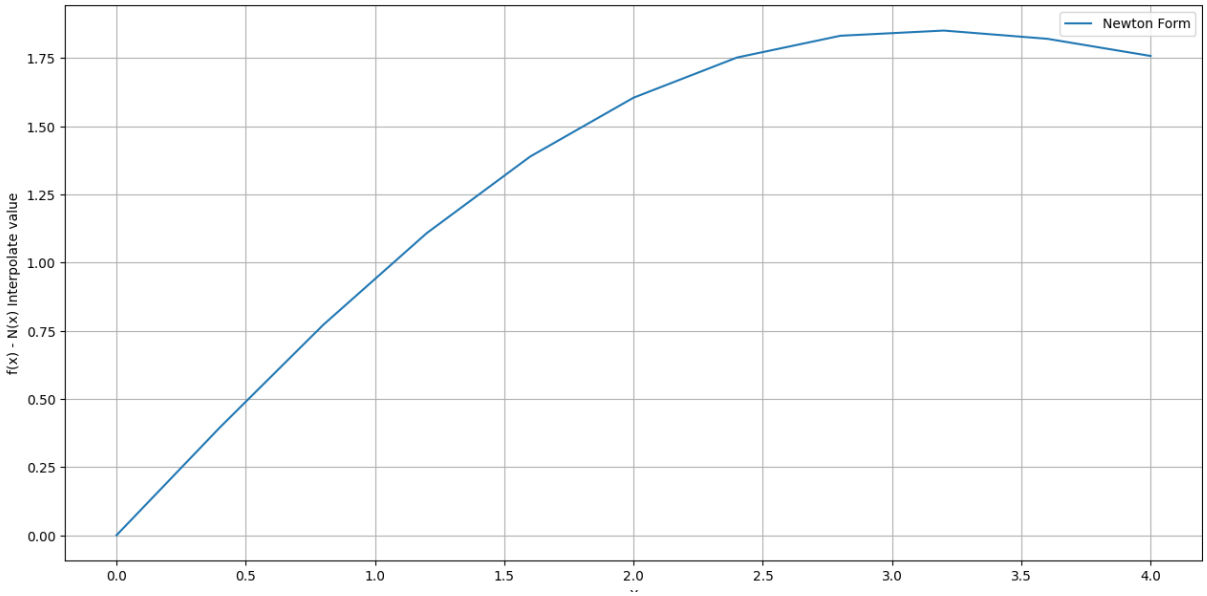

Рисунок 4 График приближения функции полиномом Ньютона

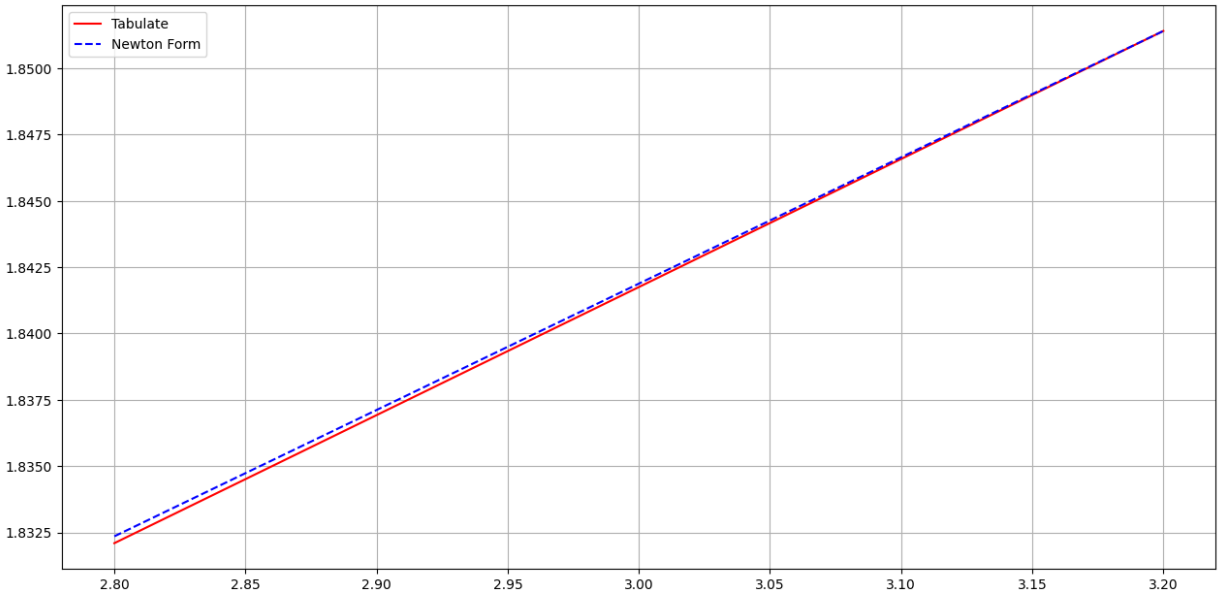

Рисунок 5 График погрешности в узлах 2.8 и 3.2

СРАВНЕНИЕ ПРИБЛИЖЕНИЙ В ФОРМАХ ЛАГРАНЖА И НЬЮТОНА

Для удобства в таблице 4 записаны данные, полученные на ранних этапах. Сравнив значения погрешностей интерполяционных, становится понятно, что они равны. Из равенства погрешностей следует равенство самих интерполяционных полиномов, следовательно можно сказать, что интерполяционные полиномы Лагранжа и Ньютона на самом деле равны друг другу при равномерном распределении узлов. Также на основе таблицы 4 построены графики на рис. 6, рис. 7.

Таблица 4 — Сравнение погрешностей

x_i	$f(x_i)$	$L(x_i)$	$N(x_i)$	$\lvert L(x_i) - f(x_i)\rvert$	$\lvert N(x_i) - f(x_i)\rvert$
0.0	0.000000	0.000000	0.000000	0.000000	0.000000
0.4	0.396461	0.395663	0.395663	0.000798	0.000798
0.8	0.772096	0.772096	0.772096	0.000000	0.000000
1.2	1.108047	1.108316	1.108316	0.000268	0.000268
1.6	1.389181	1.389181	1.389181	0.000000	0.000000
2.0	1.605413	1.605223	1.605223	0.000190	0.000190
2.4	1.752486	1.752485	1.752486	0.000000	0.000000
2.8	1.832097	1.832355	1.832355	0.000259	0.000259
3.2	1.851401	1.851401	1.851401	0.000000	0.000000
3.6	1.821948	1.821205	1.821205	0.000743	0.000743
4.0	1.758203	1.758203	1.758203	0.000000	0.000000

