МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»

РУТ (МИИТ)

Кафедра ЦТУТП

**ЛАБОРАТОРНАЯ РАБОТА №2**

ПО ДИСЦИПЛИНЕ «ЧИСЛЕННЫЕ МЕТОДЫ»

НА ТЕМУ «МЕТОД НАИМЕНЬШИХ КВАДРАТОВ»

ВАРИАНТ №6

Выполнил: студент группы

УПМ-311 Масеёнок Е.Н.

Проверил: Посвянский В.П.

МОСКВА 2022

**Постановка задачи**

1. Составить таблицу значений заданной функции на отрезке с шагом . Полученную таблицу значений функции рассматривать в дальнейшем в качестве исходных данных для последующей среднеквадратичной аппроксимации.

2. Составить системы уравнений для определения коэффициентов м.н.с.п. (многочлена наилучшего среднеквадратичного приближения) и решить их с точностью не хуже 0.0001. Под точностью решения СЛАУ понимается наибольшее значение модуля разности между значением левой части уравнений после подстановки полученного решения в систему и соответствующей правой частью уравнений. Для получения требуемой точности студент при необходимости должен использовать разные методы решения СЛАУ, прежде всего метод квадратного корня и линейный одношаговый стационарный метод итераций. Найти эти многочлены порядка не более 1,2,5,6, напечатать их коэффициенты и достигнутую точность.

3. Составить таблицу значений многочленов наилучшего среднеквадратичного приближения не более 1,2,5,6 порядка на отрезке  с шагом . Протабулировать заданную функцию на том же отрезке с шагом  *h1*. Напечатать таблицу.

4. Построить графики аппроксимируемой функции, заданной на дискретном множестве точек п.1. и интерполяционных многочленов, используя полученные значения в таблицах п.3. Вывести на печать таблицы п.3.

На каждом рисунке привести график одного многочлена наилучшего среднеквадратичного приближения и график аппроксимируемой функции.

Функция

**Код программы**

#include <fstream>

#include <cmath>

#include <iomanip>

#include <iostream>

#include <string>

using namespace std;

namespace mod\_gauss {

//Проверка переполнения и неопределённости

void check\_valid(double const& result) {

if (fabs(result) == INFINITY)

throw exception("Возможно переполнение при вычислениях!");

if (isnan(result))

throw exception("Возможна неопределённость при вычислениях");

}

//Подстановка max элемента на место главного

void set\_max(double\*\* matrix, double\* addedPart, int\* vect, int const& row, int len) {

int iMax = row;

int jMax = row;

for (int i = row; i < len; i++) { //Нахождение индексов максимального по модулю элемента

for (int j = row; j < len; j++) {

check\_valid(matrix[i][j]);

if (fabs(matrix[i][j]) > fabs(matrix[iMax][jMax])) {

iMax = i;

jMax = j;

}

}

}

if (matrix[iMax][jMax] == 0)

throw exception("Система несовместна или имеет бесконечное множество решений!");

if (iMax != row) { //Если это не та же строка, то меняем местами с max

for (int j = 0; j < len; j++)

swap(matrix[row][j], matrix[iMax][j]);

swap(addedPart[row], addedPart[iMax]);

}

if (jMax != row) { //Если это не тот же столбец, то меняем местами с max

swap(vect[row], vect[jMax]);

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

swap(matrix[i][row], matrix[i][jMax]);

}

}

//Модифицированный метод Гаусса

void resh\_sist(double\*\* matrix, double\* addedPart, double\* forResult, int const len) {

//Прямой ход

int\* vect = new int[len]; //Вектор нумерации

for (int i = 0; i < len; i++) {

vect[i] = i;

}

for (int i = 0; i < len; i++) {

#ifdef DEBUG

print\_inf(matrix, addedPart);

#endif // DEBUG

check\_valid(matrix[i][i]);

set\_max(matrix, addedPart, vect, i, len);

#ifdef DEBUG

print\_inf(matrix, addedPart);

#endif // DEBUG

for (int j = i + 1; j < len; j++) {

check\_valid(matrix[i][j] / matrix[i][i]);

matrix[i][j] /= matrix[i][i]; //Деление элементов строки на главный элемент

for (int k = i + 1; k < len; k++) {

check\_valid(matrix[k][j] - matrix[i][j] \* matrix[k][i]);

matrix[k][j] -= matrix[i][j] \* matrix[k][i]; //Преобразование других элементов столбца

#ifdef DEBUG

print\_inf(matrix, addedPart);

