Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования "Уфимский государственный авиационный технический университет"

Кафедра Высокопроизводительных вычислительных технологий и систем

Дисциплина: Численные методы

Отчет по лабораторной работе № 1

Тема: «Приближение функций»

Группа ПМ-353	Фамилия И.О.	Подпись	Дата	Оценка
Студент	Шамаев И.Р.			
Принял	Гайнетдинова А. А.			

Уфа 2021

Требования к составу отчета:

Отчет к лабораторной работе оформляется в текстовом процессоре Microsoft Word или OpenOffice (LibreOffice) Write в соответствии с требованиями стандарта СТО УГАТУ 016-2007 и содержать

- титульный лист,
- описание цели работы и краткую теоретическую справку по использованным операторам и функциям языка программирования,
- описание выполнения задания:
  i. формулировка задания, как в методичке,
  ii. блок-схема каждой разработанной функции,
  iii. исходный текст разработанного приложения,
  iv. скриншоты примеров выполнения программы.
- выводы
- список использованной литературы,

Требования к оформлению отчета

- Текст отчета набирается шрифтом Times New Roman, размер шрифта 14pt или 12pt единый во всем документе. Исходный код должен быть набран любым моноширинным шрифтом (например, Courier New), размер символов можно уменьшить до 10 pt. Поля страницы: верхнее и нижнее – 2 см., левое – 2 см, правое – 1.5 см.
- Абзац должен начинаться с красной строки, за исключением тех случаев, когда абзац разорван каким-либо математическим выражением. Выравнивание внутри текстовых абзацев «по ширине», выравнивание в блоке с исходным кодом – «по левому краю».
- Все блок-схемы оформляются в соответствие с ГОСТ 19.701. Описание ГОСТ и пример блок-схемы приведены в разделе Справочник, методических указаний по выполнению лабораторных работ на сайте. Если блок-схемы составляются в сторонних приложениях, следует предусмотреть возможность их правки в учебном классе. В противном случае отчет с некорректно составленной блок-схемой и исправлениями вручную приниматься не будут.
- Описание выполнения каждого задания начинается с новой страницы. Следует использовать заголовок с текстом «**Индивидуальное задание №__**», набранный полужирным шрифтом с выравниванием по левому краю, без красной строки. Текст задания, текст в блок-схеме и описание программы следует набирать шрифтом Times New Roman с прямым начертанием (не курсив!!!).
- Текст на скриншотах должен быть читаемым. Скриншоты должны содержать только окно с результатами выполнения программы, а не весь рабочий стол. Размер шрифта на скриншотах должен соответствовать по размеру окружающему его тексту.

Ниже и выше красным выделены места, которые необходимо изменить.

Цель: изучить различные методы интерполирования и аппроксимации; получить навык проведения вычислительного эксперимента, направленного на решение задач интерполирования и аппроксимации функций.

Теоретический материал

Построение интерполяционных многочленов

Задача 1. Интерполяционный многочлен Лагранжа на равномерной сетке

Построение интерполяционного многочлена Лагранжа $L_n(x)$ для произвольной степени n по известным значениям функции $y_i=f(x_i)$, заданным на сетке узлов

$$a=x_0<x_1<\ldots<x_{n-1}<x_n=b,$$

производится по формуле

$$L_n(x)=\sum_{i=1}^{n} f(x_i)\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}.$$

Оценка погрешности приближения функции находится следующим образом:

1. Задается равномерная сетка узлов
$$x_{i+1}=x_i+h, x_0=a, h=\frac{b-a}{n}.$$
2. Для каждого n=1…15 строится интерполяционный многочлен Лагранжа $L_n(x)$ по значениям функции на заданной равномерной сетке узлов.
3. Оценка погрешности приближения функции находится как $\Delta_n=\left|f(x)-L_n(x)\right|, x\in[a,b].$
   Оценка проводится численно посредством вычисления модуля ошибки приближений $\left|f(x)-L_n(x)\right|$ в узлах мелкой равномерной сетки, состоящей из ~10^5 узлов, с выбором максимального значения в качестве искомой оценки.
4. Оптимальная степень n_0, при которой погрешность минимальна, определяется при помощи построения графика зависимости Δ_nот n.

Задача 2. Интерполяционный многочлен Лагранжа на неравномерной сетке

Построение сетки узлов, составленных из нулей многочлена Чебышева степени n_0, найденной при решении задачи 1 производится по формуле

$$x_i = \frac{b+a}{2} + \frac{b-a}{2} * \cos\left(\frac{\pi(2i+1)}{2n_0}\right), i = 0, 1, \ldots, n_0 - 1.$$

По построенной сетке узлов можно построить многочлен Лагранжа $L_{n0}(x)$ степени n_0 по аналогии с предыдущей задачей по формуле

$$L_n(x) = \sum_{i=1}^{n} f(x_i) \prod_{j \neq i} \frac{x - x_j}{x_i - x_j},$$

где x_i теперь узлы сетки, составленные из нулей многочлена Чебышева.

Оценка погрешности приближения функции Δ_{n0} производится методом, описанным в задаче 1.

Теоретическая минимальная оценка погрешности интерполяции многочленом Лагранжа представляется в виде

$$\left|f(x) - L_n(x)\right| \leq \frac{M_{n+1}}{(n+1)!}\left|\omega(x)\right|,$$

где $M_{n+1} = \max_{x \in [a,b]} ¿ f^{n+1}(x) \vee ¿, a w(x) = (x - x_0)..(x - x_n). ¿$

Задача 3. Интерполяционный многочлен Ньютона

Интерполяционный многочлен в форме Лагранжа не удобен для вычисления тем, что при увеличении числа узлов интерполяции приходится перестраивать весь многочлен заново.

Перепишем многочлен Лагранжа в другом виде:

$$P_n(x) = P_0(x) + \sum_{i=1}^{n} \left(P_i(x) - P_{i-1}(x)\right),$$

где $P_i(x)$ – многочлены Лагранжа степени $i \le n$.

Пусть $Q_i(x) = P_i(x) - P_{i-1}(x)$ (*). Этот многочлен имеет степень i и обращается в ноль при $x=x_0$, $x=x_1$,…, $x=x_{i-1}$. Поэтому он представим в виде:

$$Q_i(x) = A_i(x - x_0)\dots(x - x_{i-1}),$$

где A_i – коэффициент при x_i. Так как x_i входит в $P_{i-1}(x)$, то A_i совпадает с коэффициентом при x_i в многочлене $P_i(x)$. Таким образом, из определения $P_i(x)$ получаем:

$$A_i = \sum_{k=0}^{i} \frac{f(x_k)}{w_{k,i}},$$

где $w_{k,i} = (x_k - x_0)\dots(x_k - x_{k-1})(x_k - x_{k+1})\dots(x_k - x_i)$.

Перепишем формулу (*) в виде

$$P_n(x) = P_{n-1} + A_n(x - x_0)\dots(x - x_{n-1}).$$

Рекуррентно выражая $P_i(x)$ получаем окончательную формулу для многочлена:

$$P_n(x) = A_0 + A_1(x - x_0) + \dots + A_n(x - x_0)\dots(x - x_{n-1}).$$

Такое представление многочлена удобно для вычисления, потому что увеличение числа узлов на единицу требует добавления только одного слагаемого.

Оценка погрешности приближения функции Δ_{n0} производится методом, описанным в задаче 1.

Тригонометрическая интерполяция

Задача 4. Тригонометрический многочлен

Интерполяция функции $g(t), t \in [0, 2\pi]$, заданной своими значениями $g(t_i)(i=1,\ldots,2n+1)$ в узлах $t_i = \frac{2\pi(i-1)}{2n+1}$ равномерной сетки, тригонометрическим многочленом $F_n(x)$ степени n:

$$F_n(t) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos(kt) + b_k \sin(kt).$$

Коэффициенты a_k и b_k находятся следующим образом:

$$a_0 = \frac{1}{2n+1} \sum_{i=1}^{2n+1} y_i,$$

$$a_k = \frac{2}{2n+1} \sum_{i=1}^{2n+1} y_i \cos k\, x_i, b_k = \frac{2}{2n+1} \sum_{i=1}^{2n+1} y_i \sin k\, x_i npu\ k \in (1,\ldots,n).$$

Оценка погрешности приближения функции Δ_{n0} производится методом, описанным в задаче 1.

