Министерство образования Республики Беларусь

Учреждение образования “Брестский Государственный Технический Университет”

Кафедра ИИТ

Лабораторная работа №3,4

По дисциплине Вычислительная математика

“**Интерполирование функций,**

**Численное интегрирование”**

Выполнил: студент 1 курса

Группы АС-59

Быбко Т.А.

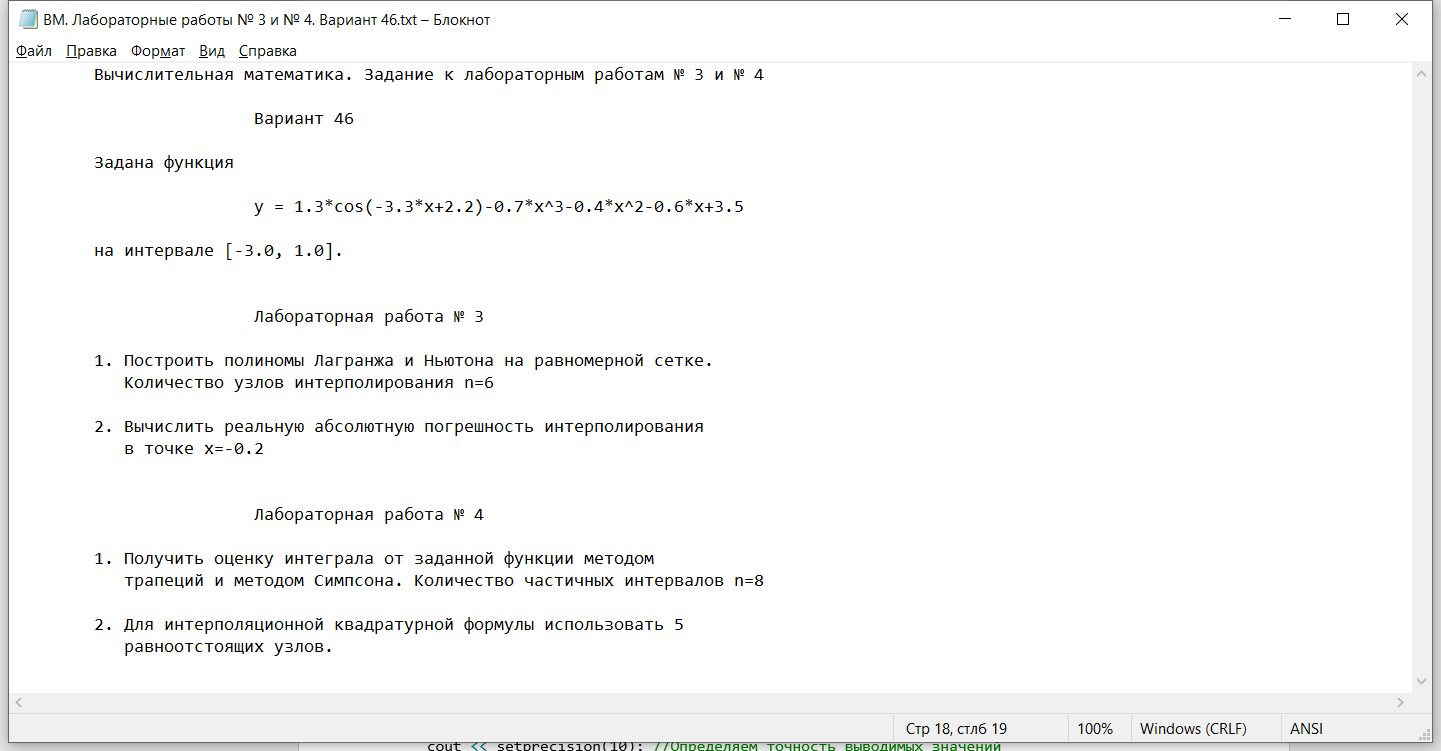
Проверил: Пролиско Е.Е.

