МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа №2**по дисциплине: «Вычислительная математика»

Выполнил: ст. группы ПВ-211

Медведев Д.С.

Проверила:

Бондаренко Т.В.

Белгород 2023 г.

Интерполяция функций

# Вариант 8

**Цель работы:** изучить понятие интерполяции, основные свойства интерполяционной функции, способы задания интерполяционной функции; понятие интерполяционного многочлена; изучить способы построения интерполяционного многочлена для случая равномерной и неравномерной сетки интерполяции; получить практические навыки решения задачи интерполяции с помощью ЭВМ.

**Ход работы**

****

1. Найти область допустимых значений переменной х для функции y = f(x) задания соответствующего варианта.

2. Составить таблицу значений функции фy=f(x),используя(n≥6) узлов интерполяции (xi ≠ a, где a точка, не являющаяся узлом интерполяционной сетке, в которой необходимо приближенно вычислить значение функции в соответствии с вариантом задания; x0 < a < xn).

3. По полученной таблице значений функции y = f(x) составить интерполяционный многочлен Лагранжа для случаев линейной, квадратичной и кубической интерполяции: L1(x), L2(x), L3(x).  
Замечание. Интервал (x0, xn), n = 1, 2, 3, используемый для построения интерполяционного многочлена Лагранжа должен содержать точку a.

4. По таблице значений функции составить интерполяционный многочлен Ньютона In(x). При построении интерполяционного многочлена Ньютона необходимо использовать конечные разности для случая равномерной сетки интерполяции и разделенные разности для неравномерной сетки интерполяции. Можно построить таблицу значений функции для равномерной сетки, выполнить построение многочлена Ньютона с конечными разностями, затем убрать 1 значение из середины таблицы и выполнить построение многочлена Ньютона с разделенными разностями для получившейся неравномерной сетки интерполяции.

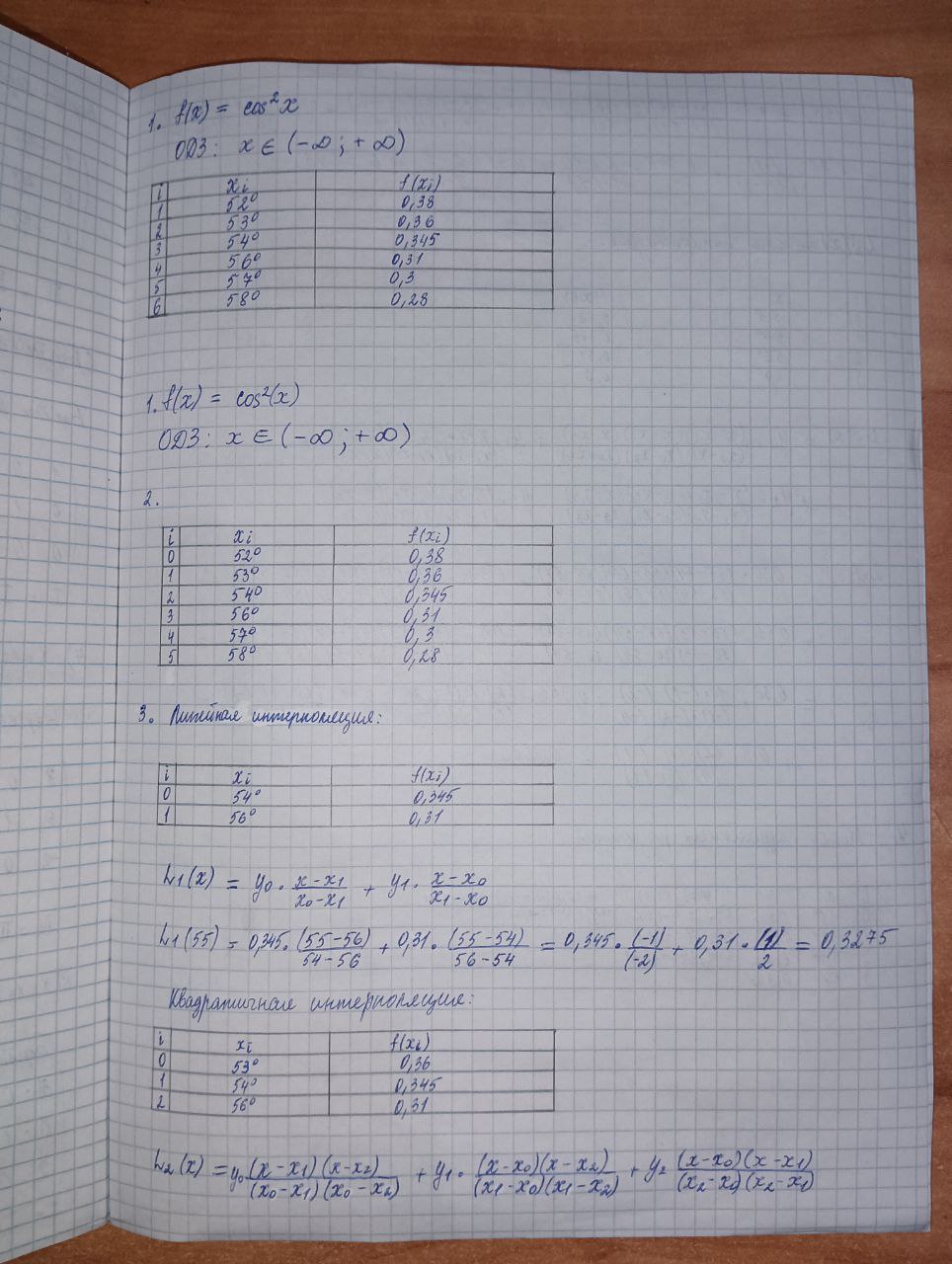
5. Вычислить точное значение функции y = f(x) при x=a (yT = f(a)).