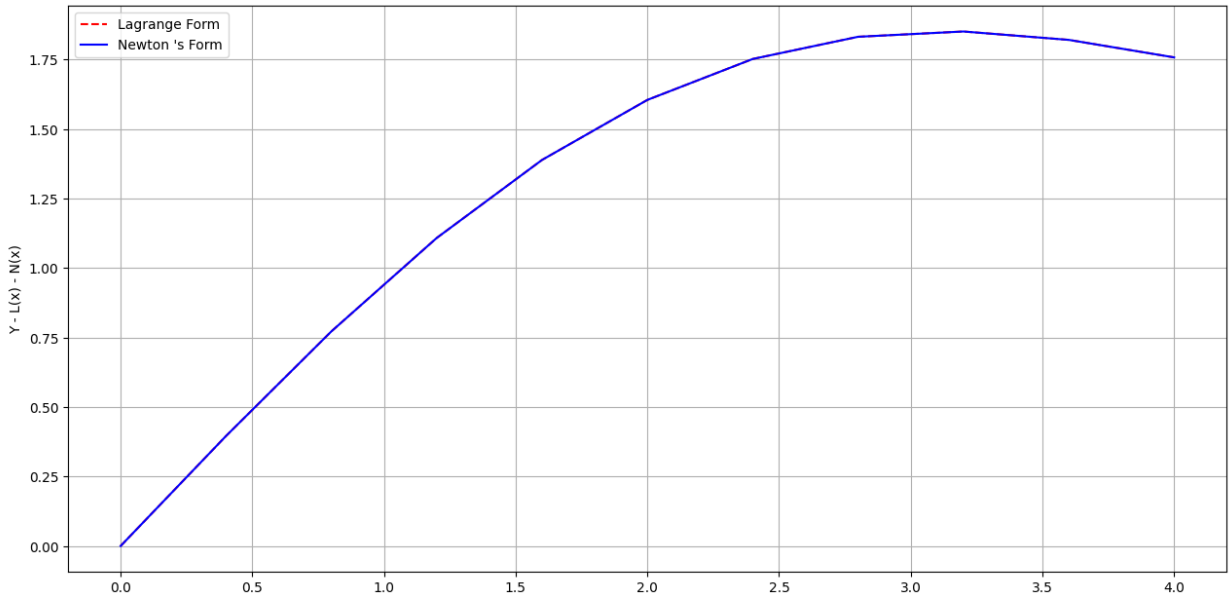

Рисунок 6 График приближенных полиномами Ньютона и Лагранжа значений

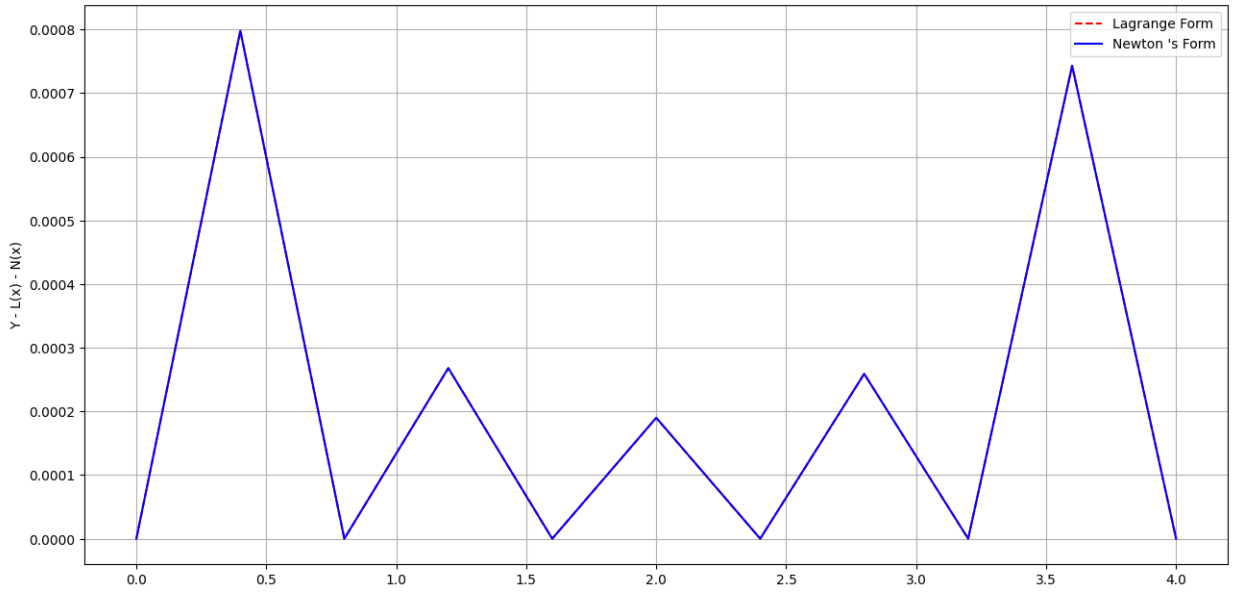

Рисунок 7 График погрешности значений интерполяционных полиномов Лагранжа и Ньютона

ИССЛЕДОВАНИЕ ЗАВИСИМОСТИ ПОГРЕШНОСТИ ОТ КОЛИЧЕСТВА УЗЛОВ

Для того чтобы исследовать зависимость погрешности нужно найти значения погрешностей при разном количестве узлов. Для упрощения визуализации дополнительно будем находить максимальную погрешность. В процессе исследования было выявлено, что при приближении количества узлов к 45 значение максимальной погрешности начинает резко расти, а при 50 узлах отклонении интерполяции кратно превышает значении функции. Также на участке от 8 до 24 погрешность минимальна т.е. такое начальное количество узлов является лучшим для интерполирования. Все это отлично отображают графики на рис.8, рис. 9.

