Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования   
«Алтайский государственный технический университет

им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой

Преподаватель

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Отчет

по лабораторной работе № 6

**«Интерполирование многочленами»**

по дисциплине «Вычислительная математика»

Студент гр. ПИ-81

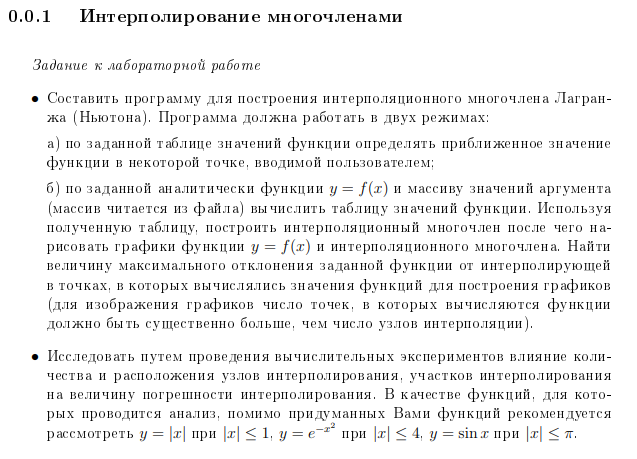
Стойко Н. И.

Преподаватель доцент, к. ф-м. н.

Кантор С.А.

Барнаул 2021

# Задания



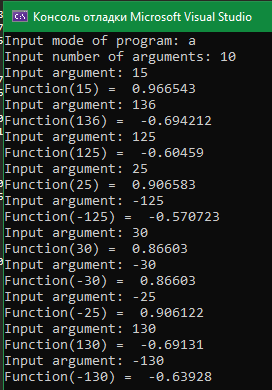
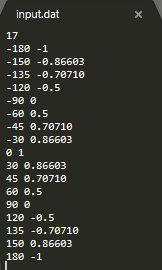
# Описание метода решения

По заданным значениям функции в точках возможно построить многочлен Ньютона для приближения значений функции:

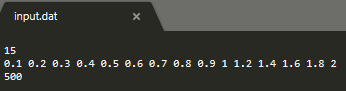
Где разделенные разности вычисляются следующим образом:  
Их удобно считать с помощью треугольной матрицы.

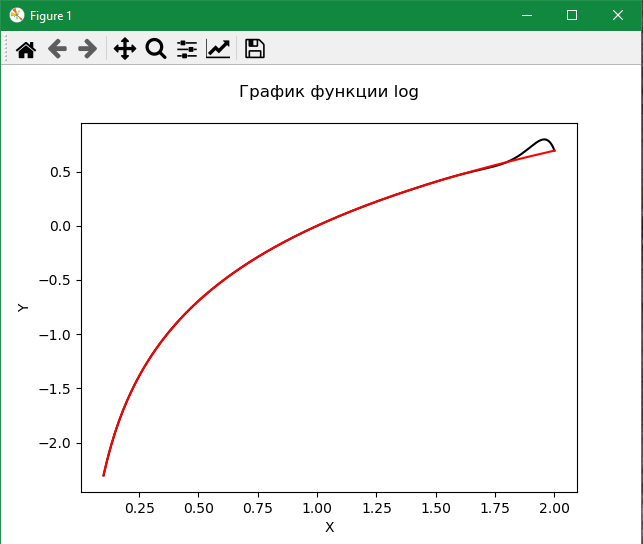
При этом можно получить, что разница функции и многочлена будет ограничена, при условии существования n+1 производной функции и ограниченности производной на отрезке.

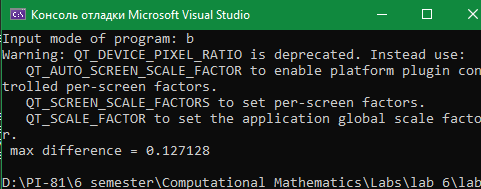
# Тестирование

Тестирование пункта а) для cos(x)  


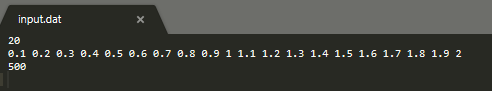
## Пункт b) для функции ln(x)