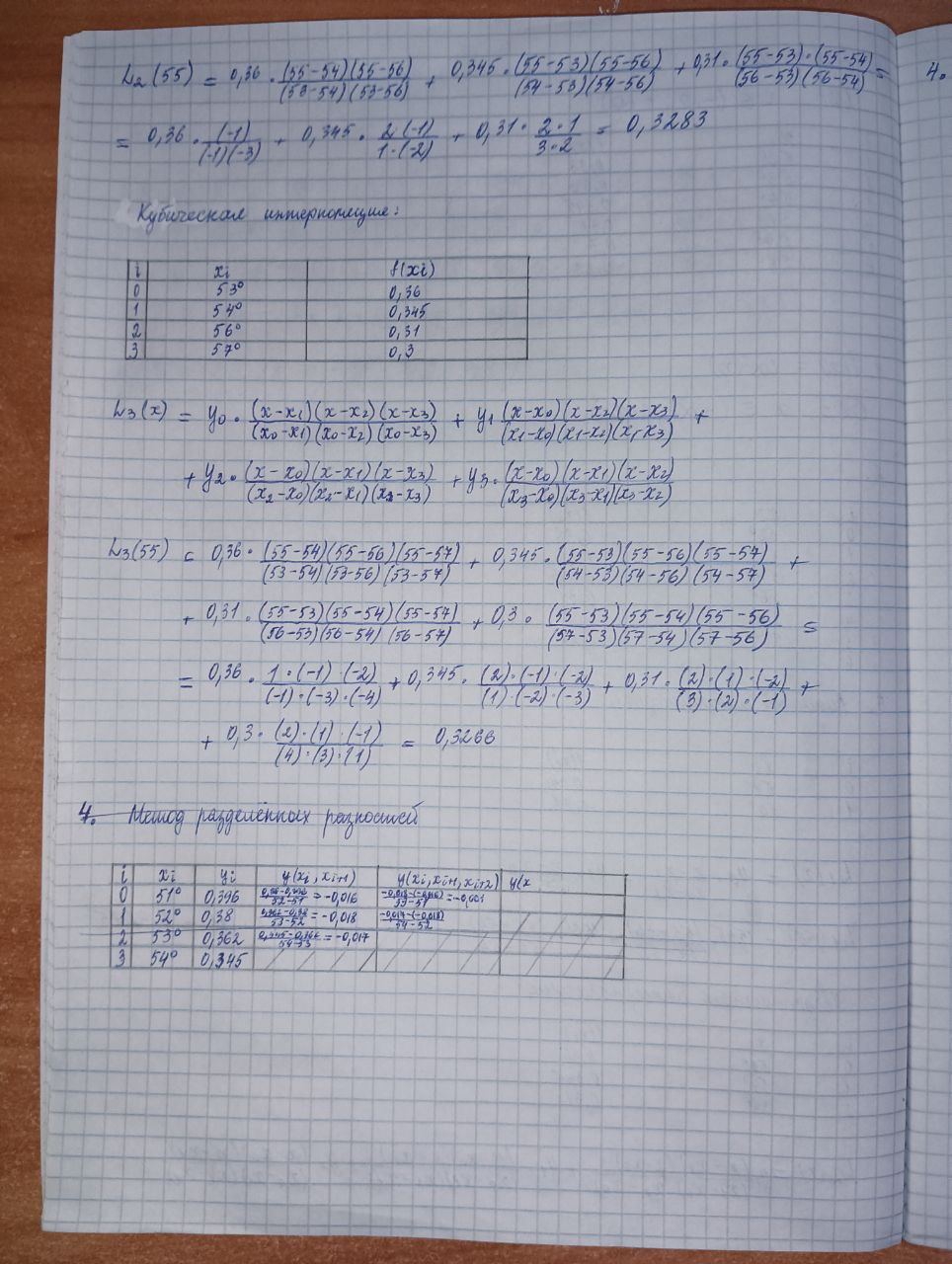
6. Вычислить приближенное значение функции при x=a по всем полученным интерполяционным многочленам.