Наилучшее равномерное приближение

Задача 5. Многочлен наилучшего равномерного приближения

Рассмотрим задачу нахождения многочлена наилучшего приближения степени n в случае, когда

$$f(x)=P_{n+1}(x)=a_0+\ldots+a_{n+1}x^{n+1},\ a_{n+1}\neq 0.$$

Тогда $f^{(n+1)}(x)=a_{n+1}(n+1)!$ и оценки сверху и снизу для $E_n(f)$ совпадают:

$$E_n(f)=|a_{n+1}|(b-a)^{n+1}2^{-2n-1}.$$

Таким образом, многочленом наилучшего приближения оказывается интерполяционный многочлен $Q_n(x)$ с узлами интерполяции

$$x_k=\frac{a+b}{2}+\frac{b-a}{2}\cos\left(\frac{\pi(2k+1)}{2(n+1)}\right),k=1,\ldots,n+1.$$

Можно получить другое представление этого многочлена наилучшего приближения, записав его в виде

$$Q_n(x)=P_{n+1}(x)-a_{n+1}T_{n+1}\left(\frac{2x-(a+b)}{b-a}\right)\frac{(b-a)^{n+1}}{2^{2n+1}}.$$

Выражение в правой части является многочленом степени n, поскольку коэффициент при x^{n+1} равен нулю. Точки $x_i=\frac{a+b}{2}+\frac{b-a}{2}\cos\frac{\pi i}{n+1},i=0,\ldots,n,$ образуют чебышевский альтернанс.

Для определения наименьшей степени n многочлена Q_n, обеспечивающего приближение исходной функции f(x) с точностью, указанной в задании, пользоваться соотношением

$$¿\vee f(x)-Q_n(x)\vee\leq\Delta.$$

Многочлен $P_m(x)$, представляющий собой отрезок ряда Тейлора, приближает функцию f(x) с указанным в задании предельным уровнем погрешности Δ:

$$f(x)\approx P_m(x)=\sum_{k=0}^{m}b_kx^k,{}^{¿}f(x)-P_m(x)\vee\leq\Delta.$$

Интерполяция сплайнами

Задача 6. Построение интерполирующего кубического сплайна

Сплайн-функцией называют кусочно-полиноминальную функцию, определенную на отрезке [a,b] и имеющую на этом отрезке некоторое число непрерывных производных.

Пусть на отрезке [a,b] задана непрерывная функция f(x). Введем сетку

$$a=x_0<x_1<\ldots<x_{n-1}<x_n=b,$$

и обозначим $\varphi_i=\varphi(x_i), i=0,1,\ldots,N$.

Коэффициенты многочлена на каждом интервале определяют из условий в узлах. Очевидно, в узлах многочлен должен принимать табличные значения функции:

$$\varphi_i=y_i, \varphi'(x_i-0)=\varphi'(x_i+0), \varphi''(x_i-0)=\varphi''(x_i+0), i=1,2,\ldots,N-1$$

Кроме того, на границе при $x=x_0$ и $x=x_n$ ставятся условия

$$\varphi''(x_0)=0, \varphi''(x_n)=0.$$

Будем искать кубический многочлен в виде

$$\varphi(x)=a_i+b_i(x-x_{i-1})+c_i(x-x_{i-1})^2+d_i(x-x_{i-1})^3, x_{i-1}\le x\le x_i.$$

Из условия $\varphi_i=y_i$ имеем

$$\varphi(x_{i-1})=a_i=y_{i-1}, y_i=\varphi(x_i)=a_i+b_ih_i+c_ih_i^2+d_ih_i^2, h_i=x_i-x_{i-1}, i=1,2,\ldots,N-1$$

Число этих уравнений вдвое меньше числа неизвестных коэффициентов, поэтому для определенности задания нужны дополнительные условия. Для их получения вычислим первую и вторую производные многочлена:

$$\varphi'(x)=b_i+2c_i(x-x_{i-1})+3d_i(x-x_{i-1})^2$$

$$\varphi''(x)=2c_i+6d_i(x-x_{i-1})$$

И потребуем непрерывности этих производных во всех точках, включая узлы. Приравнивая во внутреннем узле x_i правые и левые пределы производных, получим:

$$b_{i+1}=b_i+2c_ih_i+3d_ih_i^2, i=1,2,\ldots,N-1$$

$$c_{i+1}=c_i+3d_ih_i, i=1,2,\ldots,N-1$$

Недостающие два условия обычно получают из естественного предположения о нулевой кривизне графика на концах:

$$\frac{1}{2}\varphi''(x_0)=c_1=0, \frac{1}{2}\varphi''(x_N)=c_N+3d_Nh_N=0,$$

что соответствует свободно опущенным концам линейки.

Уравнения образуют систему линейных уравнений для определения 4N неизвестных коэффициентов. Эту систему можно решить методом Гаусса. Найдем коэффициенты:

$$d_i=\frac{c_{i+1}-c_i}{3h_i},\ i=1,2,\ldots,N-1$$

$$d_N=-c_N/3h_N$$

$$b_i=\left(\frac{y_i-y_{i-1}}{h_i}\right)-\frac{h_i\left(c_{i+1}+2c_i\right)}{3},i=1,2,\ldots,N-1$$

$$b_N=\left(\frac{y_N-y_{N-1}}{h_N}\right)-\frac{2\left(h_N c_N\right)}{3}$$

Исключая величины b_i и b_{i+1}и d_i по соответствующим уравнениям, увеличивая во втором случае индекс на единицу. Остается система линейных уравнений для коэффициентов c_i, приводящаяся к виду:

$$c_1=0$$

$$h_{i-1}c_{i-1}+2\left(h_{i-1}+h_i\right)c_i+h_i c_{i+1}=3\left(\frac{y_i-y_{i-1}}{h_i}-\frac{y_{i-1}-y_{i-2}}{h_{i-1}}\right)$$

$$npu\ i=2,\ldots,N$$

$$c_{N+1}=0$$

Матрица этой системы трехдиагональная, т.е. ненулевыми в ней являются только элементы главной диагонали и двух соседних. Решается методом прогонки. После нахождения коэффициентов c_i, находим остальные коэффициенты.

Оценка погрешности приближения функции Δ_n производится методом, описанным в задаче 1.

Индивидуальное задание №1

Задание:

1) Написать вычислительную программу на языке программирования C++ для построения интерполяционного многочлена Лагранжа $L_n(x)$ произвольной степени n по известным значениям функции $y_i=f(x_i)$, заданным на сетке узлов
$$a=x_0<x_1<\ldots<x_{n-1}<x_n=b.$$
2) Для каждого n=1,...15 построить интерполяционный многочлен Лагранжа $L_n(x)$ по значениям функции на равномерной сетке узлов
$$x_{i+1}=x_i+h,\quad x_0=a,\quad h=(b-a)/n$$
и найти оценку погрешности приближения функции
$$\Delta_n=\sup|f(x)-L_n(x)|,\quad x\in[a,b].$$
Оценку Δ_n провести численно посредством вычисления модуля ошибки приближений $|f(x)-L_n(x)|$ в узлах мелкой равномерной сетки, состоящей из $\sim10^5$ узлов, с выбором максимального значения в качестве искомой оценки.
3) Построить график зависимости Δ_n от n определить оптимальную степень n_0, при которой погрешность минимальна.
4) Построить график ошибки приближения $f(x)-L_{n0}(x)$.

Описание программы**:**

double myfanc(double x_) – значение заданной функции в точке x

double Pol_Lagr(double x, vector <double> dx, vector <double> dy, int n) – интерполяция многочленом Лагранжа