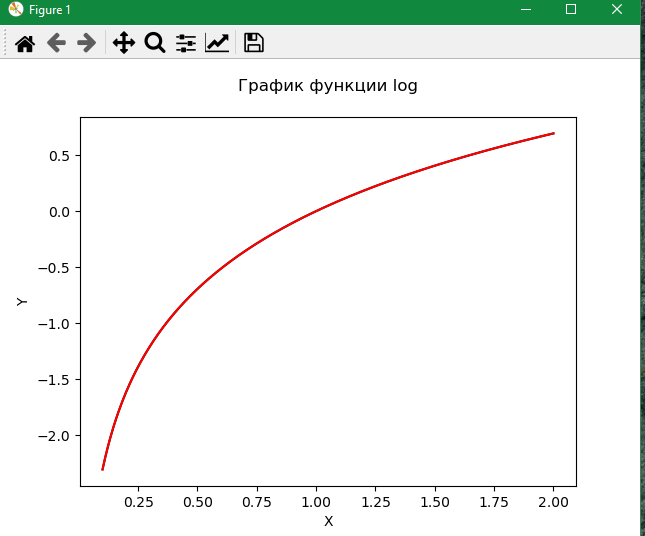


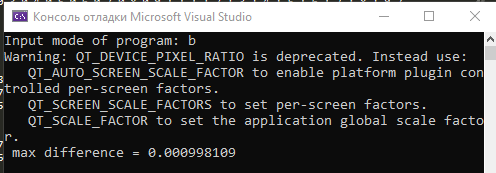




Увеличим количество узлов интерполяции:

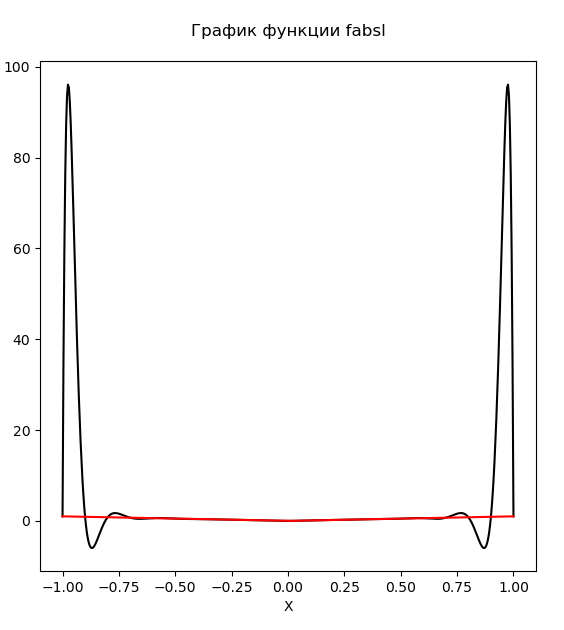
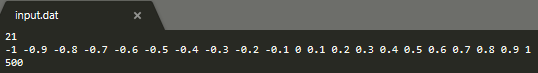
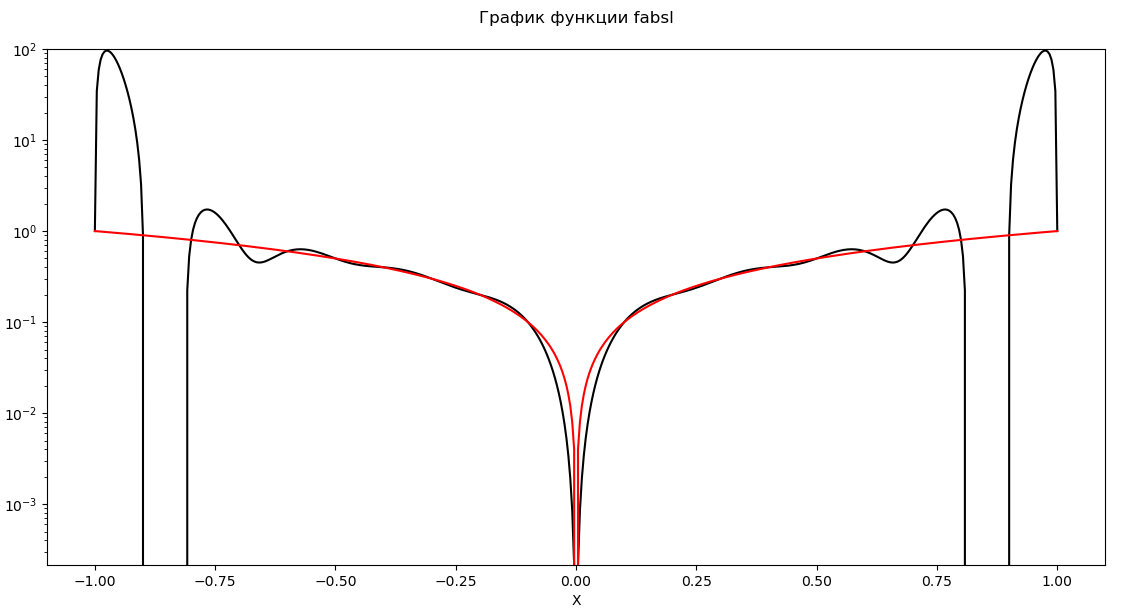
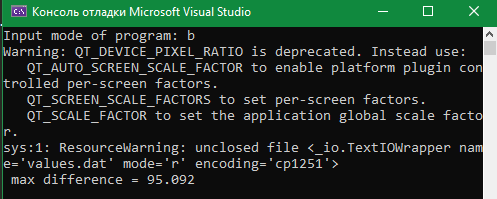