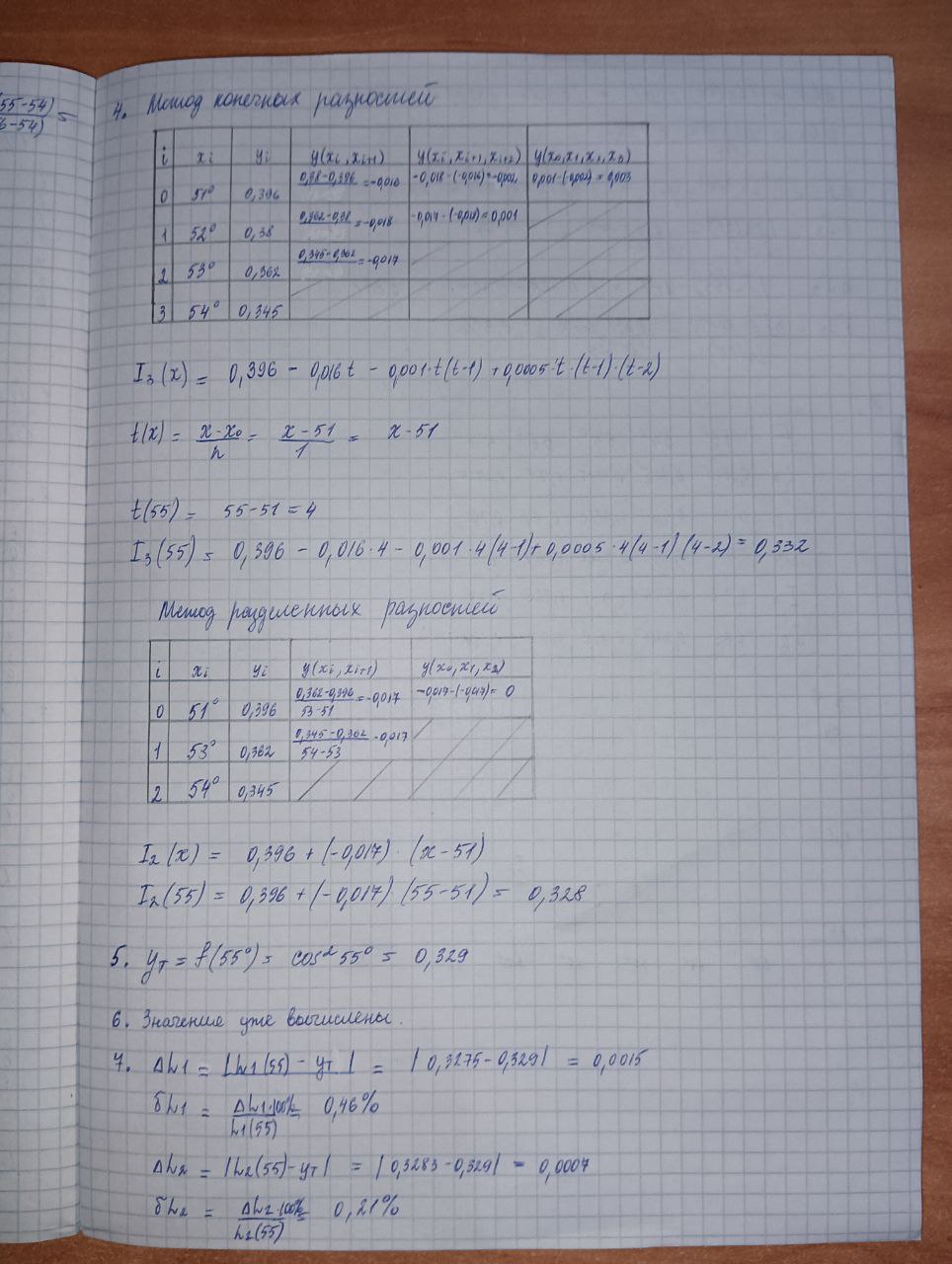
7. Определить абсолютную ∆ и относительную δ погрешность вычисления значения функции для каждого интерполяционного многочлена (интерполяционные многочлены Лагранжа и Ньютона) при заданном значении х=а.