Брест 2021

Вариант 46

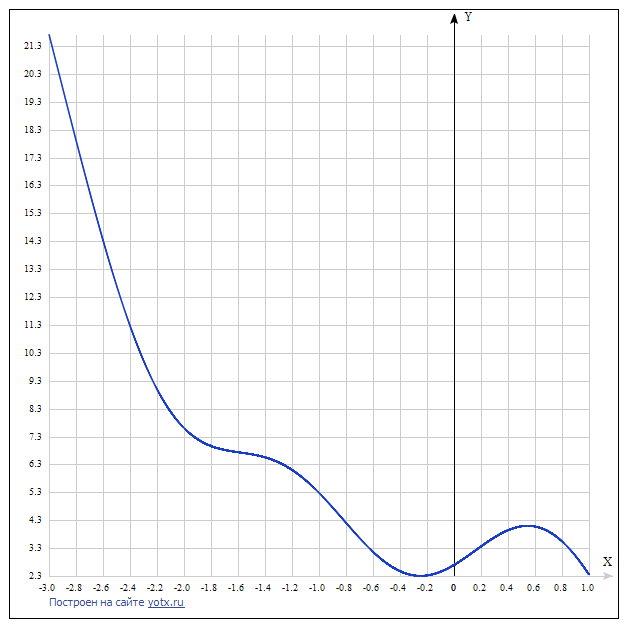
Цель: научиться строить и вычислять полиномы (Ньютона и Лагранжа), а также написать программу, выполняющую численное интегрирование по методам трапеций, Симпсона и интерполяционной квадратурной формулы.

Условие задания:



Ход работы:

Сначала нужно построить график начальной функции в заданном интервале:



Теперь напишем нужную нам программу.

Код программы:

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

double Function(double x);

int main()

{

setlocale(LC\_ALL, "Russian");

cout << setprecision(10); //Определяем точность выводимых значений

const int n = 6; //Количество узлов интерполирования

//Выберем узлы интерполирования, взяв за первое значение начало интервала, а за конечное - конец, остальные находятся равноудалённо с отступом 0.8

//Значение было найдено по формуле (1 - (-3))\5

double X[n] = { -3, -2.2, -1.4, -0.6, 0.2, 1 };

double x = -0.2;

//Найдём значения функции

double Y[n];

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

Y[i] = Function(X[i]);

cout << "Значения функции в узлах и сами узлы: " << endl;

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

cout << "Y: " << Y[i] << "\tX: " << X[i] << endl;

//Построим полином Лагранжа

double A[n] = { 0 }; //Отвечает за значение у х (в разных степенях) в формуле полинома Лагранжа

int Index[n - 1] = { 1, 2, 3, 4, 5 };

double Znam[n] = { 1, 1, 1, 1, 1, 1 }; //Отвечает за значения в формуле полинома Лагранжа, которые можно спокойно посчитать

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

{

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

{

if (j == i)

continue;

Znam[i] \*= (X[i] - X[j]);

}

for (int j = 0; j < n - 1; j++)

{

if (Index[j] == i)

Index[j] -= 1;

}

A[0] += (X[Index[0]] \* X[Index[1]] \* X[Index[2]] \* X[Index[3]] \* X[Index[4]]) \* (-1) / Znam[i] \* Y[i];

A[1] += (X[Index[0]] \* X[Index[1]] \* X[Index[2]] \* X[Index[3]] + X[Index[0]] \* X[Index[1]] \* X[Index[2]] \* X[Index[4]] +

X[Index[0]] \* X[Index[1]] \* X[Index[3]] \* X[Index[4]] + X[Index[1]] \* X[Index[2]] \* X[Index[3]] \* X[Index[4]] +

X[Index[0]] \* X[Index[2]] \* X[Index[3]] \* X[Index[4]]) / Znam[i] \* Y[i];

A[2] += (X[Index[0]] \* X[Index[1]] \* X[Index[2]] + X[Index[0]] \* X[Index[1]] \* X[Index[3]] + X[Index[1]] \* X[Index[2]] \* X[Index[3]] +

X[Index[0]] \* X[Index[2]] \* X[Index[3]] + X[Index[0]] \* X[Index[1]] \* X[Index[4]] + X[Index[1]] \* X[Index[2]] \* X[Index[4]] +

X[Index[0]] \* X[Index[2]] \* X[Index[4]] + X[Index[1]] \* X[Index[3]] \* X[Index[4]] + X[Index[0]] \* X[Index[3]] \* X[Index[4]] +

X[Index[2]] \* X[Index[3]] \* X[Index[4]]) \* (-1) / Znam[i] \* Y[i];