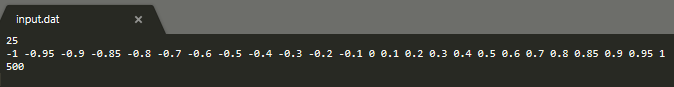


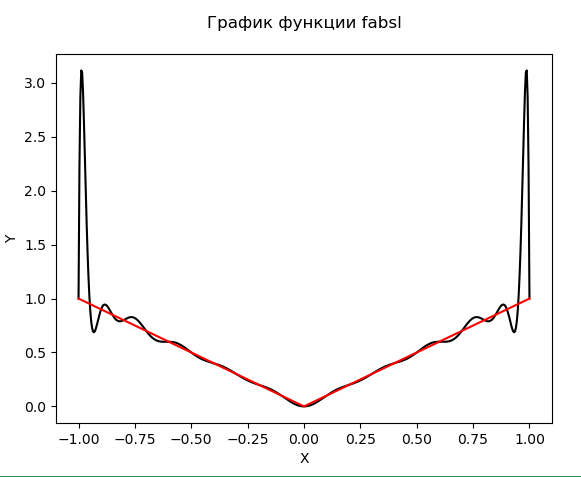


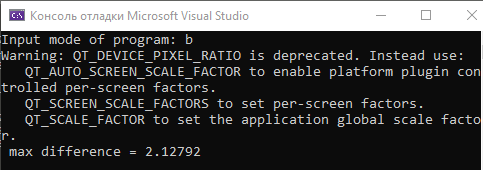
За счет уплотнения улов интерполяции в конце отрезка, удалось уменьшить величину максимального отклонения интерполяционного многочлена от исходной функции.

## График y = |x|

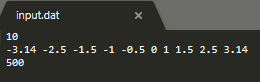
Уплотним количество узлов интерполяции ближе к концу отрезка. 

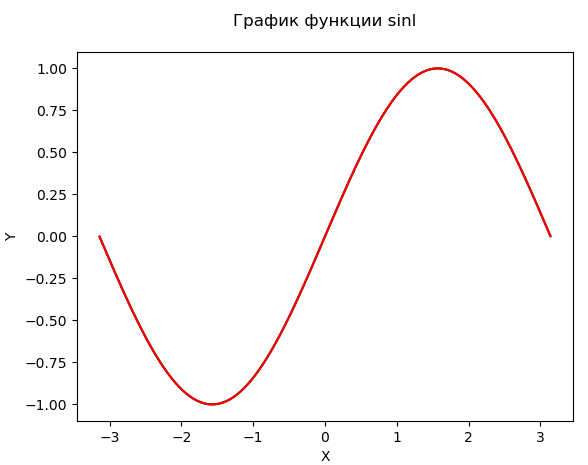


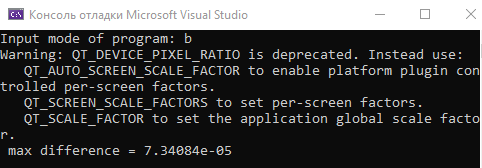


Функция модуля не является непрерывно дифференцируемой, поэтому не подходит для интерполирования. Хотя погрешность интерполирование может существенно снижаться с ростом узлов интерполяции, точность интерполирования остается не высокой.