double Sup(vector <double> y_r, vector <double> y_a) – абсолютная погрешность

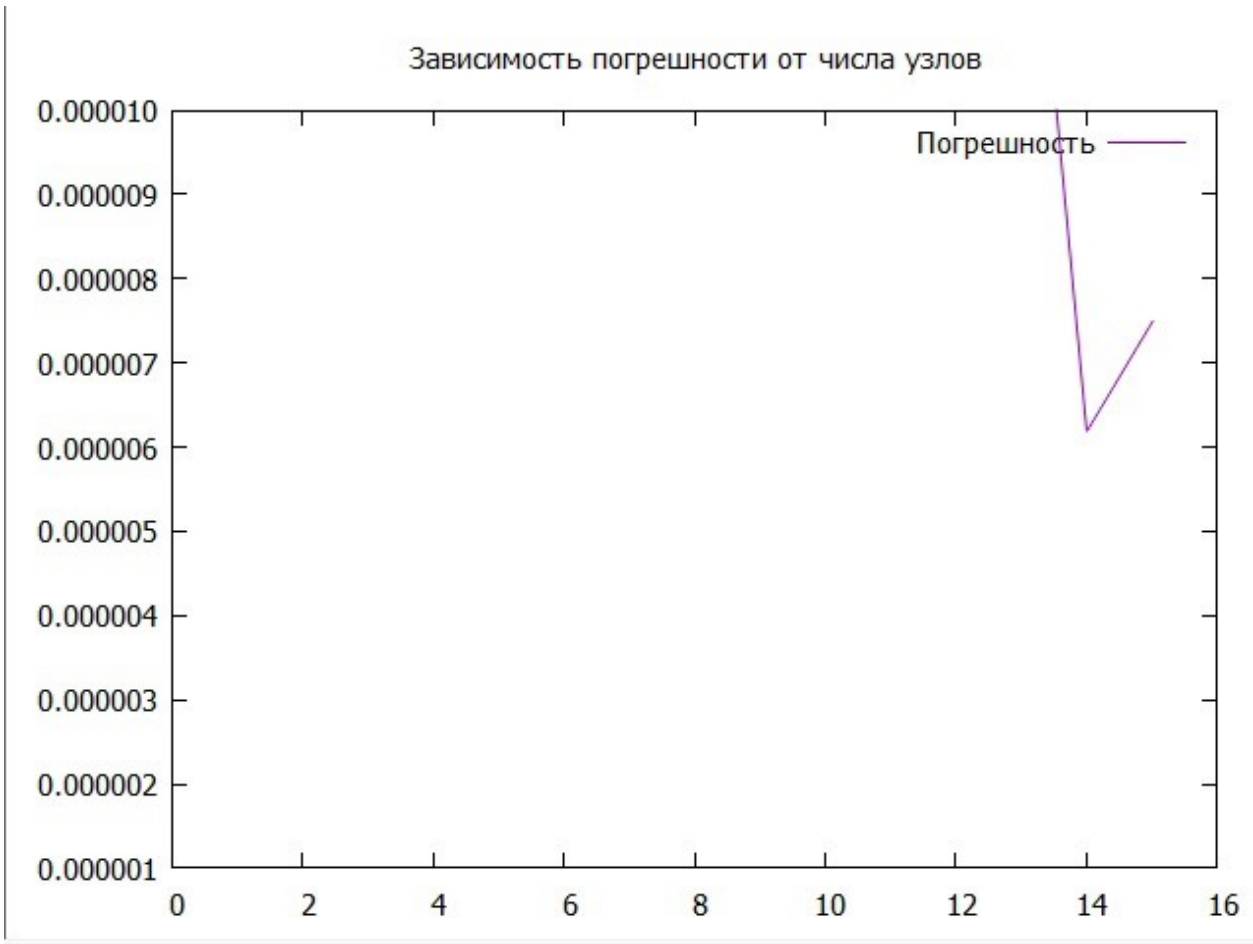
vector<double> Best_n(int r)– вывод оптимальной степени n0, r – рисовать графики или нет

Исходный код программы

Код программы представлен в приложении А.

Пример выполнения программы

```
Погрешность интерполяционного многочлена Лагранжа 6.18997e-06
n0=14
```


Зависимость погрешности от числа узлов
Погрешность
0.000010
0.000009
0.000008
0.000007
0.000006
0.000005
0.000004
0.000003
0.000002
0.000001
0
2
4
6
8
10
12
14
16

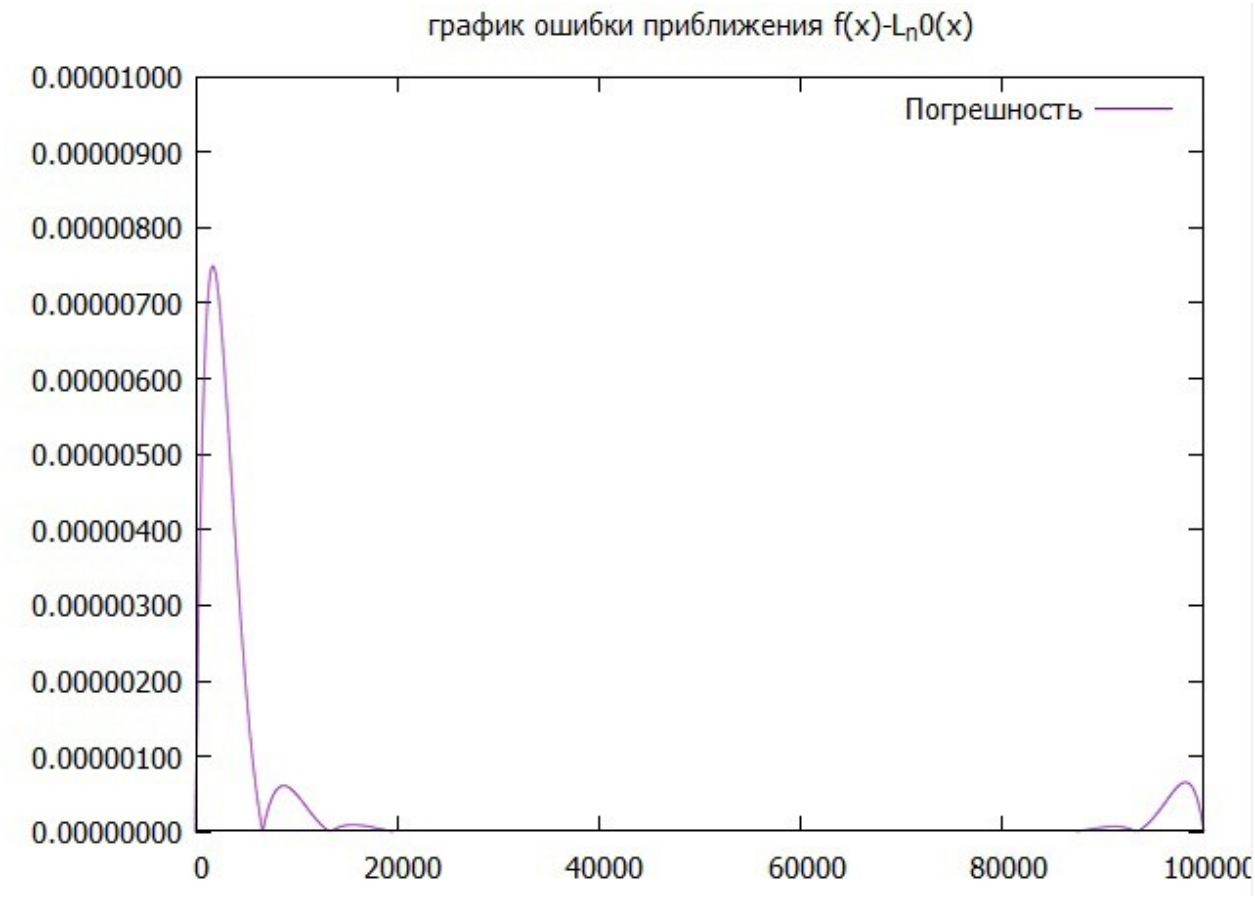


график ошибки приближения f(x)-L$_n$0(x)
Погрешность
0.00001000
0.00000900
0.00000800
0.00000700
0.00000600
0.00000500
0.00000400
0.00000300
0.00000200
0.00000100
0.00000000
0
20000
40000
60000
80000
100000

Индивидуальное задание №2

Задание:

1) Построить сетку узлов, составленных из нулей многочлена Чебышева степени n_0, найденной при решении предыдущей задачи:

$$x_i = \frac{b+a}{2} + \frac{b-a}{2}\cos\left(\frac{\pi(2i+1)}{2n_0}\right), i = 0, 1, \ldots, n_0 - 1.$$

   Найти численные значения заданной функции f(x) в этих узлах: $y_i = f(x_i)$.
2) С использованием написанной при решении Задачи 1 программы построить по этим данным многочлен Лагранжа $L_{n0}(x)$ степени n_0.
3) Найти оценку погрешности приближения функции Δ_{n0} и сравнить ее с известной теоретической минимальной оценкой погрешности интерполяции многочленом Лагранжа.
4) Выполнить сравнение двух многочленов Лагранжа $L_{n0}(x)$ на равномерной и неравномерной сетках, построенных в этой и предыдущей задачах.

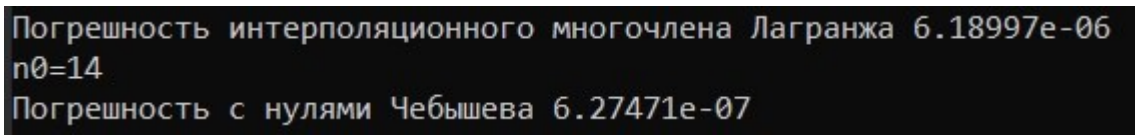
Описание программы**:**

void Cheb() - возвращает приближение функции ИМ Лагранжа на неравномерной сетке

Исходный код программы

Код программы представлен в приложении А.