A[3] += (X[Index[0]] \* X[Index[1]] + X[Index[1]] \* X[Index[2]] + X[Index[0]] \* X[Index[2]] + X[Index[1]] \* X[Index[3]] +

X[Index[0]] \* X[Index[3]] + X[Index[2]] \* X[Index[3]] + X[Index[1]] \* X[Index[4]] + X[Index[0]] \* X[Index[4]] +

X[Index[2]] \* X[Index[4]] + X[Index[3]] \* X[Index[4]]) / Znam[i] \* Y[i];

A[4] += (X[Index[0]] + X[Index[1]] + X[Index[2]] + X[Index[3]] + X[Index[4]]) \* (-1) / Znam[i] \* Y[i];

A[5] += Y[i] / Znam[i];

}

cout << "Полином Лагранжа: ";

for (int i = n - 1; i >= 0; i--)

{

cout << A[i] << "X^" << i << " ";

if (i == 3 || i == 2 || i == 1)

cout << " + ";

}

//Найдём абсолютную погрешность в точке х

double P = A[5] \* pow(x, 5) + A[4] \* pow(x, 4) + A[3] \* pow(x, 3) + A[2] \* pow(x, 2) + A[1] \* pow(x, 1) + A[0];

cout << endl << "Полином Лагранжа равен: " << P << endl;

cout << "Действительноe значение: " << Function(x) << endl;

cout << "Aбсолютная погрешность: " << fabs(Function(x) - P) << endl;

//Найдём полином Ньютона (строить не будем, т.к. уже был построен полином Лагранжа, сделано это было исключительно для графика, который будет схож)

//Разделённые разности

double Razn2[5] = { 0 }; //2 аргумента

double Razn3[4] = { 0 }; //3

double Razn4[3] = { 0 }; //4

double Razn5[2] = { 0 }; //5

double Razn6; //6

cout << endl << "Разделённые разности: " << endl;

for (int i = 0; i < 5; i++)

Razn2[i] = (Function(X[i + 1]) - Function(X[i])) / (X[i + 1] - X[i]);

for (int i = 0; i < 5; i++)

cout << Razn2[i] << "\t";

cout << endl;

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

Razn3[i] = (Razn2[i + 1] - Razn2[i]) / (X[i + 2] - X[i]);

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

cout << Razn3[i] << "\t";

cout << endl;

for (int i = 0; i < 3; i++)

Razn4[i] = (Razn3[i + 1] - Razn3[i]) / (X[i + 3] - X[i]);

for (int i = 0; i < 3; i++)

cout << Razn4[i] << "\t";

cout << endl;

for (int i = 0; i < 2; i++)

Razn5[i] = (Razn4[i + 1] - Razn4[i]) / (X[i + 4] - X[i]);

for (int i = 0; i < 2; i++)

cout << Razn5[i] << "\t";

cout << endl;

Razn6 = (Razn5[1] - Razn5[0]) / (X[5] - X[0]);

cout << Razn6 << endl;

//Найдём значение полинома Ньютона в точке х

double P2 = Function(X[0]) + Razn2[0] \* (x - X[0]) + Razn3[0] \* (x - X[0]) \* (x - X[1]) + Razn4[0] \* (x - X[0]) \* (x - X[1]) \* (x - X[2]) +

Razn5[0] \* (x - X[0]) \* (x - X[1]) \* (x - X[2]) \* (x - X[3]) + Razn6 \* (x - X[0]) \* (x - X[1]) \* (x - X[2]) \* (x - X[3]) \* (x - X[4]);

cout << "Полином Ньютона равен: " << P2 << endl;

cout << "Действительное значение: " << Function(x) << endl;

cout << "Абсолютная погрешность равна: " << fabs(Function(x) - P2);

return 0;

}

double Function(double x)