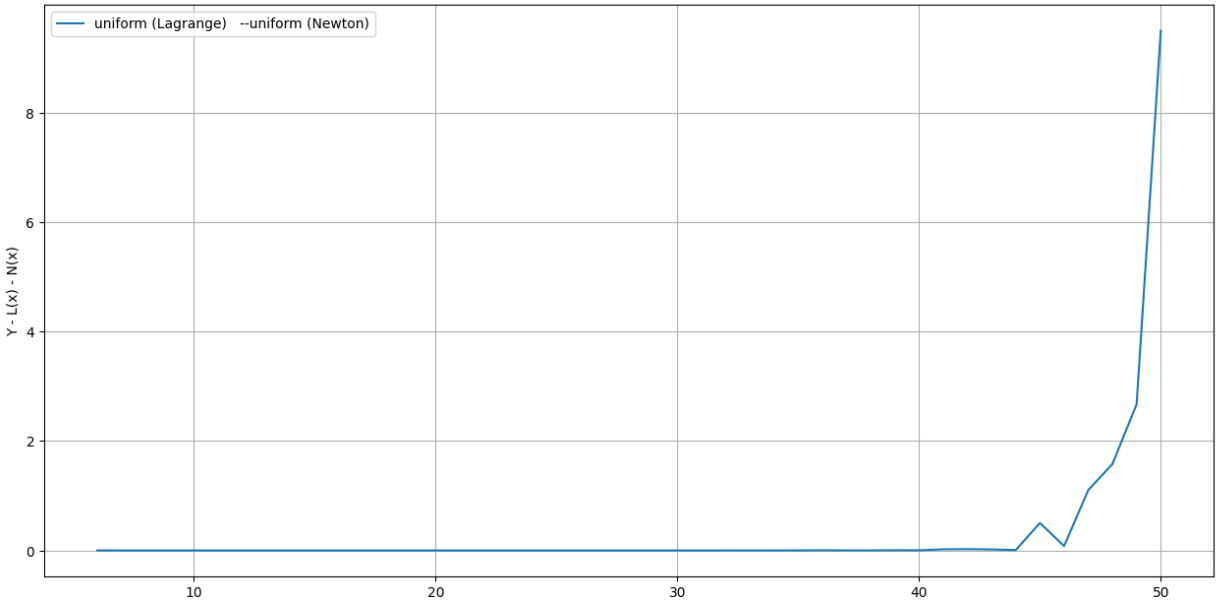

Рисунок 8 График максимальной погрешности при количестве узлов не более 50

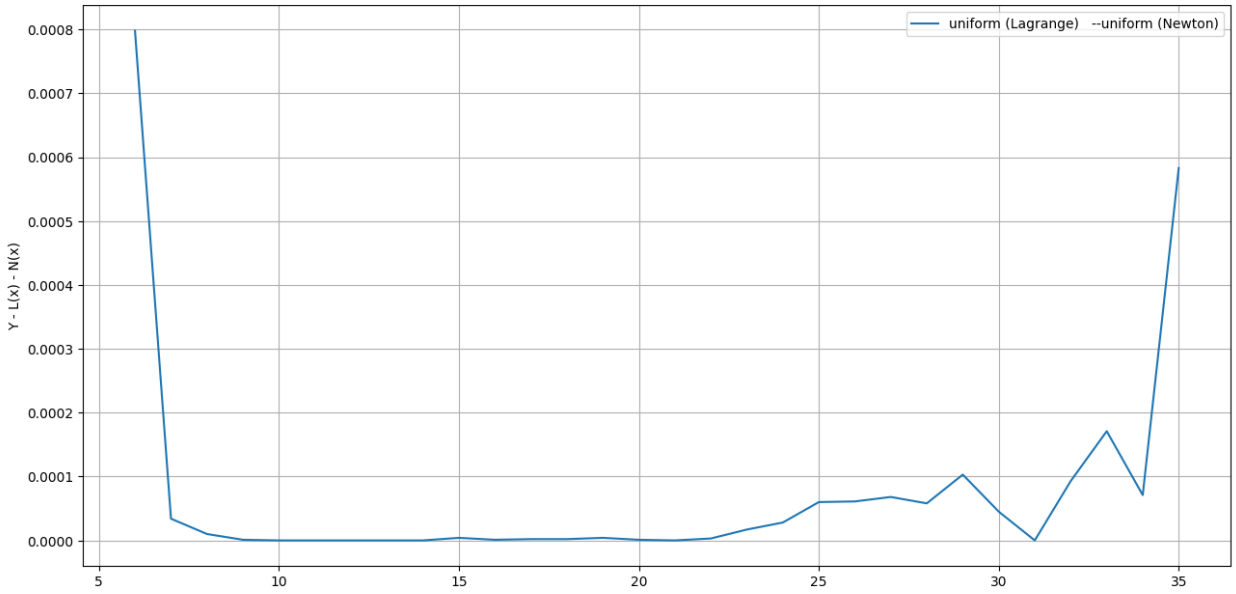

Рисунок 9 График максимальной погрешности при количестве узлов не более 35

ВЫЧИСЛЕНИЕ ПОЛИНОМА В ФОРМЕ ЛАГРАНЖА

Результаты построения полинома в 11 точках с использованием 6 узлов, протабулированные значения 11 узлов и их погрешность представлены в таблице 4. График на рис. 10 построен на основе полученных данных. График на рис. 11 построен по погрешности интерполирования полиномом Лагранжа

Таблица 5 — Интерполяция полиномом Лагранжа и ее погрешность

x_i	$L(x_i)$	$f(x_i)$	$\lvert L(x_i) - f(x_i) \rvert$
0.020357	0.020357	0.020357	0.000266
0.180736	0.180408	0.180408	0.000343
0.488501	0.482071	0.482071	0.000171
0.918718	0.876714	0.876714	0.000397
1.436535	1.281688	1.281688	0.000059
2.000000	1.605413	1.605413	0.000410
2.563465	1.792879	1.792879	0.000058
3.081282	1.851351	1.851351	0.000378
3.511499	1.831957	1.831957	0.000160
3.819264	1.790258	1.790258	0.000316
3.979643	1.762030	1.762030	0.000243