Пример выполнения программы



Из результатов видно, что на неравномерной сетке погрешность меньше, примерно, в 10 раз.

Индивидуальное задание №3

Задание:

1) Написать вычислительную программу на языке программирования C++ для построения интерполяционного многочлена Ньютона порядка n_0 (найдено при решении Задачи 1) на равномерной сетке через вычисление разделенных разностей.

2)Выполнить сравнение построенного многочлена Ньютона с аналогичным многочленом Лагранжа, построенного при решении задачи 1.

Описание программы:

void Nton(vector<double> Lagr) – расчёт приближения функции ИМ Ньютона и вывод погрешность интерполяционного многочлена Ньютона

Исходный код программы

Код программы представлен в приложении А.

Пример выполнения:

```
Погрешность интерполяционного многочлена Лагранжа 6.18997e-06
n0=14
Погрешность интерполяционного многочлена Ньютона 6.18997e-06
```

Видно, что погрешность почти не отличается.

Индивидуальное задание №4

Задание:

1) Написать вычислительную программу на языке программирования C++, осуществляющую интерполяцию функции g(t), t∈[0,2π], заданной своими значениями $g(t_i)$ (i=1,…,2n+1) в узлах $t_i = \frac{2\pi(i-1)}{2n+1}$ равномерной сетки, тригонометрическим многочленом $F_n(x)$ степени n:

$$F_n(t) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \cos(kt) + b_k \sin(kt).$$

2) Построить линейную замену переменных x=αt+β, переводящую заданный отрезок [a,b] в отрезок [0,2π]. Выполнить эту замену переменных в аргументе функции f(x): f(αt+β)=g(t).
3) С использованием написанной программы провести вычислительный эксперимент по нахождению минимальной степени m тригонометрического многочлена, обеспечивающего приближение функции с указанным в задании предельным уровнем погрешности Δ:

$$\sup|g(t) - F_m(t)| \leq \Delta.$$

Оценку погрешности производить по способу, описанному в Задаче 1.
4) Построить график ошибки приближения функции многочленом.

Описание программы:

a-левая граница

b-правая граница

TrigonometricInterpol(double t, double* X, double* g, int n) – функция тригонометрической интерполяции

G_t(double t)-исходная функция

Step-шаг интерполяции

H-сетка

T_rep-замена

Исходный код программы

Код программы представлен в приложении В.

Пример выполнения:

```
1234567890      N = 10,  MaxDelta = 0.0343586
1234567890      N = 20,  MaxDelta = 0.0197167
1234567890      N = 30,  MaxDelta = 0.0148362
1234567890      N = 40,  MaxDelta = 0.01238
1234567890      N = 50,  MaxDelta = 0.0108977
1234567890      N = 60,  MaxDelta = 0.0099036
1234567890      N = 70,  MaxDelta = 0.00920138
1234567890      N = 80,  MaxDelta = 0.00867182
1234567890      N = 90,  MaxDelta = 0.00825455
1234567890      N = 100,  MaxDelta = 0.00789326
1234567890      N = 110,  MaxDelta = 0.00765108
1234567890      N = 120,  MaxDelta = 0.00739524
1234567890      N = 130,  MaxDelta = 0.00723599
1234567890      N = 140,  MaxDelta = 0.00703226
1234567890      N = 150,  MaxDelta = 0.00689656
1234567890      N = 160,  MaxDelta = 0.00680341
1234567890      N = 170,  MaxDelta = 0.00668072
1234567890      N = 180,  MaxDelta = 0.00652998
1234567890      N = 190,  MaxDelta = 0.00642342
1234567890      N = 200,  MaxDelta = 0.00639719
```

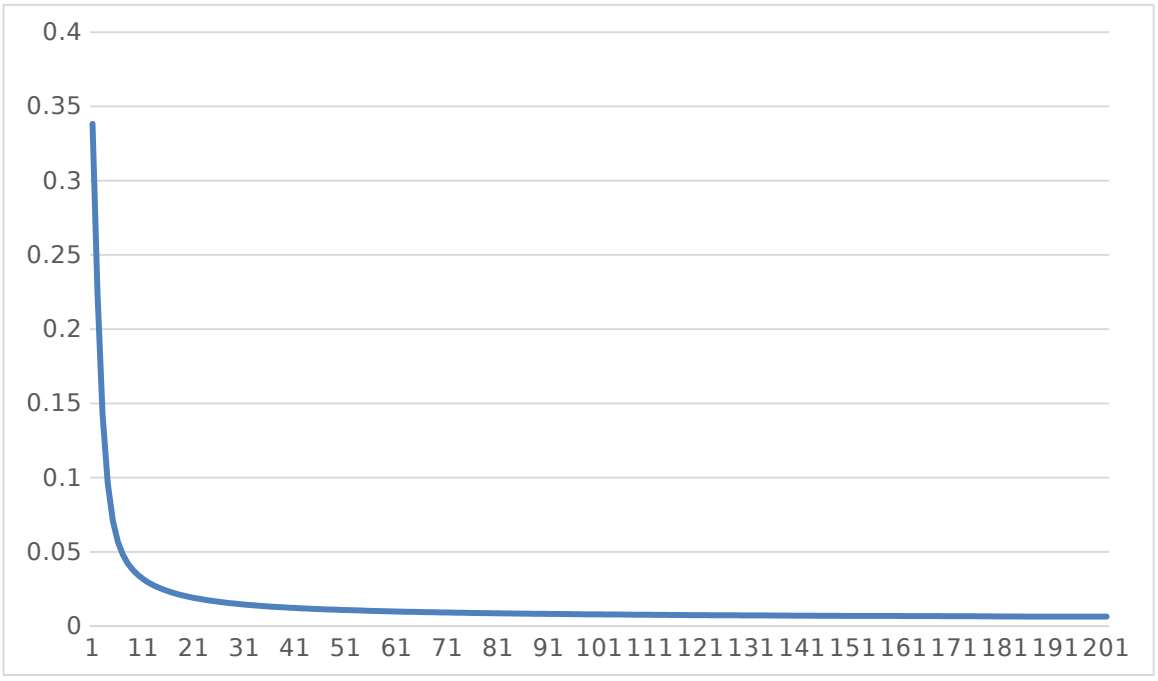


График 4. График зависимости Δ от n

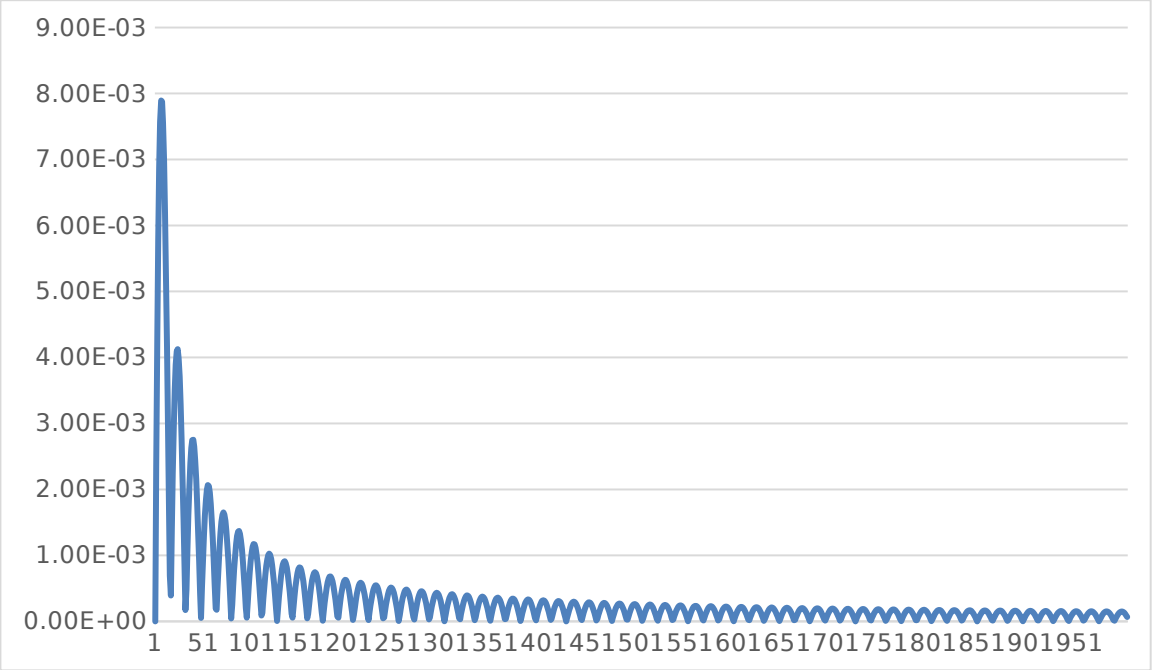


График 5. График ошибки приближения функции

Из графиков следует, что погрешность убывает очень медленно. Также на концах отрезка маленькая погрешность, поэтому мы не выходим за рамки допустимой погрешности 10^{-3}.