#endif // DEBUG

}

}

check\_valid(addedPart[i] / matrix[i][i]);

addedPart[i] /= matrix[i][i]; //Преобразование свободного члена главной строки

for (int k = i + 1; k < len; k++) {

check\_valid(addedPart[k] - addedPart[i] \* matrix[k][i]);

addedPart[k] -= addedPart[i] \* matrix[k][i]; //Преобразование свободных членов других строк

check\_valid(matrix[k][i]);

matrix[k][i] = 0; //Преобразование элементов, которые находятся под главным

#ifdef DEBUG

print\_inf(matrix, addedPart);

#endif // DEBUG

}

matrix[i][i] = 1; //Преобразование главного элемента

#ifdef DEBUG

print\_inf(matrix, addedPart);

#endif // DEBUG

}

//Обратный ход

for (int i = len - 1; i >= 0; i--) {

for (int j = 0; j < i; j++) {

addedPart[j] -= matrix[j][i] \* addedPart[i];

matrix[j][i] = 0;

#ifdef DEBUG

print\_inf(matrix, addedPart);

#endif // DEBUG

}

forResult[vect[i]] = addedPart[i];

}

#ifdef DEBUG

print\_inf(matrix, addedPart);

#endif // DEBUG

delete[] vect;

}

}

//Исходная функция

double task\_func(double x) {

return sqrt(x \* x) \* (1 / exp(x));  
}

//Подсчёт коэффициентов МНСП

void count\_coefs\_MNSP(double const\* X, double const\* Y, int const numOfSteps, double\* coefs, int const numCoefs) {

int const matrSize = numCoefs + 1; //Размерность матрицы

double\*\* matrix = new double\* [matrSize]; //Сама матрица

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

matrix[i] = new double[matrSize];

double\* rightPart = new double[matrSize]; //Правая часть

for (int i = 0; i < matrSize; i++) {

for (int j = 0; j < matrSize; j++) {

//Подсчёт коэффициентов матрицы

matrix[i][j] = 0;

for (int k = 0; k <= numOfSteps; k++)

matrix[i][j] += pow(X[k], i + j);

}

//Подсчёт коэффициентов правой части

rightPart[i] = 0;

for (int k = 0; k <= numOfSteps; k++)

rightPart[i] += Y[k] \* pow(X[k], i);

}

mod\_gauss::resh\_sist(matrix, rightPart, coefs, matrSize);

delete[] rightPart;

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

delete[] matrix[i];

delete[] matrix;

}

//Функция МНСП

double count\_MNSP(double x, double\* coefs, int const numCoef) {

double result = 0;

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

result += coefs[i] \* pow(x, i);

return result;

}

//Форматирование вывода double в файл .csv

string format\_double(double num) {

string strNum = to\_string(num);

for (int i = 0; i < strNum.length(); i++)

if (strNum[i] == ',') {

strNum[i] = '.';

break;

}

return strNum;

}

//Построение таблицы в консоли и в файле .csv

void build\_and\_parse\_table(double const a, double const b, int numOfSteps, double\*\* coefs, int const\* numCoef) {

double h1 = (b - a) / numOfSteps; //Величина шага

double x = a; //Аргумент для функций

ofstream fout; //Вывод результатов в файл

fout.open("results.csv");

bool isOpen = fout.is\_open(); //Условие открытия файла

//Построение таблицы и вывод в файл

cout << setw(8) << "x" << '|' << setw(8) << "f(x)" << '|'

<< setw(8) << "P1(x)" << '|' << setw(8) << "P2(x)" << '|'

<< setw(8) << "P5(x)" << '|' << setw(8) << "P6(x)" << '|' << endl;

if (isOpen)

fout << "x" << ';' << "f(x)" << ';'

<< "P1(x)" << ';' << "P2(x)" << ';'

<< "P5(x)" << ';' << "P6(x)" << '\n';

for (int i = 0; i <= numOfSteps; i++) {

cout << setw(8) << x << '|' << setw(8) << task\_func(x) << '|';

if (isOpen)

fout << format\_double(x) << ';' << format\_double(task\_func(x));

for (int j = 0; j < 4; j++) {

cout << setw(8) << count\_MNSP(x, coefs[j], numCoef[j]) << '|';

if (isOpen)

fout << ';' << format\_double(count\_MNSP(x, coefs[j], numCoef[j]));

}

cout << endl;

if (isOpen)

fout << '\n';

x += h1; //Увеличение аргумента

}

if (isOpen)

fout.close();