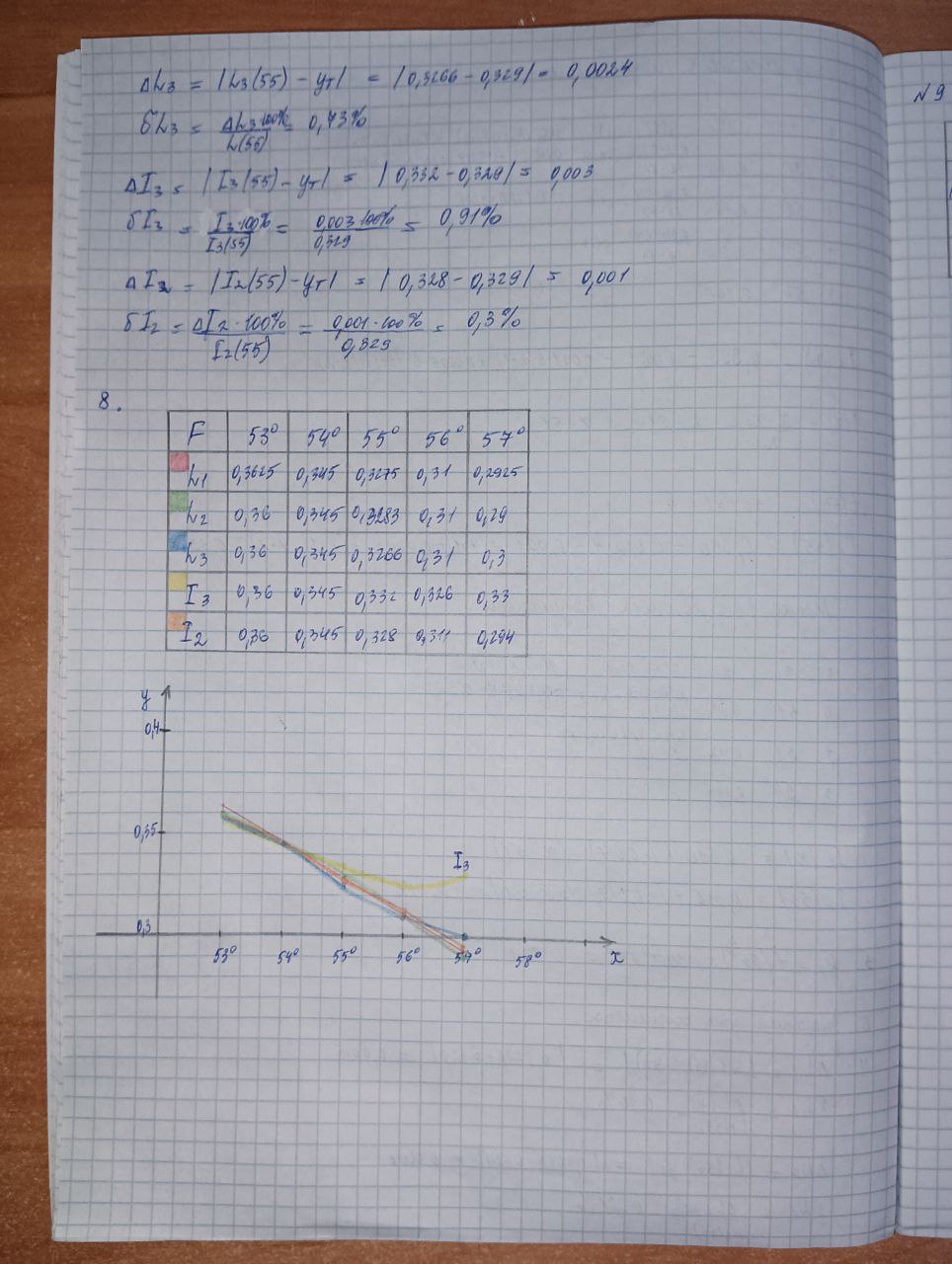
8. Построить в одной системе координат графики полученных интерполяционных функций (многочлены Лагранжа и Ньютона), исходной функции y= f(x) и отметить значения функций в точке x=a.