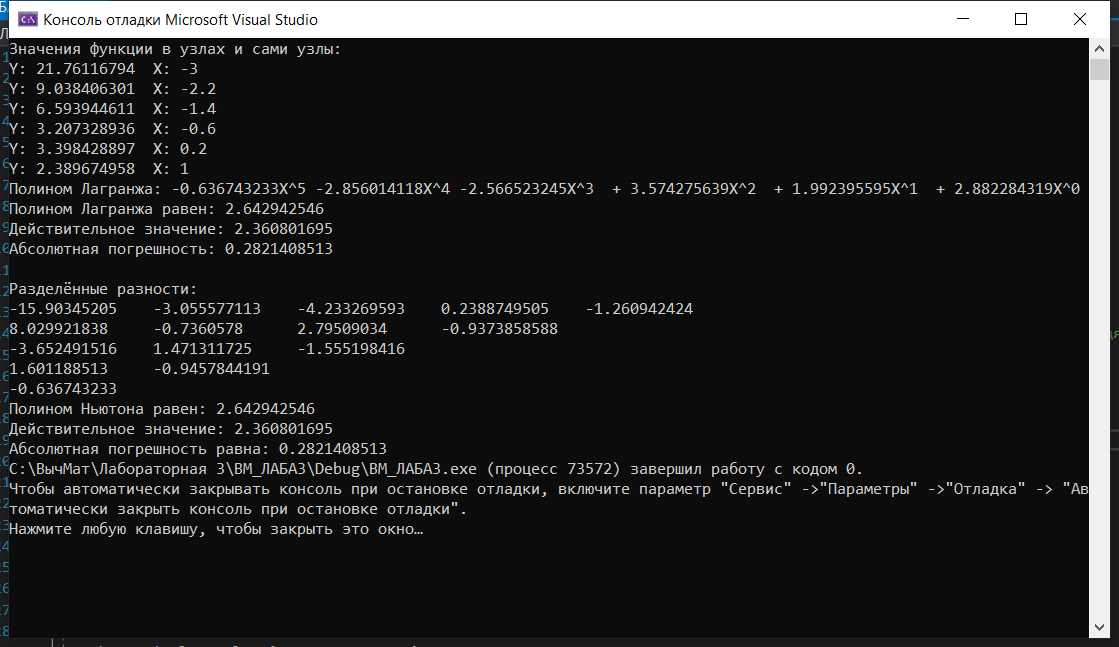
{

double y = 1.3 \* cos(-3.3 \* x + 2.2) - 0.7 \* pow(x, 3) - 0.4 \* pow(x, 2) - 0.6 \* x + 3.5; //Данная по условию функция

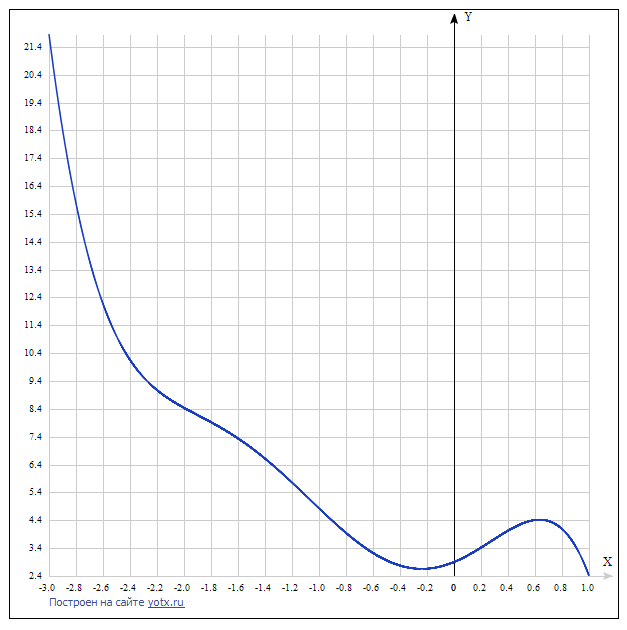
return y;

}

Результат выполнения программы:



Для того, чтобы убедиться в правильности, построим график полинома и сравним его с графиком начальной функции:



Как видно, графики отличаются лишь слегка, но во всех узлах интерполирования значения функции равны между собой, это значит, что задание было выполнено правильно.

Теперь приступим к численному интегрированию.