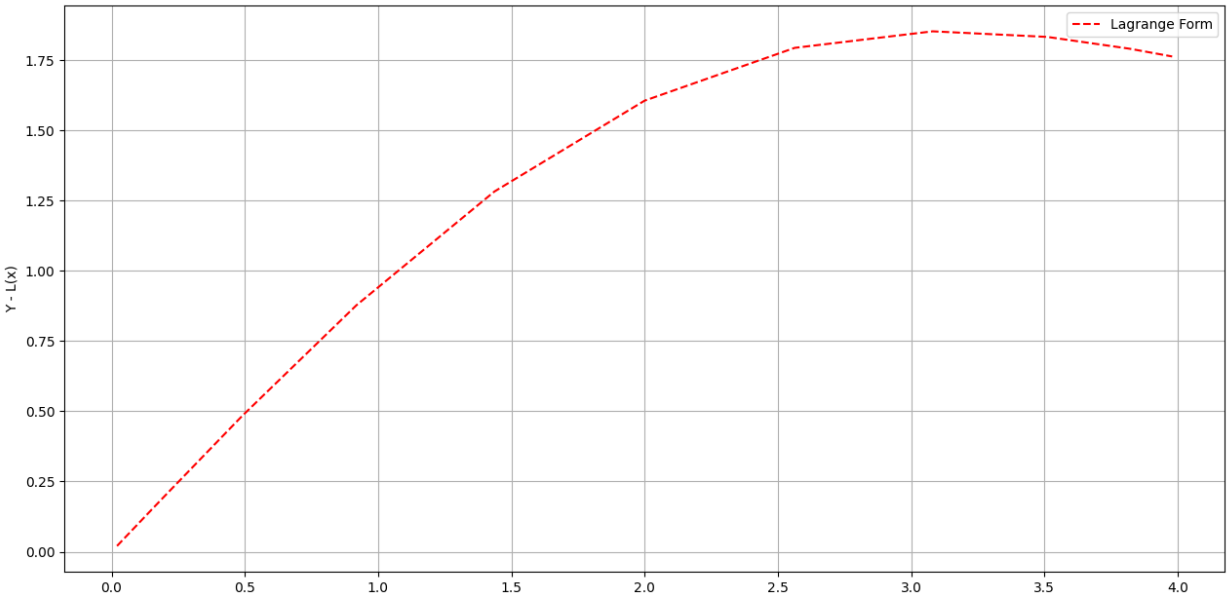

Рисунок 10 График функции приближенной полиномом Лагранжа

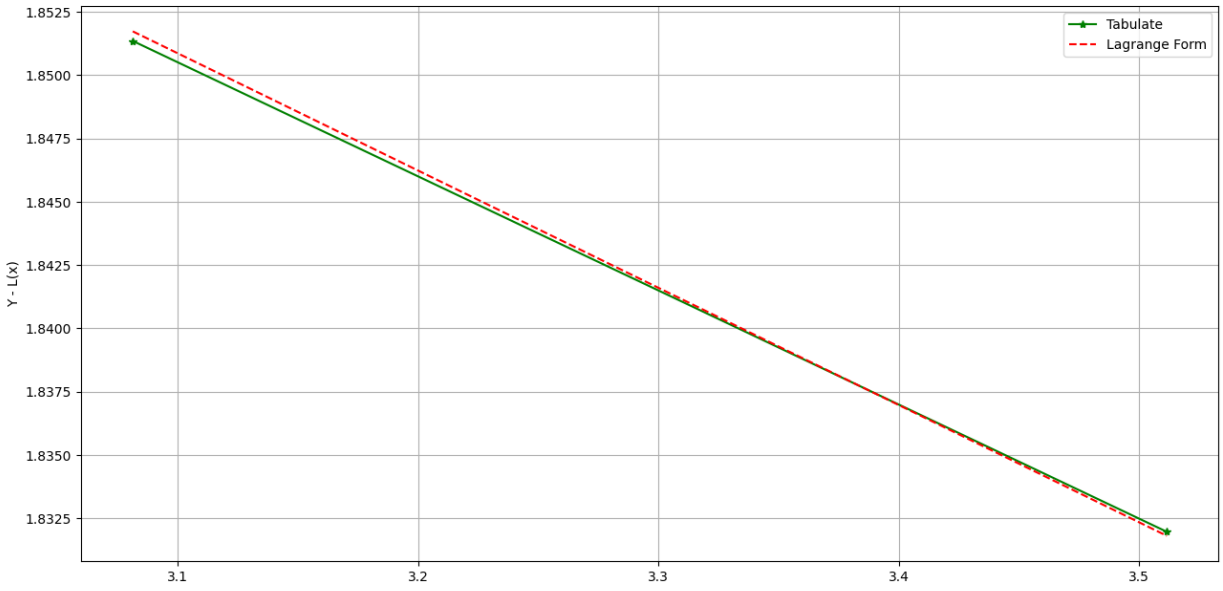

Рисунок 11 График погрешности в узлах 3.081282, 3.511499

ВЫЧИСЛЕНИЕ ПОЛИНОМА В ФОРМЕ НЬЮТОНА

Результаты построения полинома в 11 точках с использованием 6 узлов, протабулированные значения 11 узлов и их погрешность представлены в таблице 5. График на рис. 12 построен на основе полученных данных. График на рис. 13 построен по погрешности интерполирования полиномом Ньютона.

Таблица 6 — Интерполяция полиномом Ньютона и ее погрешность

x_i	$N(x_i)$	$f(x_i)$	$\lvert N(x_i) - f(x_i) \rvert$
0.020357	0.020622	0.020357	0.000266
0.180736	0.180065	0.180408	0.000343
0.488501	0.481900	0.482071	0.000171
0.918718	0.877110|	0.876714	0.000397
1.436535	1.281747	1.281688	0.000059
2.000000	1.605003	1.605413	0.000410
2.563465	1.792936	1.792879	0.000058
3.081282	1.851729	1.851351	0.000378
3.511499	1.831798	1.831957	0.000160
3.819264	1.789941	1.790258	0.000317
3.979643	1.762273	1.762030	0.000243

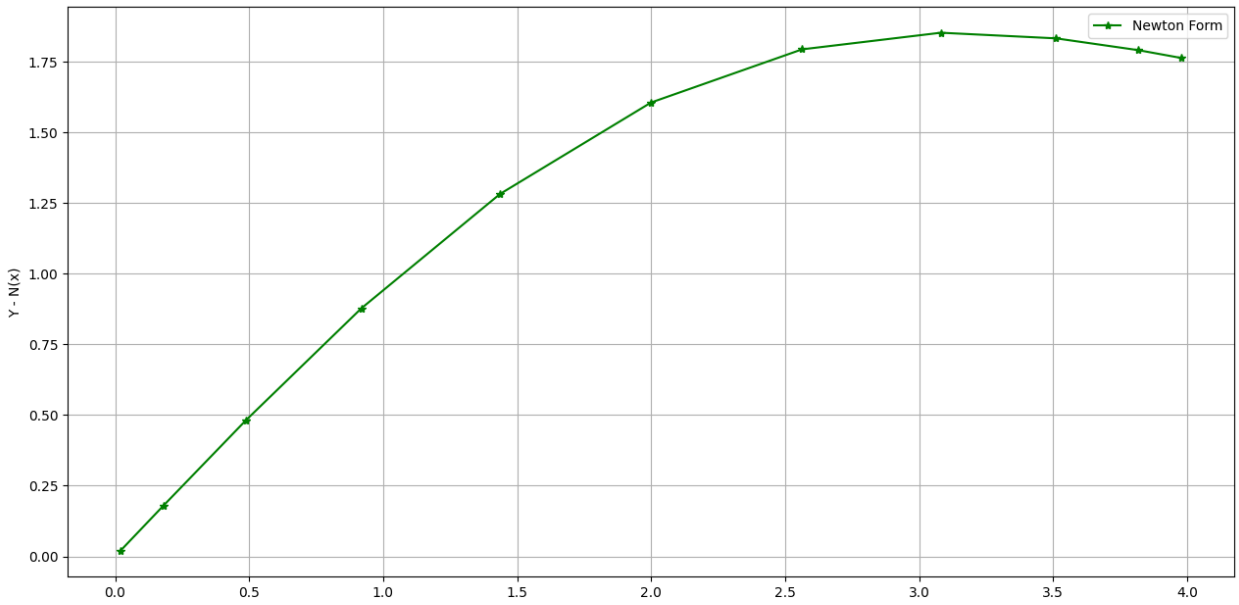

Рисунок 12 График функция приближенной полиномом Ньютона

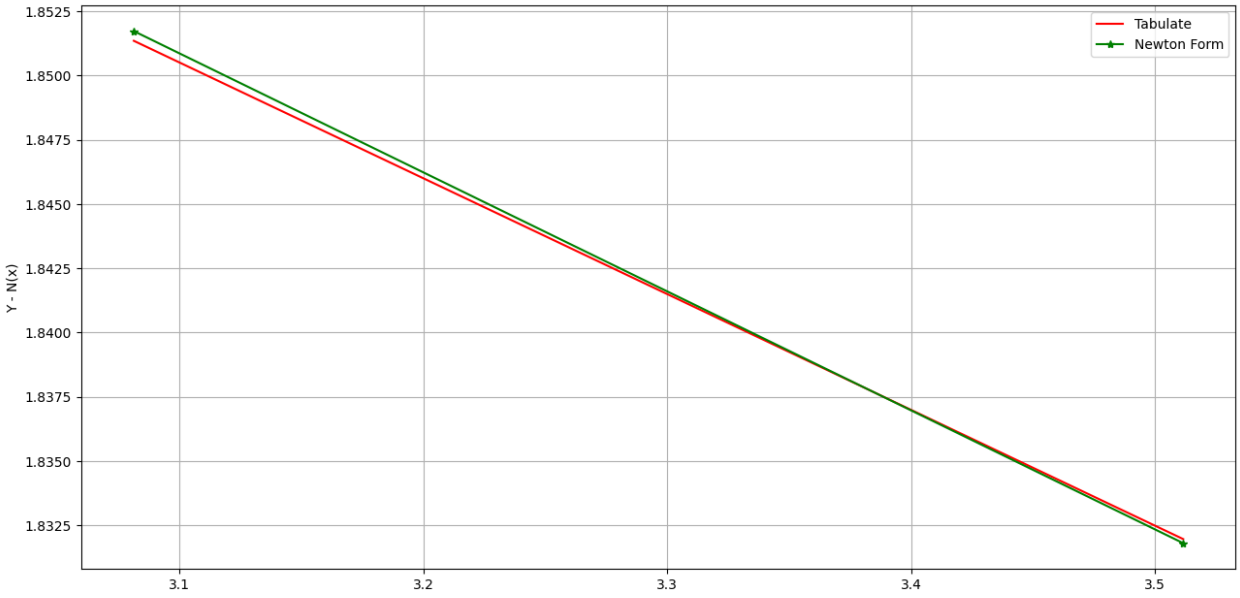

Рисунок 13 График погрешности в узлах 3.081282, 3.511499

СРАВНЕНИЕ ПРИБЛИЖЕНИЙ В ФОРМАХ ЛАГРАНЖА И НЬЮТОНА

Для удобства в таблице 8 записаны данные, полученные на ранних этапах. Сравнив значения погрешностей интерполяционных, становится понятно, что они равны. Из равенства погрешностей следует равенство самих интерполяционных полиномов, следовательно можно сказать, что интерполяционные полиномы Лагранжа и Ньютона на самом деле равны друг другу при неравномерном распределении узлов. Также на основе таблицы 8 построены графики на рис. 14, рис. 15.