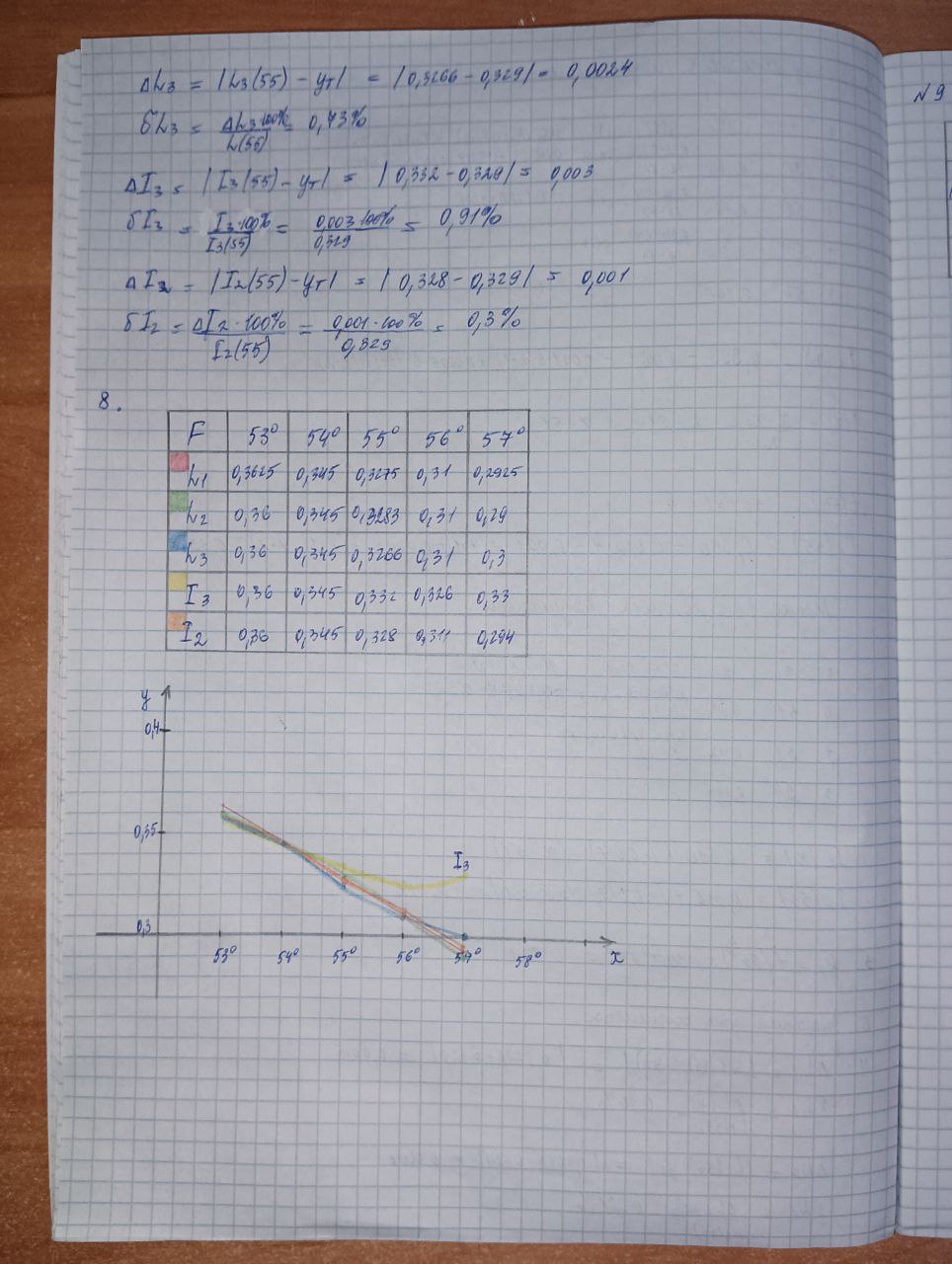
9. Представить полученные результаты в виде таблицы (см. табл. 2.1).