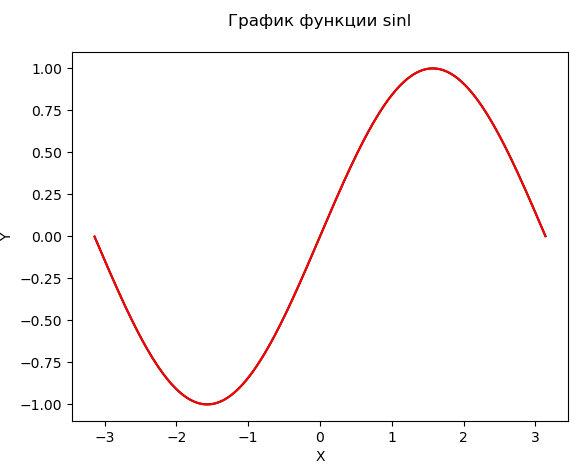
## Sin(x), |x| <= 3.14

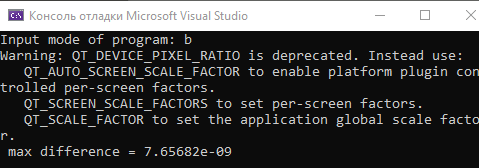






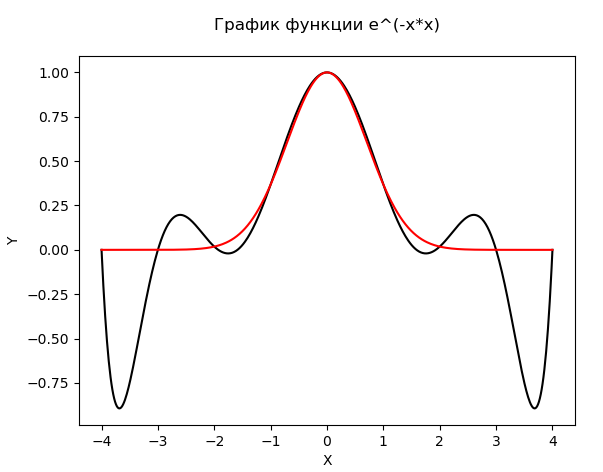
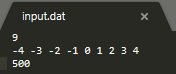


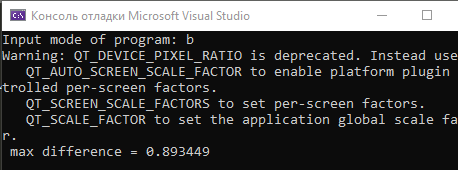


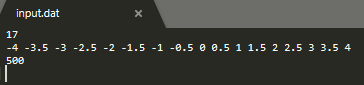


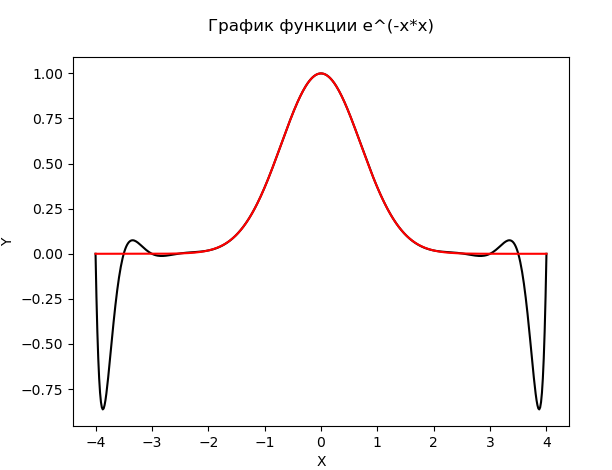
Функция бесконечно дифференцируема, поэтому увеличение числа узлов интерполяции позволяет увеличить точность.

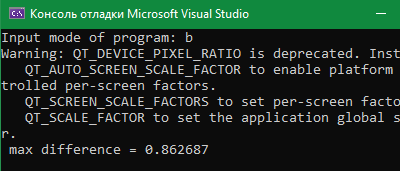
## y =











Для того чтобы получить наиболее точное значение функции с помощью интерполяционного многочлена, нужно выбирать отрезок так, чтобы искомое значение располагалось ближе к середине.

Код решения

#include <iostream>

#include <fstream>

#include <vector>

#include <cmath>

#include <string>

using namespace std;

typedef long double ld;

ld my\_exp(ld x)

{

return expl(-x \* x);

}

//#define DEBUG

#define function my\_exp

string function\_name = "e^(-x\*x)"; // "fabsl"; // "log";

class Interpolator

{

int n = 0;

vector<pair<ld, ld>> values; // таблица значений для интерполяции

vector<ld> Newton\_polynomial; // коэффициенты интерполяционного многочлена Ньютона

public:

Interpolator(vector<pair<ld, ld>> v)

{

values = v;

n = v.size();

find\_Newton\_polynomial();

}

Interpolator(ld f(ld), ld a, ld b, int quantity\_of\_segment)