Индивидуальное задание №5

Задание:

1) Написать вычислительную программу на языке C++, позволяющую построить многочлен наилучшего равномерного приближения Q_n степени n для произвольного многочлена P_{n+1} степени n+1.
2) С использованием математического пакета (Maple или MATLAB) выполнить разложение заданной функции f(x) в ряд Тейлора в окрестности точки (a+b)/2 и определить степень n, при которой соответствующий многочлен $P_n(x)$, представляющий собой отрезок ряда Тейлора, приближает функцию f(x) с указанным в задании предельным уровнем погрешности Δ:

$$f(x) \approx P_n(x) = \sum_{k=0}^{n} b_k x^k, \; |f(x) - P_n(x)| \le \Delta.$$

3) С использованием написанной программы телескопическим методом построить многочлен Q_m наилучшего равномерного приближения наименьшей степени m, обеспечивающий приближении исходной функции f(x) с той же точностью:

$$¿ \left| f(x) - Q_m(x) \right| \le \Delta.$$

4) Построить график ошибки приближения функции многочленом Q_m.

Описание программы:

float T(float x,int n) - многочлен Чебышева степени n в точке x

vector<double> Task5_1(vector<double> koef) – приближение многочлена P_{n+1} многочленом Q_n

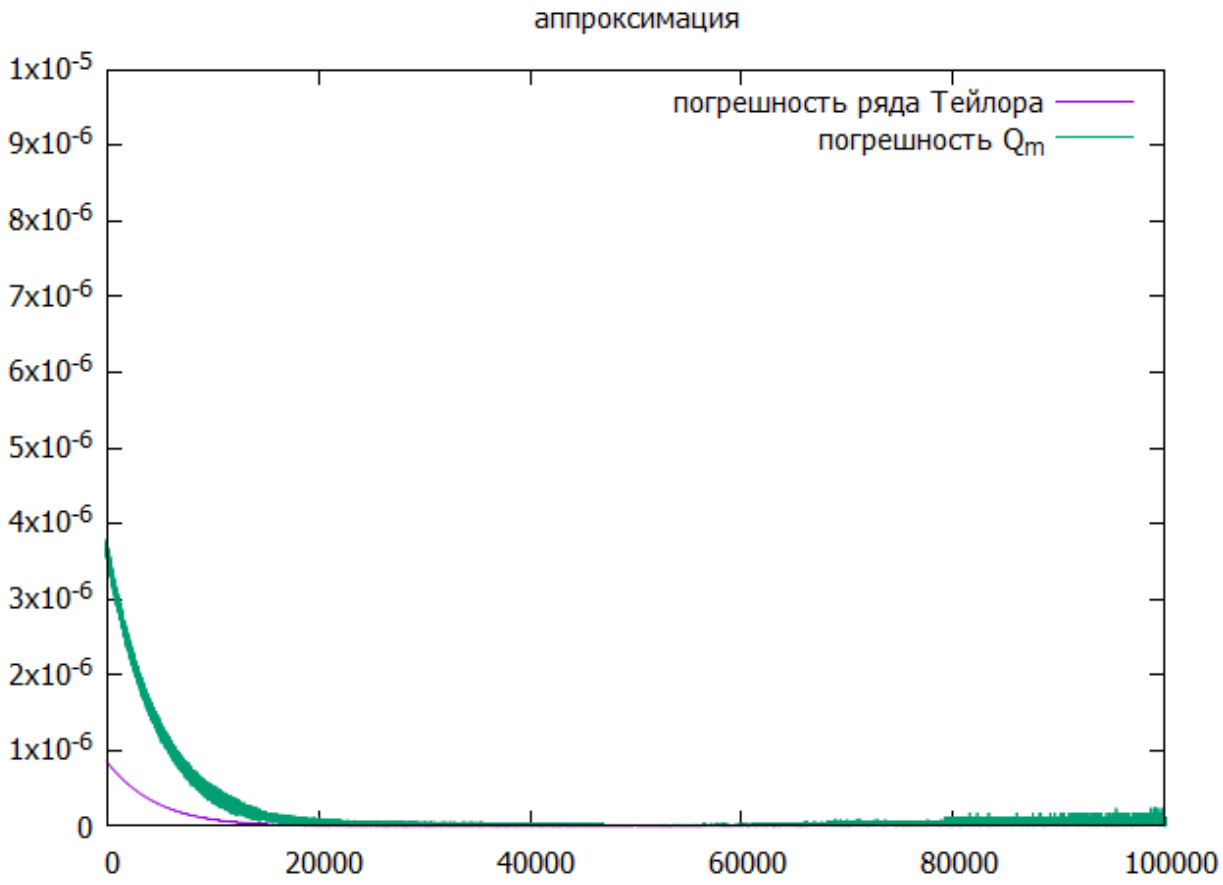
float NRP() – наилучшее равномерное приближение

Исходный код программы

Код программы представлен в приложении А.

Пример выполнения:

```
погрешность ряда Тейлора 8.58061e-07
Многочлен степени 14 с погрешностью 9.84467e-07
Многочлен степени 13 с погрешностью 3.79053e-06
Многочлен степени 12 с погрешностью 2.87322e-05
```


аппроксимация
погрешность ряда Тейлора
погрешность Qm
1x10-5
9x10-6
8x10-6
7x10-6
6x10-6
5x10-6
4x10-6
3x10-6
2x10-6
1x10-6
0
0
20000
40000
60000
80000
100000

Индивидуальное задание №6

Задание:

1) Написать вычислительную программу на языке программирования C++ для построения интерполирующего кубического сплайна по значениям функции, известным в узлах равномерной сетки.
2) С использованием написанной программы провести вычислительный эксперимент по определению минимального количества узлов равномерной сетки, обеспечивающих построение интерполирующего сплайна для заданной функции с указанным в задании предельным уровнем погрешности. Погрешность интерполяции оценивать способом, описанным в Задаче 1.
3) Построить график ошибки приближения заданной функции интерполирующим сплайном.

Описание программы:

a - левая граница

b – правая граница

f(double x) – исходная функция

Progonka(double* y, double* C, int n, double h) – прогонка

Pn(double xi, double* y, double* x, double* b, double* c, double* d, int n)- многочлен 3 степени

Исходный код программы

Код программы представлен в приложении В.

Пример выполнения:

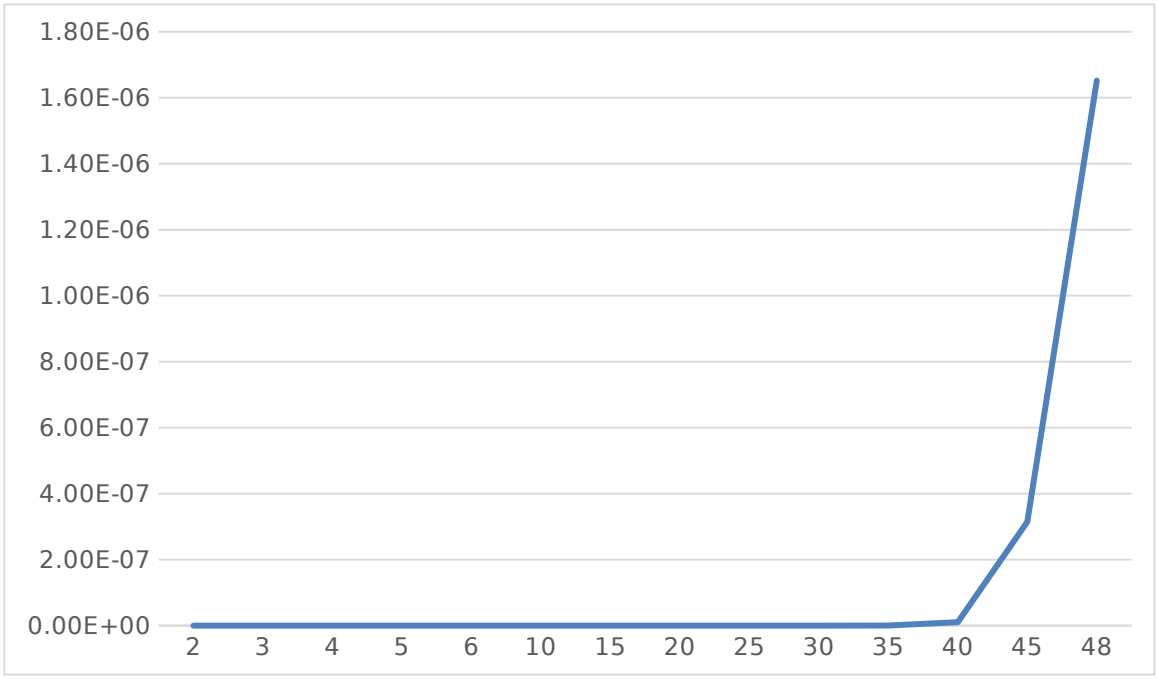


График 6. График ошибки приближения заданной функции

```
delta[19] = 1.99035e-13 - Podhodit
delta[20] = 2.53131e-14 - Podhodit
delta[21] = 1.07969e-13 - Podhodit
delta[22] = 1.26371e-13 - Podhodit
delta[23] = 5.7454e-14 - Podhodit
delta[24] = 8.95839e-13 - Podhodit
delta[25] = 1.07916e-12 - Podhodit
delta[26] = 2.99813e-12 - Podhodit
delta[27] = 7.71716e-13 - Podhodit
delta[28] = 2.50597e-12 - Podhodit
delta[29] = 4.19592e-12 - Podhodit
delta[30] = 7.75571e-12 - Podhodit
delta[31] = 3.9594e-10 - Podhodit
delta[32] = 1.01324e-11 - Podhodit
delta[33] = 2.58818e-10 - Podhodit
delta[34] = 1.99166e-10 - Podhodit
delta[35] = 2.7895e-10 - Podhodit
delta[36] = 8.15018e-10 - Podhodit
delta[37] = 1.32151e-09 - Podhodit
delta[38] = 8.3974e-09 - Podhodit
delta[39] = 4.48681e-09 - Podhodit
delta[40] = 1.06212e-08 - Podhodit
delta[41] = 3.36933e-08 - Podhodit
delta[42] = 1.73772e-08 - Podhodit
delta[43] = 3.51471e-07 - Podhodit
delta[44] = 7.94484e-08 - Podhodit
delta[45] = 3.14596e-07 - Podhodit
delta[46] = 3.09409e-07 - Podhodit
delta[47] = 3.27355e-07 - Podhodit
delta[48] = 1.65215e-06 - Podhodit
```

Вывод

В ходе лабораторной работы были получены навыки проведения вычислительного эксперимента, направленного на решение задач интерполирования и аппроксимации функций.

Список использованной литературы

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы.
2. Калиткин Н.Н. Численные методы.

Приложение А

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <math.h>
#include <vector>
#include <random>
#include <numeric>
#include <fstream>
#include "gnuplot_iostream.h"
const int N = 100000;
const double a = 0.0;
const double b = 1.5;
const int n0 = 15;
using namespace std;
double myfanc(double x_)
{
   return x_ * x_ * cos(M_PI * x_);
}
double Pol_Lagr(double x, vector <double> dx, vector <double> dy, int
n)//Лагранж
{
   double y = 0.0;
   for (int i = 0; i <= n; i++)
   {
      double temp = 1;
      for (int j = 0; j <= n; j++)
      {
         if (j != i)
         {
            temp *= (x - dx[j]) / (dx[i] - dx[j]);
         }
      }
      y += temp * dy[i];
   }
   return y;
}
double Sup(vector <double> y_r, vector <double> y_a)//погрешность
{
   double p = 0.0;
   for (int i = 0; i < N; i++)
   {
      double p_posible = abs(y_r[i] - y_a[i]);
      if (p_posible > p) p = p_posible;
   }
   return p;
}
vector<double> Best_n(int r)//1 задание, определяется оптимальная
степень n0, возращает приближение функции ИМ Лагранжа на равномерной
сетке, r - выводить график или нет
```

```
      {
         vector <vector<double>> p1_plot;//для f(x)-L_n0(x)
         vector <vector<double>> pogreshnosti;//для графика зависимости
погрешности от числа узлов
         Gnuplot gp("\"C:\\Program Files\\gnuplot\\bin\\gnuplot.exe\"");//график
ошибки приближения f(x)-L_n0(x)
         Gnuplot gp1("\"C:\\Program Files\\gnuplot\\bin\\gnuplot.exe\"");//график
зависимости погрешности от числа узлов
         vector <double> y_Lagr(N +1);
         int bestDeg = 1;
         double bestError = 100;
         for (int j = 1; j <= 15; j++)
         {
            double h = (b - a) / j;
            vector <double> x(j + 1);
            vector <double> y(j + 1);
            for (int i = 0; i <= j; i++)
            {
               x[i] = a + i * h;
               y[i] = myfanc(x[i]);
            }
            h = (b - a) / N;
            vector <double> x_1(N + 1);
            vector <double> y_1(N + 1);
            for (int k = 0; k <= N; k++)
            {
               x_1[k] = a + k * h;
               y_1[k] = myfanc(x_1[k]);
               y_Lagr[k] = Pol_Lagr(x_1[k], x, y, j);
               if (j == 15) p1_plot.push_back({ (double)k,abs(y_1[k] - y_Lagr[k]) });
            }
            pogreshnosti.push_back({ (double)j,Sup(y_1, y_Lagr) });
            if (Sup(y_1, y_Lagr) < bestError) {
               bestError = Sup(y_1, y_Lagr);
               bestDeg = j;
            }
         }
         cout << "Погрешность интерполяционного многочлена Лагранжа " <<
bestError << endl;//лучшая погрешность
        cout << "n0=" << bestDeg << endl;//оптимальное число узлов
        if (r == 1)
        {
           gp1 << "set title 'Зависимость погрешности от числа узлов'\n";
           gp1 << "set xrange[0:16]\nset yrange[0.0000000001:0.0000001]\n" <<
"plot"
              << gp1.file1d(pogreshnosti, "pogreshnosti.dat") << "with lines title
'Погрешность'," << endl;
           cin.get();
           gp << "set title 'график ошибки приближения f(x)-L_n0(x)'\n";
```

```
            gp << "set xrange[0:100000]\nset
yrange[0.000000000001:0.0000000001]\n" << "plot"
                << gp.file1d(p1_plot, "pogreshnosti.dat") << "with lines title
'Погрешность'," << endl;
            cin.get();
          }
          return y_Lagr;
       }
       void Cheb()// 2 задание,возвращает приближение функции ИМ Лагранжа
на неравномерной сетке
       {
          double h = (b - a) / N;
          vector<double> x_cheb(n0+1);
          vector<double> y_cheb(n0+1);
          vector<double> x(N+1);
          vector<double> y(N+1);
          vector<double> y_Lagr(N+1);
          for (int i = 0; i < n0; i++)
          {
             x_cheb[i] = (b + a) / 2 + (b - a) / 2 * cos(M_PI * (2 * i + 1) / (2 * n0));
             y_cheb[i] = myfanc(x_cheb[i]);
          }
          for (int i = 0; i < N; i++)
          {
             x[i] = a + i * h;
             y[i] = myfanc(x[i]);
             y_Lagr[i] = Pol_Lagr(x[i], x_cheb, y_cheb, n0);
          }
          cout << "Погрешность с нулями Чебышева " << Sup(y, y_Lagr) << endl;
       }
       double Pr(double t, vector<double> Xx,int n)
       {
          double PR = 1;
          for (int i = 0; i < n; i++)
          {
             PR *= (t - Xx[i]);
          }
          return PR;
       }
       void Nton(vector<double> Lagr)//3 задание, ИМ Ньютона
       {
          vector<double> x(n0+1);
          vector<double> y(n0+1);
          vector<double> x_N(N + 1);
          vector<double> y_N(N + 1);
          vector<double> y_chN(N + 1);
          vector<double> vh;//вспомогательный вектор, чтобы посчитать р.р.
          float h = (b - a) / n0;
          for (int i = 0; i <= n0; i++)
```