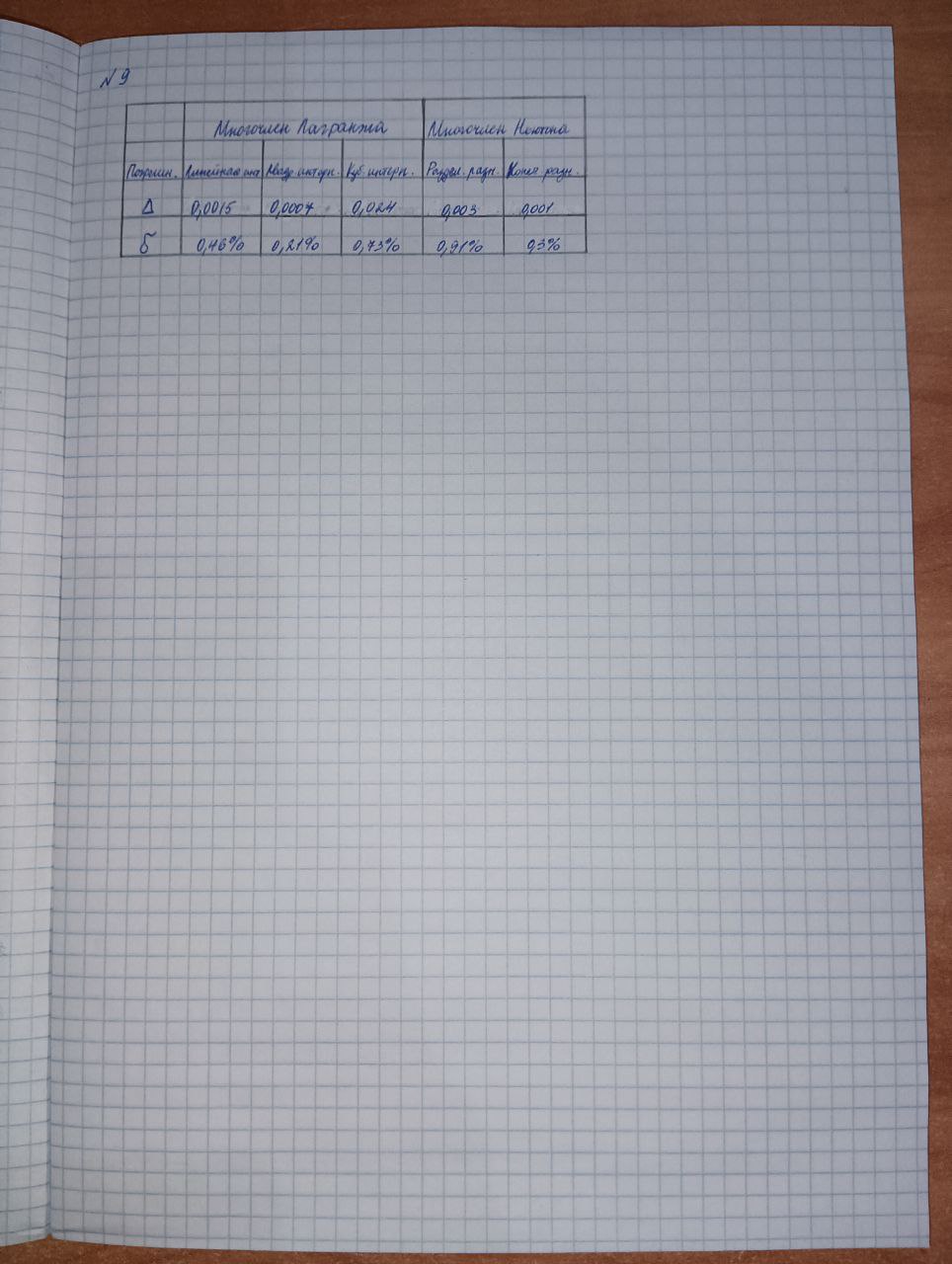












10. Составить программу, реализующую вычисление приближенного значения функции в произвольной точке путем построения интерполяционного многочлена Ньютона для случая равномерной и неравномерной сетки.

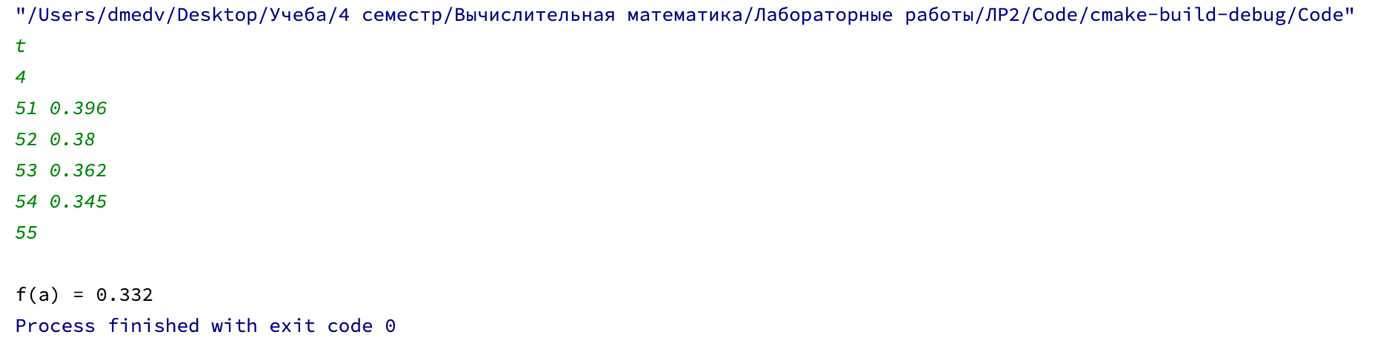
Входные данные: значения узлов интерполяционной сетки xi, значения функции в узлах интерполяции yi. Предусмотреть возможность ввода значений с клавиатуры и из файла.