{

n = quantity\_of\_segment + 1;

values.resize(n);

ld x;

for (int i = 0; i < n; i++)

{

x = a + i \* (b - a) / quantity\_of\_segment;

values[i] = { x, f(x) };

}

find\_Newton\_polynomial();

}

Interpolator(ld f(ld), vector<ld> v)

{

n = v.size();

values.resize(n);

for (int i = 0; i < n; i++)

{

values[i] = { v[i], f(v[i]) };

}

find\_Newton\_polynomial();

}

void find\_Newton\_polynomial()

{

vector<vector<ld>> f(n, vector<ld>(n, 0));

for (int i = 0; i < n; i++)

f[0][i] = values[i].second;

// динамически вычисляем разделенные разности

Newton\_polynomial.assign(n, 0);

Newton\_polynomial[0] = f[0][0];

for (int i = 1; i < n; i++)

{

for (int j = 0; j < n - i; j++)

{

f[i][j] = (f[i - 1][j + 1] - f[i - 1][j]) / (values[j + i].first - values[j].first);

}

Newton\_polynomial[i] = f[i][0];

}

#ifdef DEBUG

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n - i; j++)

{

cout << f[i][j] << '\t';

}

cout << endl;

}

cout << endl;

#endif

}

// вычисление значения с помощью интерполяционного многочлена

ld calculate(ld x)

{

ld res = 0;

ld mult = 1;

for (int i = 0; i < n; i++)

{

res += Newton\_polynomial[i] \* mult;

mult \*= x - values[i].first;

}

return res;

}

ld operator()(ld x) {

return calculate(x);

}

};

int main()

{

ifstream fin("input.dat");

char mode = '0'; // вариант работы программы (a/b)

cout << "Input mode of program: ";

cin >> mode;

if (mode == 'a' || mode == 'A') {

vector<pair<ld, ld>> v;

int n = 0;

fin >> n;

v.resize(n);

// чтение значений функции

for (int i = 0; i < n; i++)

fin >> v[i].first >> v[i].second;

Interpolator i(v);

cout << "Input number of arguments: ";

int l = 0;

cin >> l;

ld k;

// вычисление значений с помощью интерполяции

while (l-- > 0) {

cout << "Input argument: ";

cin >> k;

cout << "Function("<< k << ") = "<< i(k) << endl;

};

}

else if (mode == 'b' || mode == 'B')

{

int n = 0;

vector<ld> x;

vector<pair<ld,ld>> x\_y;

vector<ld> d;

fin >> n;

x.resize(n);

// чтение аргументов функции

for (int i = 0; i < n; i++)

fin >> x[i];

Interpolator interpol\_func(function, x);

ofstream file\_for\_graph("values.dat");

int number\_of\_intervals = 0; // кол-во разбиений для построения графика

fin >> number\_of\_intervals;

x\_y.resize(number\_of\_intervals + 1);

d.resize(number\_of\_intervals + 1);

ld a = x.front(), b = x.back();

for (int i = 0; i <= number\_of\_intervals; i++)

{

ld x = a + i \* (b - a) / number\_of\_intervals;

x\_y[i] = { x , interpol\_func(x) };

}

// запись в файл графопостроителя значений

file\_for\_graph << function\_name << endl; // имени функции

for (auto p : x\_y)

file\_for\_graph << p.first << ' '; // значений аргументов

file\_for\_graph << endl;

for (auto p : x\_y)

file\_for\_graph << p.second << ' '; // значений полученных интерполяцией

file\_for\_graph << endl;

ld max\_modul\_of\_difference = 0;

for (int i = 0; i <= number\_of\_intervals; i++) {

file\_for\_graph << function(x\_y[i].first) << ' '; // значений полученных функцией

d[i] = function(x\_y[i].first) - x\_y[i].second;

if (max\_modul\_of\_difference < fabsl(d[i]))

max\_modul\_of\_difference = fabsl(d[i]);

}

file\_for\_graph << endl;

file\_for\_graph.close();

system("python main.py");

cout << " max difference = " << max\_modul\_of\_difference << endl;

}

return 0;

}

# графопостроитель

import matplotlib.pyplot as plt

fin = open("values.dat", "r")

lable = fin.readline()

values = []

for l in fin:

values.append([float(i) for i in l.split()])

fig = plt.figure()

ax = fig.add\_subplot(111)

ax.set(title = 'График функции ' + lable,

xlabel = 'X',

ylabel = 'Y')

ax.plot(values[0], values[1], color = 'black')

ax.plot(values[0], values[2], color = 'red')

plt.show()