```
        {
            x[i] = a + i * h;
            y[i] = myfanc(x[i]);
        }
        vector<double> koef;//вектор коэффициентов
        for (int i = 0; i < n0; i++)//р.р. 1-го порядка
        {
            vh.push_back((y[i] - y[i+1])/(x[i]-x[i+1]));
        }
        koef.push_back(vh[0]);
        for (int i = 0; i < n0-1; i++)//подсчёт разделённых разностей порядка
>=2
        {
            for (int j = 0; j < n0-i-1; j++)
            {
                vh[j] = (vh[j] - vh[j+1])/(x[j]-x[j+i+2]);
            }
            koef.push_back(vh[0]);
        }
        h = (b - a) / N;
        for (int i = 0; i <= N; i++)//интерполяция Ньютоном
        {
            x_N[i] = a + i * h;
            y_N[i] = myfanc(x_N[i]);
            y_chN[i] = y[0];
            for (int j = 1; j <= n0; j++)
            {
                y_chN[i] += koef[j-1] * Pr(x_N[i], x, j);
            }
        }
        cout << "Погрешность интерполяционного многочлена Ньютона
"<<Sup(y_N, y_chN)<<endl;

    }
    double T(double x,int n)//многочлен Чебышева
    {
        return (pow(x + sqrt(abs(x * x - 1)), n) + pow(x - sqrt(abs(x * x - 1)), n)) / 2.;
    }
    vector<double> Task5_1(vector<double> koef)
    {
        int step=koef.size()-1;//n+1=15
        vector<double> Q_n(N+1);
        vector<double> P(N+1);
        double sum = 0.0;
        double h = (b - a) / N;
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < step+1; j++) {
                sum += koef[j] * pow(a + i * h-0.75, j);
```

```
            }
            Q_n[i] = sum - koef[step] * T((2. * (a + i * h) - (a + b) / (b - a)), step) *
pow(b - a, step) / pow(2., 2 * (step-1) + 1);
            P[i] = sum;
            sum = 0.0;
         }
         return Q_n;
      }
      double NRP()//5 задание
      {
         Gnuplot gp("\"C:\\Program Files\\gnuplot\\bin\\gnuplot.exe\"");
         vector<double> Teilor(N + 1);
         vector<double> nrp(N + 1);
         vector<double> real_y(N + 1);
         vector<double> koef = { (-9. / 32.) * sqrt(2.),(-9. / 32. * sqrt(2.) * M_PI - 3. /
4. * sqrt(2.)) ,(9. / 64. * sqrt(2.) * M_PI * M_PI - 0.5 * sqrt(2.) - 3. / 4. * sqrt(2.) *
M_PI),
         (3. / 8. * sqrt(2.) * M_PI * M_PI + 3. / 64. * sqrt(2.) * pow(M_PI, 3) - 0.5 *
sqrt(2.) * M_PI) ,(1. / 4. * sqrt(2.) * M_PI * M_PI + 1. / 8. * sqrt(2.) * pow(M_PI, 3) - 3. /
256. * sqrt(2.) * pow(M_PI, 4)) ,
         (1. / 12. * sqrt(2.) * pow(M_PI, 3) - 3. / 1280. * sqrt(2.) * pow(M_PI, 5) - 1. /
32. * sqrt(2.) * pow(M_PI, 4)) ,(1. / 2560. * sqrt(2) * pow(M_PI, 6) - 1. / 48. * sqrt(2.) *
pow(M_PI, 4) - 1. / 160. * sqrt(2) * pow(M_PI, 5)) ,
         ((1. / 17920.) * sqrt(2.) * pow(M_PI, 7) + (1. / 960.) * sqrt(2.) * pow(M_PI, 6) -
(1. / 240.) * sqrt(2.) * pow(M_PI, 5)) ,(1. / 1440. * sqrt(2.) * pow(M_PI, 6) + 1. / 6720.
* sqrt(2.) * pow(M_PI, 7) - 1. / 143360. * sqrt(2.) * pow(M_PI, 8)) ,
         (-1. / 53760. * sqrt(2.) * pow(M_PI, 8) + 1. / 10080. * sqrt(2.) * pow(M_PI, 7) -
1. / 1290240. * sqrt(2.) * pow(M_PI, 9)) ,
         (1. / 12902400. * sqrt(2.) * pow(M_PI, 10) - 1. / 80640. * sqrt(2.) * pow(M_PI,
8) - 1. / 483840. * sqrt(2.) * pow(M_PI, 9)) ,
         (1. / 141926400. * sqrt(2.) * pow(M_PI, 11) + 1. / 4838400. * sqrt(2.) *
pow(M_PI, 10) - 1. / 725760. * sqrt(2.) * pow(M_PI, 9)) ,
         ((1. / 7257600.) * sqrt(2.) * pow(M_PI, 10) - (1. / 1703116800.) * sqrt(2.) *
pow(M_PI, 12) + (1. / 53222400.) * sqrt(2.) * pow(M_PI, 11)),
         ((1. / 79833600.) * sqrt(2.) * pow(M_PI, 11) - (1. / 638668800.) * sqrt(2.) *
pow(M_PI, 12) - (1. / 22140518400.) * sqrt(2.) * pow(M_PI, 13)) ,
         (-(1. / 958003200.) * sqrt(2.) * pow(M_PI, 12) + (1. / 309967257600.) *
sqrt(2.) * pow(M_PI, 14) - (1. / 8302694400.) * sqrt(2.) * pow(M_PI, 13)) ,
         ((1. / 4649508864000.) * sqrt(2.) * pow(M_PI, 15) - (1. / 12454041600.) *
sqrt(2.) * pow(M_PI, 13) + (1. / 116237721600.) * sqrt(2.) * pow(M_PI, 14)) };
         float h = (b - a) / N;
         for (int i = 0; i < N; i++)
         {
            Teilor[i] = Teilor[i] = (-9. / 32.) * sqrt(2.) + (-9. / 32. * sqrt(2.) * M_PI - 3. /
4. * sqrt(2.)) * ((a + i * h) - 0.75) +
               (9. / 64. * sqrt(2.) * M_PI * M_PI - 0.5 * sqrt(2.) - 3. / 4. * sqrt(2.) * M_PI)
* pow((a + i * h) - 0.75, 2) +
               (3. / 8. * sqrt(2.) * M_PI * M_PI + 3. / 64. * sqrt(2.) * pow(M_PI, 3) - 0.5 *
sqrt(2.) * M_PI) * pow((a + i * h) - 0.75, 3) +
```

```
                (1. / 4. * sqrt(2.) * M_PI * M_PI + 1. / 8. * sqrt(2.) * pow(M_PI, 3) - 3. /
256. * sqrt(2.) * pow(M_PI, 4)) * pow((a + i * h) - 0.75, 4) +
                (1. / 12. * sqrt(2.) * pow(M_PI, 3) - 3. / 1280. * sqrt(2.) * pow(M_PI, 5) -
1. / 32. * sqrt(2.) * pow(M_PI, 4)) * pow((a + i * h) - 0.75, 5) +
                (1. / 2560. * sqrt(2) * pow(M_PI, 6) - 1. / 48. * sqrt(2.) * pow(M_PI, 4) - 1.
/ 160. * sqrt(2) * pow(M_PI, 5)) * pow((a + i * h) - 0.75, 6) +
                ((1. / 17920.) * sqrt(2.) * pow(M_PI, 7) + (1. / 960.) * sqrt(2.) *
pow(M_PI, 6) - (1. / 240.) * sqrt(2.) * pow(M_PI, 5)) * pow((a + i * h) - 0.75, 7) +
                (1. / 1440. * sqrt(2.) * pow(M_PI, 6) + 1. / 6720. * sqrt(2.) * pow(M_PI,
7) - 1. / 143360. * sqrt(2.) * pow(M_PI, 8)) * pow((a + i * h) - 0.75, 8) +
                (-1. / 53760. * sqrt(2.) * pow(M_PI, 8) + 1. / 10080. * sqrt(2.) *
pow(M_PI, 7) - 1. / 1290240. * sqrt(2.) * pow(M_PI, 9)) * pow((a + i * h) - 0.75, 9)+
                (1. / 12902400. * sqrt(2.) * pow(M_PI, 10) - 1. / 80640. * sqrt(2.) *
pow(M_PI, 8) - 1. / 483840. * sqrt(2.) * pow(M_PI, 9)) * pow((a + i * h) - 0.75, 10) +
                (1. / 141926400. * sqrt(2.) * pow(M_PI, 11) + 1. / 4838400. * sqrt(2.) *
pow(M_PI, 10) - 1. / 725760. * sqrt(2.) * pow(M_PI, 9)) * pow((a + i * h) - 0.75, 11) +
                ((1. / 7257600.) * sqrt(2.) * pow(M_PI, 10) - (1. / 1703116800.) * sqrt(2.)
* pow(M_PI, 12) + (1. / 53222400.) * sqrt(2.) * pow(M_PI, 11)) * pow((a + i * h) -
0.75, 12) +
                ((1. / 79833600.) * sqrt(2.) * pow(M_PI, 11) - (1. / 638668800.) * sqrt(2.)
* pow(M_PI, 12) - (1. / 22140518400.) * sqrt(2.) * pow(M_PI, 13)) * pow((a + i * h) -
0.75, 13) +
                (-(1. / 958003200.) * sqrt(2.) * pow(M_PI, 12) + (1. / 309967257600.) *
sqrt(2.) * pow(M_PI, 14) - (1. / 8302694400.) * sqrt(2.) * pow(M_PI, 13)) * pow((a + i
* h) - 0.75, 14) +
                ((1. / 4649508864000.) * sqrt(2.) * pow(M_PI, 15) - (1. / 12454041600.)
* sqrt(2.) * pow(M_PI, 13) + (1. / 116237721600.) * sqrt(2.) * pow(M_PI, 14)) *
pow((a + i * h) - 0.75, 15);;
            real_y[i] = myfanc(a + i * h);
        }
        float k = Sup(real_y, Teilor);
        cout <<"погрешность ряда Тейлора "<< k << endl;
        vector<double> ko = koef;
        while(k < 0.00001)
        {
            k = Sup(real_y, Task5_1(ko));
            cout << "Многочлен степени " << ko.size() - 2 << " с погрешностью "
<< k<<endl;
            ko.resize(ko.size() - 1);
        }
        koef.resize(ko.size() + 2);
        vector<vector<double>> error_Q_m;
        vector<double> kk = Task5_1(koef);
        for (int i = 0; i < N; i++)
        {
            error_Q_m.push_back({ (double)i, abs(real_y[i] - kk[i]) });
        }
        gp << "set title ' график ошибки приближения функции многочленом
Qm'\n";
```