Таблица 7 — Сравнение погрешностей

x_i	$f(x_i)$	$L(x_i)$	$N(x_i)$	$\|L(x_i) - f(x_i)\|$	$\|N(x_i) - f(x_i)\|$
0.020357	0.020357	0.020357	0.020622	0.000266	0.000266
0.180736	0.180408	0.180408	0.180065	0.000343	0.000343
0.488501	0.482071	0.482071	0.481900	0.000171	0.000171
0.918718	0.876714	0.876714	0.877110|	0.000397	0.000397
1.436535	1.281688	1.281688	1.281747	0.000059	0.000059
2.000000	1.605413	1.605413	1.605003	0.000410	0.000410
2.563465	1.792879	1.792879	1.792936	0.000058	0.000058
3.081282	1.851351	1.851351	1.851729	0.000378	0.000378
3.511499	1.831957	1.831957	1.831798	0.000160	0.000160
3.819264	1.790258	1.790258	1.789941	0.000316	0.000317
3.979643	1.762030	1.762030	1.762273	0.000243	0.000243

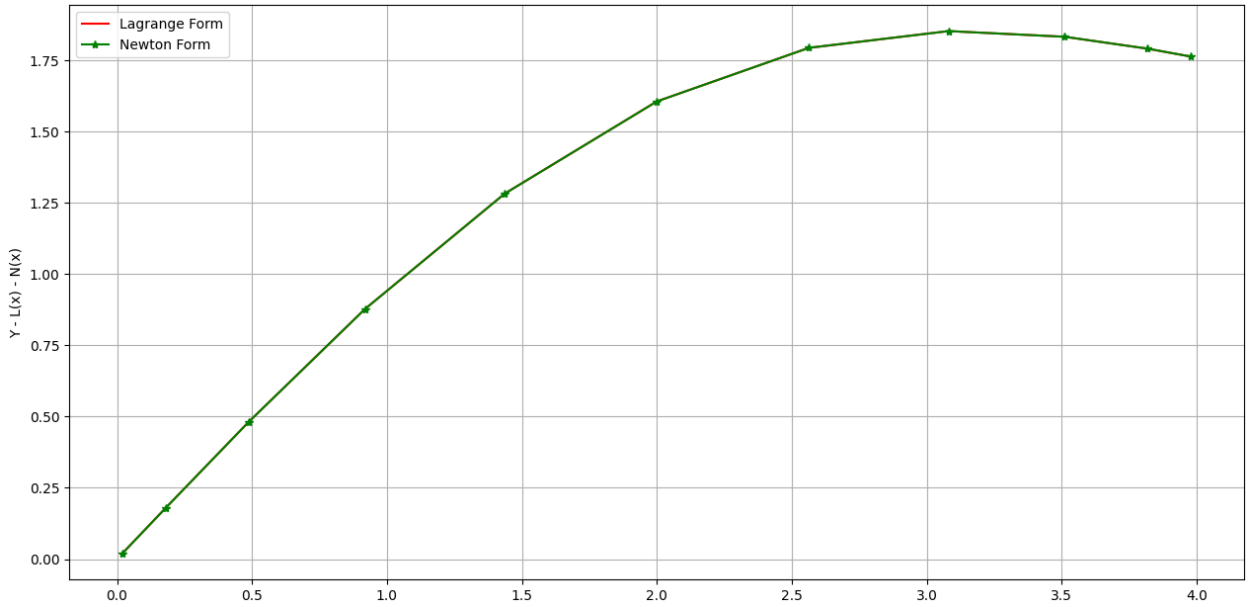

Рисунок 14 График приближенных полиномами Ньютона и Лагранжа значений

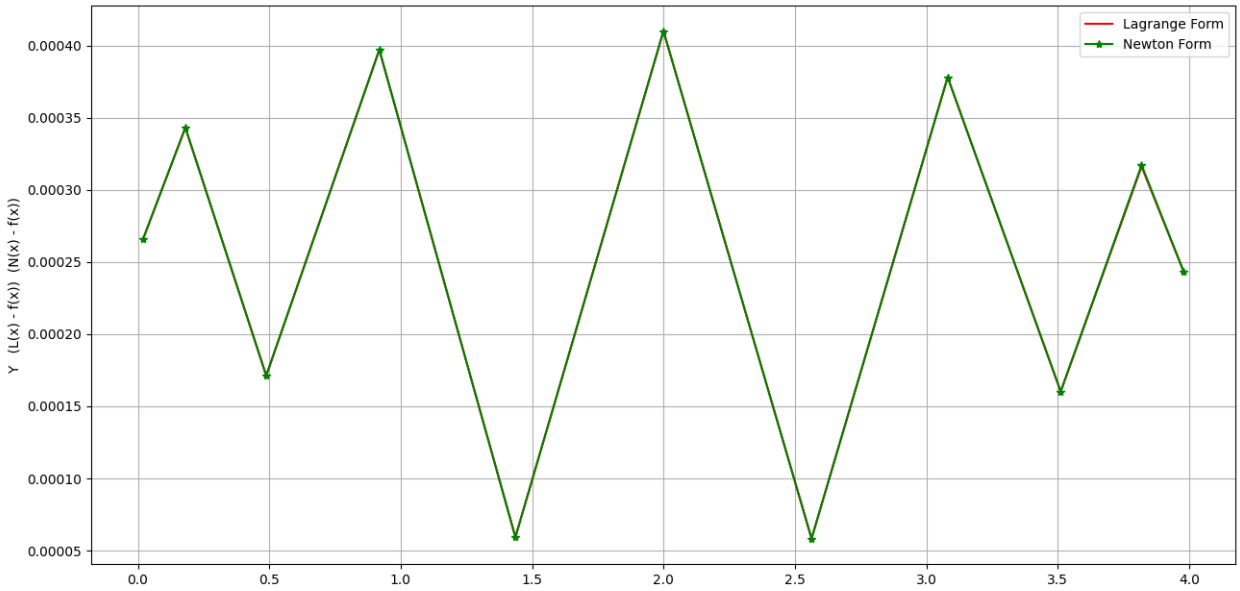

Рисунок 15 График погрешности значений интерполяционных полиномов Лагранжа и Ньютона

ИССЛЕДОВАНИЕ ЗАВИСИМОСТИ ПОГРЕШНОСТИ ОТ КОЛИЧЕСТВА УЗЛОВ

Для того что исследовать зависимость погрешности нужно найти значения погрешностей при разном количестве узлов. Для упрощения визуализации так как и в случае с равномерно распределенными узлами дополнительно будем находить максимальную погрешность. В процессе исследования было выявлено, что погрешность интерполирования как полиномом Лагранжа, так и полиномом Ньютона стремиться к нулю при неравномерно распределенных узлах. Все это отлично отображает график на рис. 16.