Код программы:

#include <iostream>

#include <iomanip>

#include <cmath>

using namespace std;

double Function(double x);

int main()

{

setlocale(LC\_ALL, "Russian");

cout << setprecision(10);

//Метод трапеций

const int n = 9; //Кол-во узлов (на 1 больше, чем количество частичных интервалов)

double I = 26.84061; //Значение интеграла данной нам функции

double h = (1.0 + 3.0) / 8;

double X[n];

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

X[i] = i \* h - 3.0;

double Y[n];

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

Y[i] = Function(X[i]);

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

cout << "Y: " << Y[i] << "\tX: " << X[i] << endl;

double I1 = 0.0; //Значение интеграла с шагом 0.5

for (int i = 0; i < n - 1; i++) //Условие n - 1 для того, чтобы не проходить через последний элемент (так надо для формулы)

{

if (i == 0)

{

I1 += ((Y[i] + Y[n - 1]) / 2) \* h;

continue;

}

I1 += Y[i] \* h;

}

cout << "Значение интеграла с шагом 0.5: " << I1 << endl;

//Разобьём интеграл на (n-1)\*2 = 16 частей

const int n2 = n \* 2 - 1; //Получится 17 узлов

double h2 = (1.0 + 3.0) / ((n - 1) \* 2);

double X2[n2];

for (int i = 0; i < n2; i++)

X2[i] = i \* h2 - 3.0;

double Y2[n2];

for (int i = 0; i < n2; i++)

Y2[i] = Function(X2[i]);

for (int i = 0; i < n2; i++)

cout << "Y2: " << Y2[i] << "\tX2: " << X2[i] << endl;

double I2 = 0.0; //Значение интеграла с шагом 0.25

for (int i = 0; i < n2 - 1; i++)

{

if (i == 0)

{

I2 += ((Y2[i] + Y2[n2 - 1]) / 2) \* h2;

continue;

}

I2 += Y2[i] \* h2;

}

cout << "Значение интеграла с шагом 0.25: " << I2 << endl;

cout << "Погрешность по правилу Рунге: " << fabs(I2 - I1) / 3 << endl;

cout << "Абсолютная погрешность: " << fabs(I - I2) << endl;

//Применим метод Симпсона

//Разобьём исходный отрезок на 4 сдвоенных отрезка, n, h будут равны

double I3 = 0.0; //Интеграл для метода Симпсона с шагом 0.5

I3 += (Y[0] + Y[n - 1]) \* (h / 3);

for (int i = 1; i < ((n - 1) / 2); i++) //В условии будет i < 4

I3 += Y[i \* 2] \* 2 \* (h / 3);

for (int i = 1; i < (n - 1) / 2 + 1; i++) //В условии будет i < 5

I3 += Y[i \* 2 - 1] \* 4 \* (h / 3);

cout << "Интеграл в методе Симпсона с 4 сдвоенными отрезками равен: " << I3 << endl;

//Разобьём исходный отрезок на 8 сдвоенных отрезка, n2, h2 будут равны

double I4 = 0.0; //Интеграл для метода Симпсона с шагом 0.25

I4 += (Y2[0] + Y2[n2- 1]) \* (h2 / 3);

for (int i = 1; i < ((n2 - 1) / 2); i++) //В условии будет i < 8

I4 += Y2[i \* 2] \* 2 \* (h2 / 3);

for (int i = 1; i < (n2 - 1) / 2 + 1; i++) //В условии будет i < 9

I4 += Y2[i \* 2 - 1] \* 4 \* (h2 / 3);

cout << "Интеграл в методе Симпсона с 8 сдвоенными отрезками равен: " << I4 << endl;

cout << "Погрешность по правилу Рунге: " << fabs(I4 - I3) / 15 << endl;

cout << "Абсолютная погрешность: " << fabs(I - I4) << endl;

//Используем интерполяционную квадратурную формулу

const int n3 = 5; //Равноотстоящие узлы

double h3 = (1.0 + 3.0) / (n3 - 1); //Определим шаг для 5 узлов

double X3[n3];

for (int i = 0; i < n3; i++)

X3[i] = i \* h3 - 3.0;

double Y3[n3];

for (int i = 0; i < n3; i++)

Y3[i] = Function(X3[i]);

for (int i = 0; i < n3; i++)

cout << "Y3: " << Y3[i] << "\tX3: " << X3[i] << endl;

//Создадим и заполним матрицу

double A[n3][n3];

double B[n3];

for (int i = 0; i < n3; i++)

{

B[i] = (pow(X3[n3 - 1], i + 1) - pow(X3[0], i + 1)) / ((double)i + 1.0);

for (int j = 0; j < n3; j++)