}

int main() {

setlocale(LC\_ALL, "ru");

cout.setf(ios::fixed);

cout.precision(4);

//Отрезок аппроксимации

double a, b;

cout << "Введите границы отрезка: ";

cin >> a >> b;

cout << endl;

int const numOfSteps = 10; //Количество отрезков между a и b

//Значения аргументов и функции

double X[numOfSteps + 1], Y[numOfSteps + 1];

for (int i = 0; i <= numOfSteps; i++) {

X[i] = a + i \* (b - a) / numOfSteps;

Y[i] = task\_func(X[i]);

}

//Построение таблицы значений аргументов и функции

cout << setw(3) << "X|";

for (int i = 0; i <= numOfSteps; i++) {

cout << setw(8) << X[i] << '|';

}

cout << endl;

cout << setw(3) << "Y|";

for (int i = 0; i <= numOfSteps; i++) {

cout << setw(8) << Y[i] << '|';

}

cout << endl << endl;

int numCoef[4] = { 1, 2, 5, 6 }; //Массив для количества коэффициентов МНСП

double\*\* coefs = new double\* [4]; //Массив для коэффициентов МНСП

for (int i = 0; i < 4; i++)

coefs[i] = new double[numCoef[i] + 1];

for (int i = 0; i < 4; i++) {

count\_coefs\_MNSP(X, Y, numOfSteps, coefs[i], numCoef[i]); //Подсчёт коэффициентов МНСП

//Вывод таблицы коэффициентов МНСП

cout << "n=" << numCoef[i] << '|';

for (int j = 0; j <= numCoef[i]; j++) {

cout << setw(8) << coefs[i][j] << '|';

}

cout << endl;

}

cout << endl << endl;

//Вывод таблицы для построения графиков на экран и в файл

build\_and\_parse\_table(a, b, 50, coefs, numCoef);

for (int i = 0; i < 4; i++)

delete[] coefs[i];

delete[] coefs;

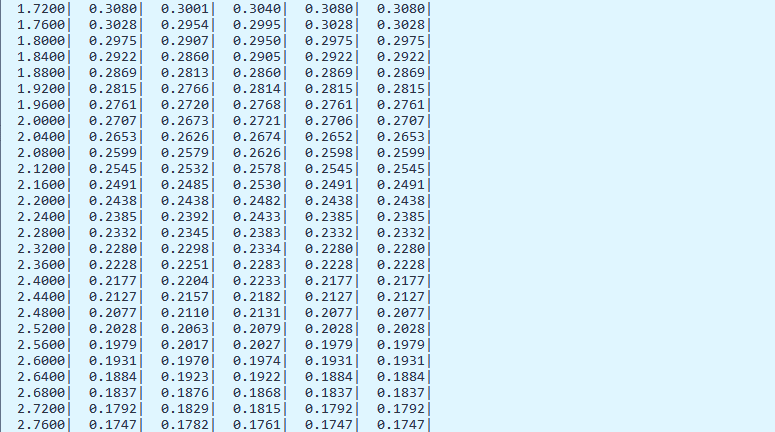
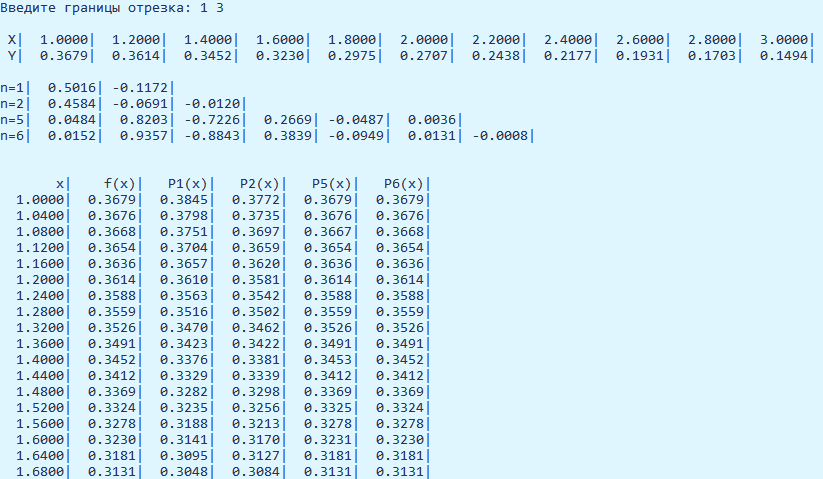
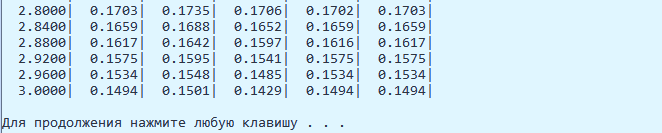
cout << endl;

system("pause");