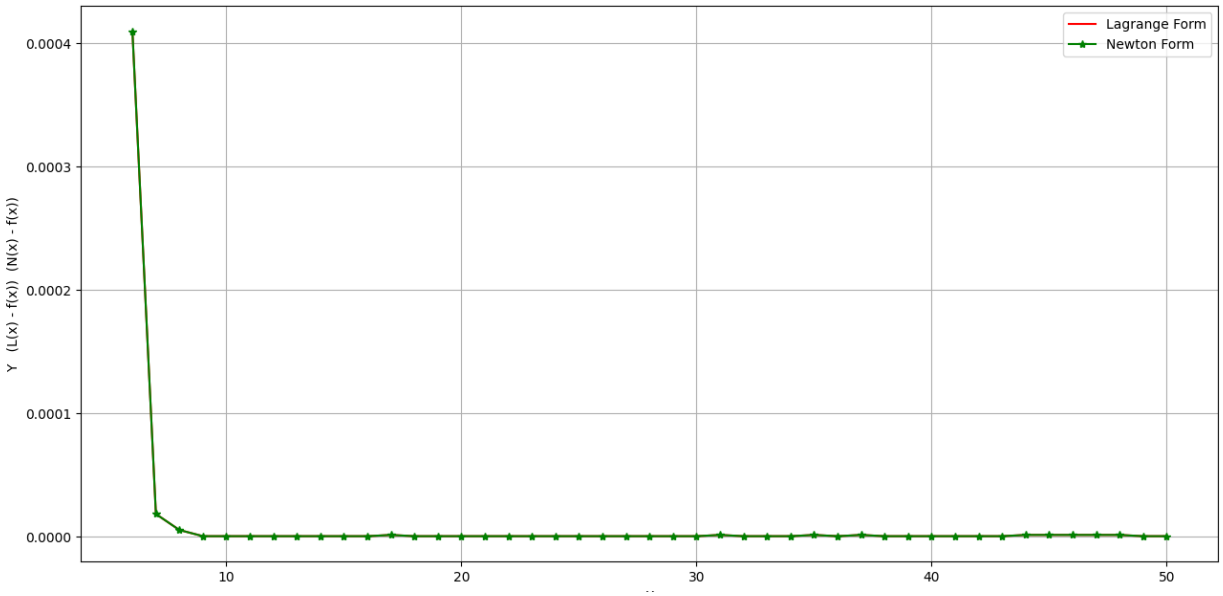

Рисунок 16 График максимальной погрешности при количестве узлов не более 50

СРАВНЕНИЕ РАЗБИЕНИЙ НА РАВНОМЕРНО РАСПРЕДЕЛЕННЫЕ И НА УЗЛЫ ЧЕБЫШЕВА

Для удобства сравнения графиков погрешностей наложим их друг на друга на графике изображенном на рис. 17. Как видно на графике погрешность при равномерном распределении выше таковой при распределении на узлы Чебышева (неравномерное распределение). Исключением служат точки, совпадающие с известными протабулированными значениями узлов, к примеру на графике это узлы 0, 0.4, …, 4. В этих узлах погрешность при неравномерном распределении будет выше, чем при равномерном распределении.

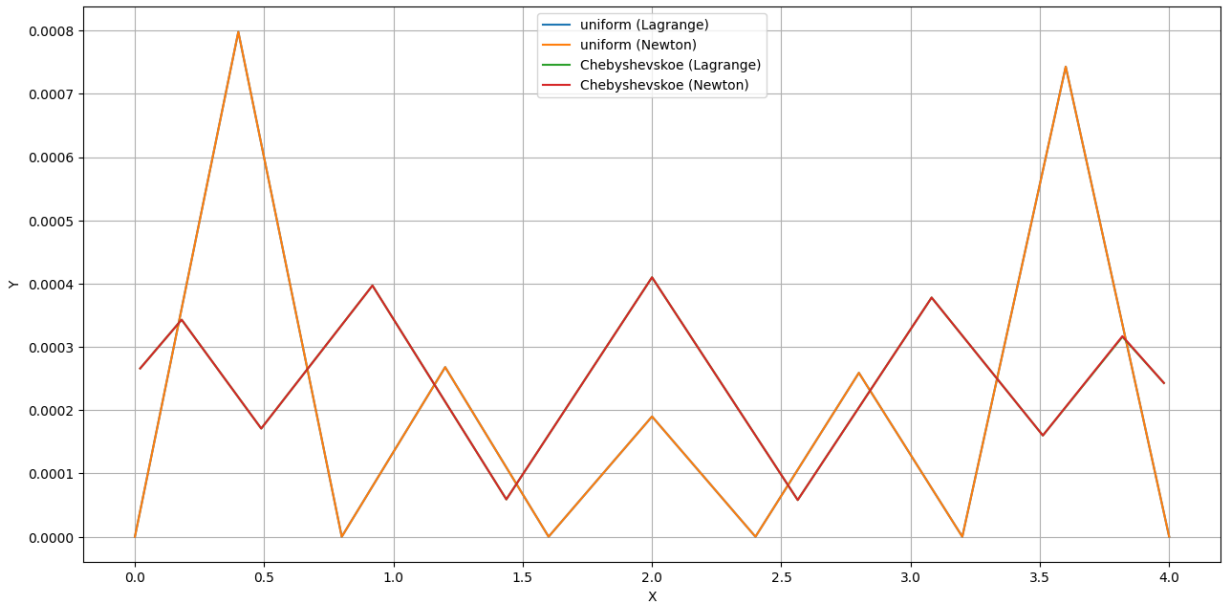

Рисунок 17 График наложения погрешностей при различный распределениях и интерполяциях

ВЫВОДЫ ПО РАБОТЕ

В ходе проделанной работы были построены интерполяционные полиномы Лагранжа, Ньютона для функции интегрального синуса на основе равно распределённых узлов интерполяции и узлов Чебышева. Экспериментальным путем доказано, что максимальная погрешность при использовании неравно распределённых ниже, чем таковая при равно распределенных узлах при интерполяции как полиномом Лагранжа, так и Ньютона. Исследование максимальной погрешности при увеличении узлов показывает, что при увеличении количества узлов начального отрезка равномерно распределенных в обоих случаях интерполирования полиномами Лагранжа и Ньютона приводит к кратному росту погрешности, к примеру, при 45 начальных узлах значение погрешности превышает среднее значении погрешности на интервале от 5 до 40 примерно в два раза, при 50 узлах значение погрешности уже превышает значение самой функции примерно в 8 раз. Интервал количества узлов от 5 до 24 является лучшим для интерполирования исследуемой функции при 11 значениях в интерполяционном полиноме т. к. погрешность на данном интервале минимальная с 25 узлов погрешность начинает расти. При использовании узлов Чебышева погрешность при использовании в качестве интерполяционных полиномов, полиномов Лагранжа и Ньютона постоянно уменьшается. Также экспериментальным путем доказано, что интерполяционные полиномы Лагранжа и Ньютона при не большом количестве узлов интерполяции равны.