A[i][j] = pow(X3[j], i);

}

cout << "Исходная матрица: " << endl;

for (int i = 0; i < n3; i++)

{

for (int j = 0; j < n3; j++)

cout << setw(8) << A[i][j] << " ";

cout << setw(8) << B[i] << endl;

}

//Создадим расширенную матрицу АВ

double AB[n3][n3 + 1];

for (int i = 0; i < n3; i++)

{

for (int j = 0; j < n3; j++)

{

AB[i][j] = A[i][j];

}

AB[i][n3] = B[i];

}

//Решим систему методом Гаусса

for (int i = 0; i < n3; i++)

{

double first = AB[i][i]; //Выберем первый ненулевой элемент, для того, чтобы сделать из него единицу, а значит разделить на его значение всю строку

for (int j = 0; j < n3 + 1; j++)

AB[i][j] /= first;

for (int j = i + 1; j < n3; j++)

{

double obnul = AB[j][i]; //Начинаем двигаться вниз по столбцу, выбирая элемент и приводя его к нулю

for (int g = 0; g < n3 + 1; g++) //Двигаемся по строке, чей эелемент мы выбрали строчкой выше

AB[j][g] -= obnul \* AB[i][g]; //Отнимаем из текущей строки начальную строку, домноженную на эелемент obnul, что приведёт к "обнулению" нужного нам элемента

}

}

cout << "Изменённая матрица: " << endl;

for (int i = 0; i < n3; i++)

{

for (int j = 0; j < n3 + 1; j++)

cout << setw(8) << AB[i][j] << " ";

cout << endl;

}

double C[n3]; //Массив искомых коэффициентов

C[n3 - 1] = AB[4][5];

C[n3 - 2] = AB[3][5] - 4 \* C[n3 - 1];

C[n3 - 3] = AB[2][5] - 6 \* C[n3 - 1] - 3 \* C[n3 - 2];

C[n3 - 4] = AB[1][5] - 4 \* C[n3 - 1] - 3 \* C[n3 - 2] - 2 \* C[n3 - 3];

C[n3 - 5] = AB[0][5] - C[n3 - 1] - C[n3 - 2] - C[n3 - 3] - C[n3 - 4];

cout << "Коэффициенты: " << endl;

for (int i = 0; i < n3; i++)

cout << C[i] << "\t";

cout << endl;

double I5 = 0.0; //Интеграл для интерполяционной квадратурной формулы

for (int i = 0; i < n3; i++)

I5 += Y3[i] \* C[i];

cout << "Интеграл по интерполяционной квадратурной формуле: " << I5 << endl;

cout << "Абсолютная погрешность равна: " << fabs(I - I5) << endl;

return 0;

}

double Function(double x)