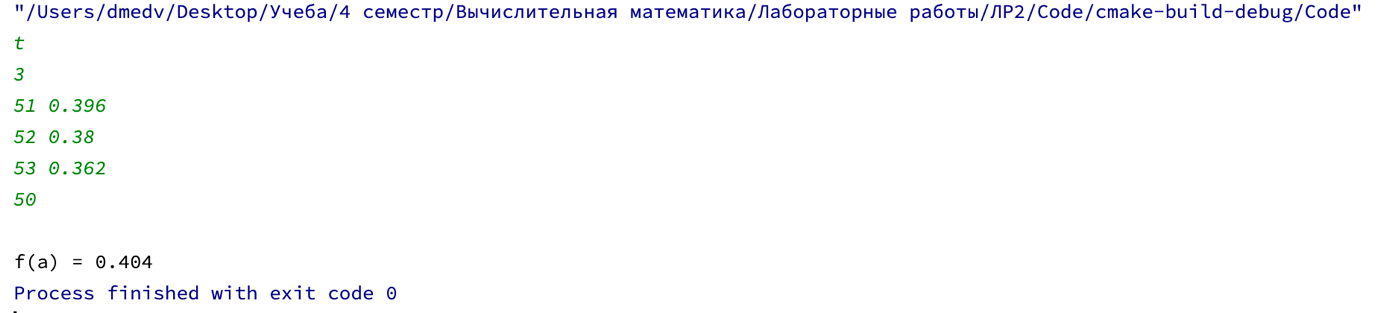
В программе выполняется проверка равномерности заданной интерполяционной сетки и в зависимости от результата используется метод конечных или разделенных разностей.

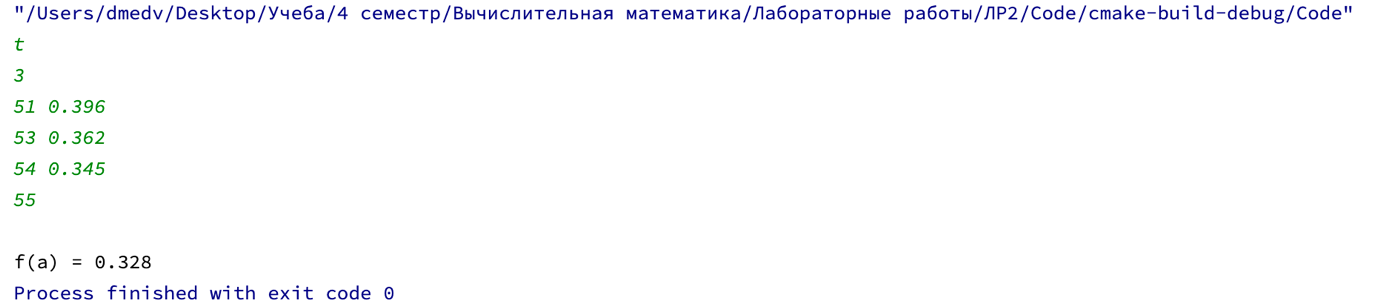
Результатом работы программы является таблица конечных или разделенных разностей и значение интерполяционного многочлена в произвольной точке x=a, удовлетворяющей условию: x0<a< xn.

Код программы:

#include **<iostream>**#include **<fstream>**#include **"iostream"**#include **"vector"**#define **EPS** 0.00001  
  
**using namespace** std;  
  
**typedef** pair<**float**, **float**> Point;  
  
*//Сравнение двух вещественных чисел***bool** fcmp(**float** f1, **float** f2) {  
 **return** (abs(f1 - f2) < **EPS**);  
}  
  
*//Проверка сетки на равномерность***bool** checkIfUniform(vector<Point> points) {  
 **for** (**int** i = 1; i <= points.size() - 2; i++) {  
 **if** (!fcmp(points[i].first - points[i - 1].first,  
 points[i + 1].first - points[i].first))  
 **return false**;  
 }  
  
 **return true**;  
}  
  
*//Получить вектор конечных разностей*vector<vector<**float**>> getFiniteDifferences(vector<Point> points) {  
 vector<vector<**float**>> finiteDifferences(points.size());  
 **for** (**int** i = 0; i < points.size(); i++) {  
 finiteDifferences[i] = vector<**float**>**{**points[i].second**}**;  
 }  
  
 **for** (**int** column = 1; column <= points.size() - 1; column++) {  
 **for** (**int** row = 0; row < points.size() - column; row++) {  
 finiteDifferences[row].push\_back(finiteDifferences[row + 1][column - 1] -  
 finiteDifferences[row][column - 1]);  
 }  
 }  
  
 **return** finiteDifferences;  
}  
  
*//Получить вектор разделенных разностей*vector<vector<**float**>> getDividedDifferences(vector<Point> points) {  
 vector<vector<**float**>> dividedDifferences(points.size());  
 **for** (**int** i = 0; i < points.size(); i++) {  
 dividedDifferences[i] = vector<**float**>**{**points[i].second**}**;  
 }  
  