ЛИСТИНГ

```
float Get_Step(float beginPoint, float endPoint, int interavalCount)
{
	return (endPoint - beginPoint) / interavalCount;
}

std::vector<float> Get_Nodes_Row(float beginPoint, float endPoint, int
interavalCount)
{
	std::vector<float> Nodes;
	float step = Get_Step(beginPoint, endPoint, interavalCount);
	for (float x = beginPoint; x <= endPoint; x += step)
	{
		Nodes.push_back(x);
	}
	if (Nodes.size() != interavalCount + 1)
	{
		Nodes.push_back(endPoint);
	}
	return Nodes;
}

std::vector<float> Get_Nodes_Cheb(float beginPoint, float endPoint, int
interavalCount)
{
	std::vector<float> Nodes = Get_Nodes_Row(beginPoint, endPoint,
interavalCount);
	std::vector<float> Nodes_Cheb;
	for (int i = 0; i < Nodes.size(); i++)
	{
		float a = Nodes[0], b = Nodes[Nodes.size() - 1], n = Nodes.size() - 1;
		Nodes_Cheb.push_back(0.5 * (a + b) + 0.5 * (b - a) * cos(((2 * i + 1) /
(2 * n + 2)) * M_PI));
	}
	std::reverse(Nodes_Cheb.begin(), Nodes_Cheb.end());
	return Nodes_Cheb;
}

std::vector<std::vector<float>> Tabulate(std::vector<float> Nodes_R_CH)
{
	std::vector<float> ValueNodes;
	std::vector<std::vector<float>> result;
	for (int i = 0; i < Nodes_R_CH.size(); i++)
	{
		float n = 0,x = Nodes_R_CH[i], a = x, sum = a, q = 0;
		while (fabs(a) > 1e-6)
		{
			q = -(x * x * (2 * n + 1)) / ((2 * n + 3) * (2 * n + 3) * (2 * n
+ 2));
			a *= q;
			sum += a;
			n++;
		}
		ValueNodes.push_back(sum);
	}
	result.push_back(Nodes_R_CH);
	result.push_back(ValueNodes);
	return result;
}
```

```
double Create_Base_Polinom_Lagrange(double x, int indexCur,
std::vector<std::vector<float>> TabulPolinom)
{
	double result = 1;
	int count = TabulPolinom[0].size();
	for (int j = 0; j < count; j++)
	{
		if (j != indexCur)
		{
			result *= (x - TabulPolinom[0][j]);
			result /= (TabulPolinom[0][indexCur] - TabulPolinom[0][j]);
		}
	}
	return result;
}

double Create_Lagrange_Polinom(double x, std::vector<std::vector<float>>
TabulPolinom)
{
	int count = TabulPolinom[0].size();
	double result = 0;
	for (int i = 0; i < count; i++)
	{
		result += TabulPolinom[1][i] * Create_Base_Polinom_Lagrange(x, i,
TabulPolinom);
	}
	return result;
}

std::vector<std::vector<float>> Lagrange_interpolate(std::vector<std::vector<float>>
TabulPolinom, std::vector<float> Points)
{
	std::vector<float> InterPolinomF;
	for(int i = 0; i < Points.size(); i++)
	{
		InterPolinomF.push_back(Create_Lagrange_Polinom(Points[i],
TabulPolinom));
	}
	std::vector<std::vector<float>> interpolate_polinom = { Points, InterPolinomF
};
	return interpolate_polinom;
}

std::vector<std::vector<float>> Newton_interpolate(std::vector<std::vector<float>>
TabulPolinom, std::vector<float> Points)
{
	std::vector<float> InterPolinomF;
	std::vector<float> raz = TabulPolinom[1];
	for (float k = 1; k < TabulPolinom[0].size(); k++)
	{
		for (float i = TabulPolinom[0].size() - 1; i > k - 1; i--)
		{
			raz[i] = (raz[i] - raz[i - 1]) / (TabulPolinom[0][i] -
TabulPolinom[0][i - k]);
		}
	}
	for (float j = 0; j < Points.size(); j++)
	{
		InterPolinomF.push_back(raz[0]);
		for (float k = 1; k < TabulPolinom[0].size(); k++)
		{
			float cur = raz[k];
			for (float i = 0; i < k; i++)
			{
				cur *= Points[j] - TabulPolinom[0][i];
```

```
                }
                InterPolinomF[j] += cur;
            }
        }
        std::vector<std::vector<float>> result = { Points, InterPolinomF };
        return result;
}


std::vector<std::vector<float>> Get_Pogreshnost(std::vector<std::vector<float>>
TabulatePolinom, std::vector<std::vector<float>> InterpolatePolinom)
{
        std::vector<float> Pogreshnost;
        for (size_t i = 0; i < InterpolatePolinom[1].size(); i++)
        {
            Pogreshnost.push_back(fabs(TabulatePolinom[1][i] -
InterpolatePolinom[1][i]));
        }
        std::vector<std::vector<float>> result = { TabulatePolinom[0], Pogreshnost };
        return result;
}

float Get_Max_Pogreshnost(std::vector<std::vector<float>> Pogreshnost)
{
        float max = Pogreshnost[1][0], point = Pogreshnost[0][0], count =
Pogreshnost[1].size();
        for (float i = 1; i < count; i++)
        {
            if (max < Pogreshnost[1][i])
            {
                max = Pogreshnost[1][i];
                point = Pogreshnost[0][i];
            }
        }
        return max;
}


std::vector<std::vector<float>> Experiment_Lagrange_Row(std::vector<float> data)
{
        std::vector<std::vector<float>> experiment_result;
        for (float i = data[4]; i < data[5]; i++)
        {
            std::vector<float> nodes = Get_Nodes_Row(data[0], data[1], i);
            std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
            std::vector<float> points_interpolate = Get_Nodes_Row(data[0], data[1],
data[3]);
            std::vector<std::vector<float>> interpolate_polinom =
Lagrange_interpolate(tabulate_polinom, points_interpolate);
            nodes = Get_Nodes_Row(data[0], data[1], data[3]);
            tabulate_polinom = Tabulate(nodes);
            experiment_result.push_back({ i + 1,
Get_Max_Pogreshnost(Get_Pogreshnost(tabulate_polinom, interpolate_polinom)) });
        }
        return experiment_result;
}

std::vector<std::vector<float>> Experiment_Lagrange_Cheb(std::vector<float> data)
{
        std::vector<std::vector<float>> experiment_result;
        for (float i = data[4]; i < data[5]; i++)
        {
            std::vector<float> nodes = Get_Nodes_Cheb(data[0], data[1], i);
            std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
```

```
			std::vector<float> points_interpolate = Get_Nodes_Cheb(data[0],
data[1], data[3]);
			std::vector<std::vector<float>> interpolate_polinom =
Lagrange_interpolate(tabulate_polinom, points_interpolate);
			nodes = Get_Nodes_Cheb(data[0], data[1], data[3]);
			tabulate_polinom = Tabulate(nodes);
			experiment_result.push_back({ i + 1,
Get_Max_Pogreshnost(Get_Pogreshnost(tabulate_polinom, interpolate_polinom)) });
		}
		return experiment_result;
}

std::vector<std::vector<float>> Experiment_Newton_Row(std::vector<float> data)
{
		std::vector<std::vector<float>> experiment_result;
		for (float i = data[4]; i < data[5]; i++)
		{
			std::vector<float> nodes = Get_Nodes_Row(data[0], data[1], i);
			std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
			std::vector<float> points_interpolate = Get_Nodes_Row(data[0], data[1],
data[3]);
			std::vector<std::vector<float>> interpolate_polinom =
Newton_interpolate(tabulate_polinom, points_interpolate);