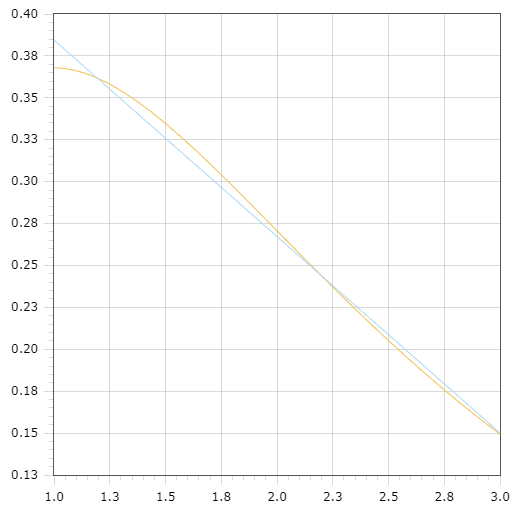
return 0;

}

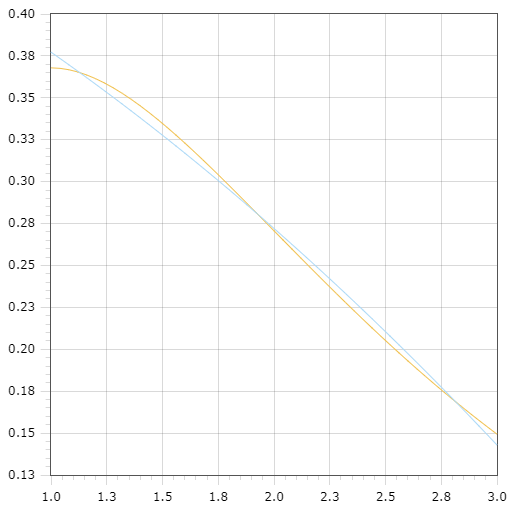
**Результаты работы программы**

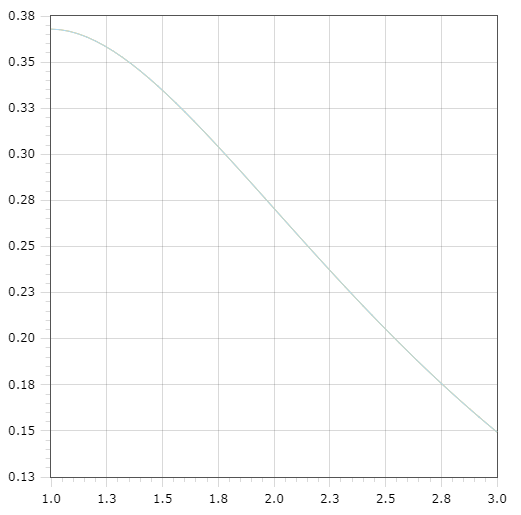
  


Графики функций

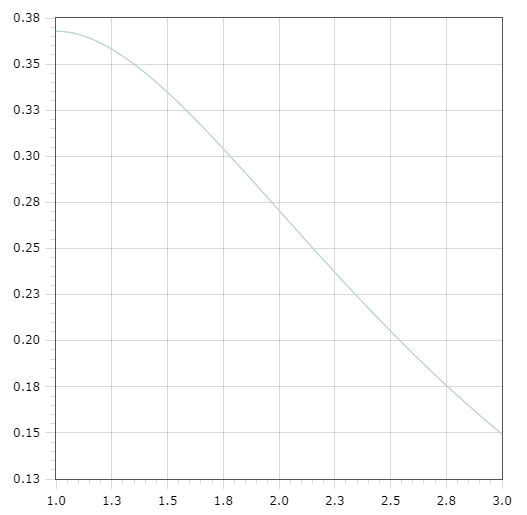


Жёлтый – f(x), синий – P1(x)

  
Жёлтый – f(x), синий – P2(x)



Жёлтый – f(x), синий – P5(x)

  
Жёлтый – f(x), синий – P6(x)

**Вывод**

Как видно по графикам, функция аппроксимируется с помощью МНСП достаточно точно. Графики многочленов 5-ой и 6-ой степени совпадают с графиком исходной функции.