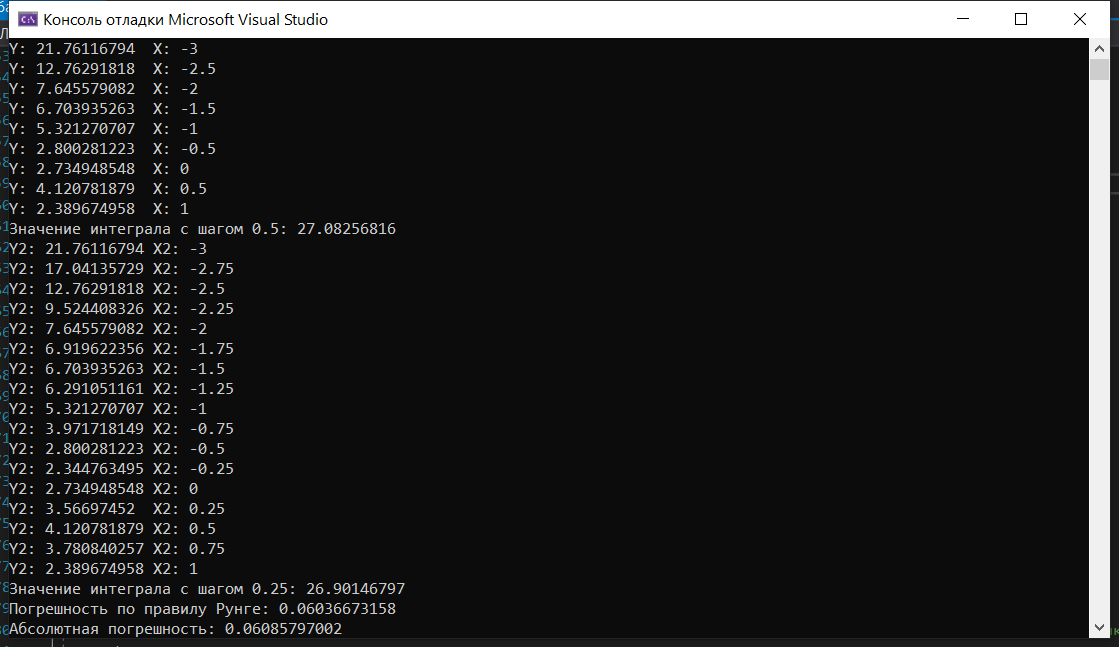
{

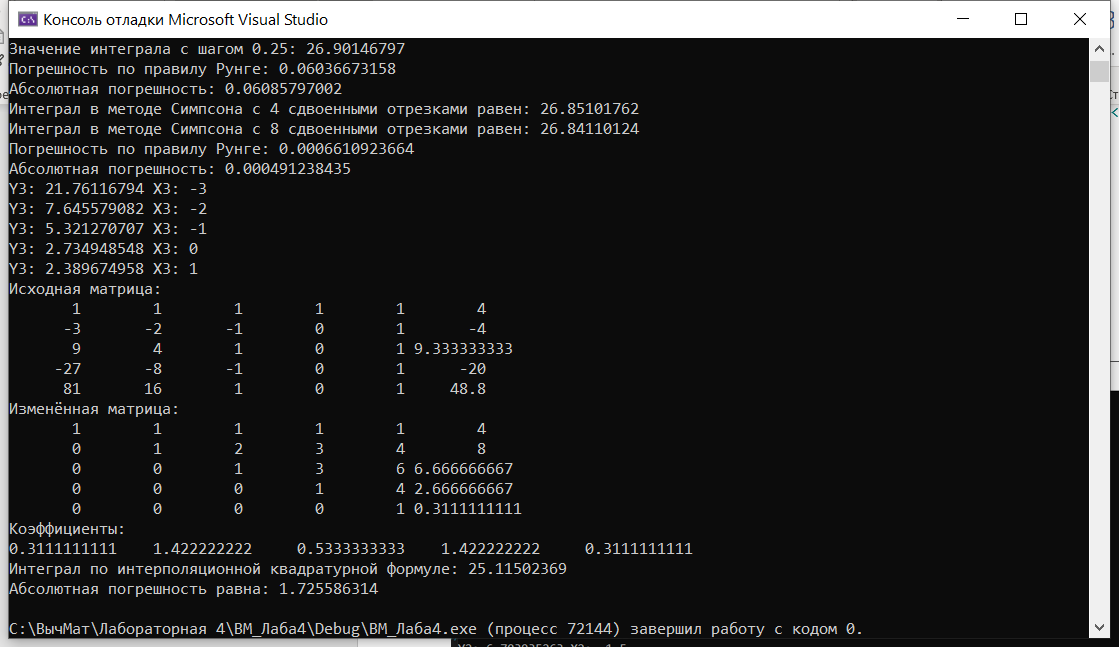
double y = 1.3 \* cos(-3.3 \* x + 2.2) - 0.7 \* pow(x, 3) - 0.4 \* pow(x, 2) - 0.6 \* x + 3.5; //Данная по условию функция

return y;

}

Результат выполнения программы:





Вывод: в ходе выполнения этой работы пришлось столкнуться с множеством трудностей, особенно при построении полинома Лагранжа, что пришлось делать вручную. Однако удалось получить необходимый результат. В численном интегрировании метод трапеций и метод Симпсона получили свои результаты с минимальной погрешностью, чего, к сожалению, нельзя сказать про интерполяционную квадратурную формулу. Однако проверив все значения функций в узлах (достаточно сравнить Y3 и предыдущие массивы, хранящие значения функций), а также убедившись в том, что коэффициенты, полученные путём решения матрицы также верны (их вывод осуществлялся фактически вручную), можно сделать вывод о том, что ответ и должен быть таким.