			nodes = Get_Nodes_Row(data[0], data[1], data[3]);
			tabulate_polinom = Tabulate(nodes);
			experiment_result.push_back({ i + 1,
Get_Max_Pogreshnost(Get_Pogreshnost(tabulate_polinom, interpolate_polinom)) });
		}
		return experiment_result;
}

std::vector<std::vector<float>> Experiment_Newton_Cheb(std::vector<float> data)
{
		std::vector<std::vector<float>> experiment_result;
		for (float i = data[4]; i < data[5]; i++)
		{
			std::vector<float> nodes = Get_Nodes_Cheb(data[0], data[1], i);
			std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
			std::vector<float> points_interpolate = Get_Nodes_Cheb(data[0],
data[1], data[3]);
			std::vector<std::vector<float>> interpolate_polinom =
Newton_interpolate(tabulate_polinom, points_interpolate);
			nodes = Get_Nodes_Cheb(data[0], data[1], data[3]);
			tabulate_polinom = Tabulate(nodes);
			experiment_result.push_back({ i + 1,
Get_Max_Pogreshnost(Get_Pogreshnost(tabulate_polinom, interpolate_polinom)) });
		}
		return experiment_result;
}

std::vector<std::vector<float>> Function_Nodes_Row(std::vector<float> data)
{
		std::vector<float> nodes = Get_Nodes_Row(data[0], data[1], data[2]);
		std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
		return tabulate_polinom;
}

std::vector<std::vector<float>> Function_Nodes_Cheb(std::vector<float> data)
{
		std::vector<float> nodes = Get_Nodes_Cheb(data[0], data[1], data[2]);
		std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
		return tabulate_polinom;
}

std::vector<std::vector<float>> Function_Nodes_Row_2(std::vector<float> data)
{
```

```
	std::vector<float> nodes = Get_Nodes_Row(data[0], data[1], data[3]);
	std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
	return tabulate_polinom;
}

std::vector<std::vector<float>> Function_Nodes_Cheb_2(std::vector<float> data)
{
	std::vector<float> nodes = Get_Nodes_Cheb(data[0], data[1], data[3]);
	std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
	return tabulate_polinom;
}

std::vector<std::vector<float>> Function_Lagrange_Row(std::vector<float> data)
{
	std::vector<float> nodes = Get_Nodes_Row(data[0], data[1], data[2]);
	std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
	std::vector<float> points_interpolate = Get_Nodes_Row(data[0], data[1],
data[3]);
	std::vector<std::vector<float>> interpolate_polinom =
Lagrange_interpolate(tabulate_polinom, points_interpolate);
	return interpolate_polinom;
}

std::vector<std::vector<float>> Function_Lagrange_Cheb(std::vector<float> data)
{
	std::vector<float> nodes = Get_Nodes_Cheb(data[0], data[1], data[2]);
	std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
	std::vector<float> points_interpolate = Get_Nodes_Cheb(data[0], data[1],
data[3]);
	std::vector<std::vector<float>> interpolate_polinom =
Lagrange_interpolate(tabulate_polinom, points_interpolate);
	return interpolate_polinom;
}

std::vector<std::vector<float>> Function_Newton_Row(std::vector<float> data)
{
	std::vector<float> nodes = Get_Nodes_Row(data[0], data[1], data[2]);
	std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
	std::vector<float> points_interpolate = Get_Nodes_Row(data[0], data[1],
data[3]);
	std::vector<std::vector<float>> interpolate_polinom =
Newton_interpolate(tabulate_polinom, points_interpolate);
	return interpolate_polinom;
}

std::vector<std::vector<float>> Function_Newton_Cheb(std::vector<float> data)
{
	std::vector<float> nodes = Get_Nodes_Cheb(data[0], data[1], data[2]);
	std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
	std::vector<float> points_interpolate = Get_Nodes_Cheb(data[0], data[1],
data[3]);
	std::vector<std::vector<float>> interpolate_polinom =
Newton_interpolate(tabulate_polinom, points_interpolate);
	return interpolate_polinom;
}

std::vector<std::vector<float>> Pogreshnost_Lagrange_Row(std::vector<float> data)
{
	std::vector<float> nodes = Get_Nodes_Row(data[0], data[1], data[2]);
	std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
	std::vector<float> points_interpolate = Get_Nodes_Row(data[0], data[1],
data[3]);
	std::vector<std::vector<float>> interpolate_polinom =
Lagrange_interpolate(tabulate_polinom, points_interpolate);
	nodes = Get_Nodes_Row(data[0], data[1], data[3]);
```

```
	tabulate_polinom = Tabulate(nodes);
	std::vector<std::vector<float>> pogreshnost =
Get_Pogreshnost(tabulate_polinom, interpolate_polinom);
	return pogreshnost;
}

std::vector<std::vector<float>> Pogreshnost_Lagrange_Cheb(std::vector<float> data)
{
	std::vector<float> nodes = Get_Nodes_Cheb(data[0], data[1],
data[2]);//получили разбиение отрезка
	std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
	std::vector<float> points_interpolate = Get_Nodes_Cheb(data[0], data[1],
data[3]);//получили разбиение для интерполяции
	std::vector<std::vector<float>> interpolate_polinom =
Lagrange_interpolate(tabulate_polinom, points_interpolate);
	nodes = Get_Nodes_Cheb(data[0], data[1], data[3]);
	tabulate_polinom = Tabulate(nodes);
	std::vector<std::vector<float>> pogreshnost =
Get_Pogreshnost(tabulate_polinom, interpolate_polinom);
	return pogreshnost;
}

std::vector<std::vector<float>> Pogreshnost_Newton_Row(std::vector<float> data)
{
	std::vector<float> nodes = Get_Nodes_Row(data[0], data[1],
data[2]);//получили разбиение отрезка
	std::vector<std::vector<float>> tabulate_polinom =
Tabulate(nodes);//протабулировали
	std::vector<float> points_interpolate = Get_Nodes_Row(data[0], data[1],
data[3]);//получили разбиение для интерполяции
	std::vector<std::vector<float>> interpolate_polinom =
Newton_interpolate(tabulate_polinom, points_interpolate);
	nodes = Get_Nodes_Row(data[0], data[1], data[3]);
	tabulate_polinom = Tabulate(nodes);
	std::vector<std::vector<float>> pogreshnost =
Get_Pogreshnost(tabulate_polinom, interpolate_polinom);
	return pogreshnost;
}

std::vector<std::vector<float>> Pogreshnost_Newton_Cheb(std::vector<float> data)
{
	std::vector<float> nodes = Get_Nodes_Cheb(data[0], data[1], data[2]);
	std::vector<std::vector<float>> tabulate_polinom = Tabulate(nodes);
	std::vector<float> points_interpolate = Get_Nodes_Cheb(data[0], data[1],
data[3]);//получили разбиение для интерполяции
	std::vector<std::vector<float>> interpolate_polinom =
Newton_interpolate(tabulate_polinom, points_interpolate);
	nodes = Get_Nodes_Cheb(data[0], data[1], data[3]);
	tabulate_polinom = Tabulate(nodes);
	std::vector<std::vector<float>> pogreshnost =
Get_Pogreshnost(tabulate_polinom, interpolate_polinom);
	return pogreshnost;
}
```