 **for** (**int** column = 1; column <= points.size() - 1; column++) {  
 **for** (**int** row = 0; row < points.size() - column; row++) {  
 dividedDifferences[row].push\_back((dividedDifferences[row + 1][column - 1] -  
 dividedDifferences[row][column - 1]) /  
 (points[row + 1].first - points[row].first));  
 }  
 }  
  
 **return** dividedDifferences;  
}  
  
*//Получить многочлен N степени с вычитанием***float** getNPowerPolynomialNegative(**int** power, **float** value) {  
 **float** result = 1;  
 **for** (**int** i = 0; i < power; i++) {  
 result \*= (value - i);  
 }  
  
 **return** result;  
}  
  
*//Получить многочлен N степени с прибавлением***float** getNPowerPolynomialPositive(**int** power, **float** value) {  
 **float** result = 1;  
 **for** (**int** i = 0; i < power; i++) {  
 result \*= (value + i);  
 }  
  
 **return** result;  
}  
  
*//Получить многочлен для метода разделенных разностей***float** getPolynomialForDividedDifferences(**int** power, **float** value, vector<Point> points) {  
 **float** result = 1;  
 **for** (**int** i = 0; i < power; i++) {  
 result \*= value - points[i].first;  
 }  
  
 **return** result;  
}  
  
*//Найти значение факториала***int** factorial(**int** n) {  
 **int** result = 1;  
 **for** (**int** i = 2; i <= n; i++) {  
 result \*= i;  
 }  
  
 **return** result;  
}  
  
**int** main(**int** argc, **char** \*argv[]) {  
 *//Выбор способа ввода* **char** source;  
 cin >> source;  
  
 **float** substitutedValue; *// Подставляемое значение* vector<pair<**float**, **float**>> points;  
  
 *//Считывание из терминала* **if** (source == **'t'**) {  
 **int** numPoints;  
 cin >> numPoints;  
  
 **for** (**int** i = 0; i < numPoints; i++) {  
 Point point;  
 cin >> point.first >> point.second;  
 points.push\_back(point);  
 }  
 cin >> substitutedValue;  
  
 *//Считывание из файла* } **else if** (source == **'f'**) {  
 string filePath;  
 cin >> filePath;  
  
 ifstream inputFile(filePath);  
 **if** (inputFile.is\_open()) {  
 **int** numPoints;  
 inputFile >> numPoints;  
 **for** (**int** i = 0; i < numPoints; i++) {  
 Point point;  
 inputFile >> point.first >> point.second;  
 points.push\_back(point);  
 }  
 cin >> substitutedValue;  
  
 inputFile.close();  
 } **else** {  
 cerr << **"Can't open file"**;  
 exit(1);  
 }  
 } **else** {  
 cerr << **"Wrong input"**;  
 exit(1);  
 }  
  
 **if** (checkIfUniform(points)) {  
 vector<vector<**float**>> finiteDifferences = getFiniteDifferences(points);  
  
 **float** value = 0;  
 **if** (substitutedValue >= points[0].first) {  
 *//Интерполяция вперед* **float** t = (substitutedValue - points[0].first) / (points[1].first - points[0].first);  
  
 **for** (**int** i = 0; i < finiteDifferences[0].size(); i++) {  
 value += finiteDifferences[0][i] \* getNPowerPolynomialNegative(i, t) / factorial(i);  
 }  
  
 } **else** {  
 *//Интерполяция назад* **float** q = (substitutedValue - points[points.size() - 1].first) / (points[1].first - points[0].first);  
 **for** (**int** row = finiteDifferences.size() - 1; row >= 0; row--) {  
 **int** column = finiteDifferences[0].size() - 1 - row;  
 value += finiteDifferences[row][column] \*  
 getNPowerPolynomialPositive(column, q);  
 }  
 }  
  
 cout << **"\nf(a) = "** << value;  
  
 *//Метод разделенных разностей* } **else** {  
 vector<vector<**float**>> dividedDifferences = getDividedDifferences(points);  
  
 **float** value = 0;  
 **for** (**int** column = 0; column < dividedDifferences[0].size(); column++) {  
 value += dividedDifferences[0][column] \* getPolynomialForDividedDifferences(column, substitutedValue,  
 points);  
 }  
 cout << **"\nf(a) = "** << value;  
 }  
}

Равномерная сетка, интерполяция вперед:

Равномерная сетка, интерполяция назад:

Неравномерная сетка:

**Вывод:** в ходе лабораторной работы мы изучили понятие интерполяции, основные свойства интерполяционной функции, способы задания интерполяционной функции; понятие интерполяционного многочлена; изучили способы построения интерполяционного многочлена для случая равномерной и неравномерной сетки интерполяции; получили практические навыки решения задачи интерполяции с помощью ЭВМ.