```
        gp << "set xrange[0:100000]\nset yrange[0.00000000001:0.00000001]\n"
<< "plot"
            << gp.file1d(error_Q_m, "P_Task5.dat") << "with lines title 'погрешность
Q_m'," << endl;
        cin.get();
        system("pause");
    }
    int main()
    {
        system("chcp 1251");
        system("cls");
        vector<double> ravn=Best_n(0);//1 задание
        //Cheb();//2 задание
        Nton(ravn);//3 задание
        NRP();//задание 5
    }
```

Приложение В

Задание 4

```
#define _USE_MATH_DEFINES

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string>
#include <math.h>
#include <fstream>

using namespace std;

double TrigonometricInterpol(double t, double* X, double* g, int n)
{
	double A0 = 0;
	double Ak = 0;
	double Bk = 0;

	double sum = 0;
	double result = 0;

	for (int i = 0; i < 2 * n + 1; i++){

		sum += g[i];
	}

	A0 = sum / (2 * n + 1);
	result += A0;

	for (int k = 1; k <= n; k++){
			sum = 0;

			for (int i = 0; i < 2 * n + 1; i++) {
				sum += g[i] * cos(k * X[i]);
			}
			Ak = 2 * sum / (2 * n + 1);
```

```
                    sum = 0;

                    for (int i = 0; i < 2 * n + 1; i++) {
                          sum += g[i] * sin(k * X[i]);
                    }

                    Bk = 2 * sum / (2 * n + 1);
                    result += Ak * cos(k * t) + Bk * sin(k * t);

      }

            return result;
}


double G_t(double t)
{
      return atan(t)/(1+t*t);//функция
}

int main() {

      double a = 0;
      double b = 2;
      int c = 1000;

      ofstream fout;
      ofstream fout2;

      fout.open("delta.txt");//погрешность дельта
      fout2.open("graph.txt");//расчет интерполяции

      for (int n = 1; n < 300; n++) {

            double h = (b - a) / n;//сетка

            double* t = new double[2 * n + 1];
            double* g = new double[2 * n + 1];

            for (int i = 0; i < 2 * n + 1; i++) {
                  t[i] = 2 * M_PI * (i) / (2 * n + 1);//узлы

                  g[i] = G_t(t[i]);//значения узлов
            }

            double* t_rep = new double[c];//замена

            double step = (b - a) / c;

            for (int i = 0; i < c - 1; i++){

                  t_rep[i] = a + step * i;//
            }
                  double MaxDelta = 0;

                  for (int i = 0; i < c - 1; i++) {

                        double delta = abs(TrigonometricInterpol(t_rep[i], t, g,
n) - G_t(t_rep[i]));//дельта

                        if (n == 100) {

                              fout2 << delta << endl;
                        }
```

```
                        if (delta > MaxDelta) {

                              MaxDelta = delta;
                        }

                  }

                  cout << n % 10;

                  if (MaxDelta < 0.001) {
                        cout << endl << "END\n" << "N = " << n << ", " << "
MaxDelta = " << MaxDelta << " < 10e3" << endl;
                        break;
                  }

                  if (n % 10 == 0)

                        cout << "\tN = " << n << ", " << " MaxDelta = " <<
MaxDelta << endl;

                  fout << MaxDelta << endl;

            }

      return 0;

     }
```

Задание 6

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <math.h>
#define pi 3.14159
#define _USE_MATH_DEFINES

const double a = 0, b = 2;

using namespace std;

double f(double x)
{
      return atan(x) / (1 + x * x);
}
void Progonka(double* y, double* C, int n, double h)//4N неизвестных
{
      double* alpha = new double[n + 1];///
      double* beta = new double[n + 1];///
      double* F = new double[n + 1];///
      double z = 0;///
      double c = 4 * h;
      double b = h;
      double a = h;
      int N = n - 1;///

      for (int i = 1; i <= N; i++)
      {
            F[i] = (3 / h) * (y[i + 1] - 2 * y[i] + y[i - 1]);//уравнение для C
(28)
      }

      alpha[1] = -c / b;
      beta[1] = F[1] / b;

      for (int i = 2; i <= N - 1; i++)
```

```
	{
		z = b + a * alpha[i - 1];
		alpha[i] = -c / z;
		beta[i] = (F[i] - a * beta[i - 1]) / z;
	}

	beta[N] = (F[N] - a * beta[N - 1]) / (b + a * alpha[N - 1]);
	C[n] = 0;
	C[N] = beta[N];

	for (int i = N - 1; i >= 1; i--)
	{
		C[i] = alpha[i] * C[i + 1] + beta[i];
	}

	C[0] = 0;
}

void coefPn(double* y, double* B, double* C, double* D, int n, double h)//(25-27)
{
	for (int i = 1; i <= n - 1; i++)
	{
		D[i] = (C[i] - C[i - 1]) / (3 * h);
	}

	D[n] = -C[n - 1] / (3 * h);

	for (int i = 1; i <= n - 1; i++)
	{
		B[i] = ((y[i] - y[i - 1]) / h) - (h / 3) * (C[i] + 2 * C[i - 1]);
	}

	B[n] = ((y[n] - y[n - 1]) / h) - (2 * h / 3) * C[n - 1];
}
double Pn(double xi, double* y, double* x, double* b, double* c, double* d, int
n)//Многочлен 3 степени
{
	for (int i = 0; i <= n; i++)
	{
		if (xi >= x[i] && xi <= x[i + 1])
			return (y[i] + b[i + 1] * (xi - x[i]) + c[i] * pow((xi - x[i]),
2) + d[i + 1] *
				pow((xi - x[i]), 3));
	}
}
int main()
{
	int n;
	for (int n = 1; n < 49; n++)
	{
		double h = (b - a) / n;////////////
		double* x = new double[n + 1];
		double* y = new double[n + 1];

		for (int i = 0; i <= n; i++)
		{
			x[i] = i * h;
			y[i] = f(x[i]);
		}

		double* B = new double[n + 1];
		double* C = new double[n + 1];
		double* D = new double[n + 1];
		Progonka(y, C, n, h);
		coefPn(y, B, C, D, n, h);
```

```
		double delta = 0.0;
		double maxDelta = 0.0;

		for (double i = 0; i <= b; i += h)
		{
			delta = abs(f(i) - Pn(i, y, x, B, C, D, n));
			if (maxDelta <= delta)
				maxDelta = delta;
		}

		cout << "delta[" << n << "] = " << maxDelta;
		if (maxDelta <= 0.001)
		{
			cout << " <- optimal" << endl;
		}
	}
	return 0;
}
```