МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ
АКАДЕМИКА С.П.КОРОЛЕВА»

Институт информатики, математики и электроники
Кафедра программных систем

Дисциплина
Вычислительные методы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту

Вариант 1

Выполнил:
Гижевская Валерия
Дмитриевна
6313-020302D

Проверил:
доцент кафедры ПС
Лобанков А.А.

Самара 2020 г.

Содержание

Введение

Под численным определением производных понимается приближенное вычисление производных по результатам вычислительного эксперимента. Оценка производных производится по результатам вычисления функции при заданных значениях аргумента. Численное дифференцирование является основой многих разделов вычислительной математики: численное решение обыкновенных дифференциальных уравнений и дифференциальных уравнений в частных производных, поиск экстремумов функций, решение систем нелинейных алгебраических уравнений и др.

В основе численного дифференцирования лежит аппроксимация функции, от которой берётся производная, интерполяционным многочленом. Все основные формулы численного дифференцирования могут быть получены при помощи первого интерполяционного многочлена Ньютона (формулы Ньютона для начала таблицы).

Основными задачами являются вычисление производной на краях таблицы и в её середине. Для равномерной сетки формулы численного дифференцирования «в начале таблицы» можно представить в общем виде следующим образом:

$$f'_i = \frac{1}{bh} \sum_j a_j f_{i+j} + \Delta(f),$$

где $\Delta(f)$ — погрешность формулы. Здесь коэффициенты a_j и b зависят от степени n использовавшегося интерполяционного многочлена, то есть от необходимой точности (скорости сходимости к точному значению при уменьшении шага сетки) формулы. Коэффициенты представлены в таблице

n	a_0	a_1	a_2	a_3	a_4	a_5	b
1	-1	1	0	0	0	0	1
2	-3	4	-1	0	0	0	2

3	-11	18	-9	2	0	0	6
4	-25	48	36	16	-3	0	12
5	-137	300	-300	200	-75	12	60

Погрешность вычислений

Погрешность вычисляется по формуле

$$\Delta(f) = (-1)^n \frac{f^{(n)}(\xi)}{n+1} h^n,$$

где h — шаг сетки, а точка ξ расположена где-то между i-ым и (i+n)-ым узлами. Примером может служить известная формула (n=2)

$$f'_i = \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2h} + \frac{f'''(\xi)}{3} h^2.$$

При n=1 формула может быть получена и из определения производной. Эта формула известна под названием формула дифференцирования вперёд.

Формулы «в конце таблицы» могут быть представлены в общем виде

$$f'_i = -\frac{1}{bh} \sum_j a_j f_{i-j} + \frac{f^{(n)}(\xi)}{n+1} h^n,$$

в которых коэффициенты a_j берутся из уже приведённой таблицы. В частности, при n=1 получается известная формула дифференцирования назад.

1. Описание и анализ вычислительных методов, постановка задачи

1.1. Описание и анализ вычислительных методов, близких к заданному

а) Методы односторонней разности

Производная функции $f(x)$ определяется выражением:

$$f'(x_0) = \frac{df}{dx} = \lim_{dx \to 0} \frac{f(x_0 + dx) - f(x_0)}{dx} \qquad (1.1)$$

Заменяя приращение *dx* на конечную величину Δ*x*, называемую шагом дифференцирования, получаем выражение:

$$f'(x_0) = \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \qquad (1.2)$$

Если дифференцируемая функция задана в виде непрерывной функции (рис.1.1), то для вычисления значения дифференциала необходимо получить значение функции $f(x)$ в точке x_0 и в точке $x_0 + \Delta x$. После чего можно вычислить значение производной функции $f'(x_0)$.

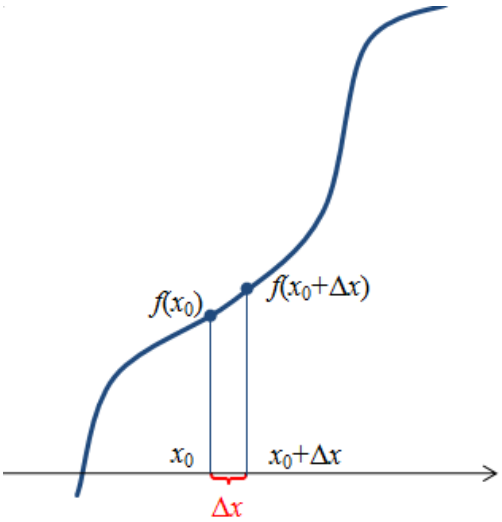
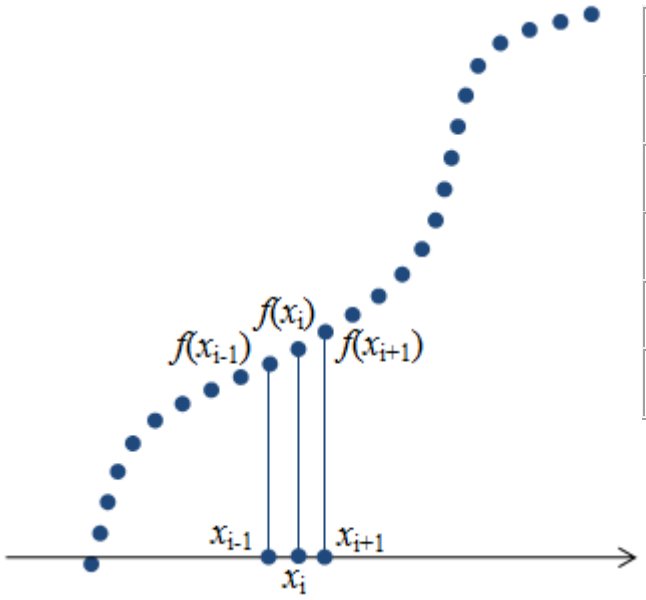


Рис.1.1.Непрерывная функция

Если функция задана выборкой, то есть набором значений функции в точках (рис.1.2), то выражение для численного дифференцирования (при условии, что *x* образуют возрастающую последовательность) можно переписать в виде:

$$f_i' = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} \qquad (1.3)$$



f_1	x_1
f_2	x_2
…	…
f_i	x_i
…	…
f_n	x_n

Рис.1.2. Дискретная функция

Как видно из этих выражений, значение производной в точке x_i оценивается по значению функции в этой и в следующей точке x_{i+1}. Такой способ можно условно назвать правосторонней разностью. Нетрудно записать выражение для левосторонней разности:

$$f'(x_0) = \frac{f(x_0) - f(x_0 - \Delta x)}{\Delta x} \qquad (1.4)$$

или

$$f_i' = \frac{f_i - f_{i-1}}{x_i - x_{i-1}} \qquad (1.5)$$

b) Метод двусторонней разности

С точки зрения точности методы левосторонней и правосторонней разностей равнозначны. Более точное значение дает метод двусторонней разности (что особенно справедливо для гладких функций). Теорема Лагранжа говорит о том, что уравнение:

$$f'(x_0) = \frac{f(b) - f(a)}{b - a} \qquad (1.6)$$

(при условии, что (a, b) – замкнутый промежуток, на котором функция $f(x)$ дифференцируема) имеет по меньшей мере один корень $x = \zeta$. Значение этого корня, вообще говоря, зависит от вида функции $f(x)$. Если она квадратичная, то уравнение первой степени и его корень лежит в точности на середине отрезка (a, b), то есть:

$$\zeta = \frac{b + a}{2} \qquad (1.7)$$

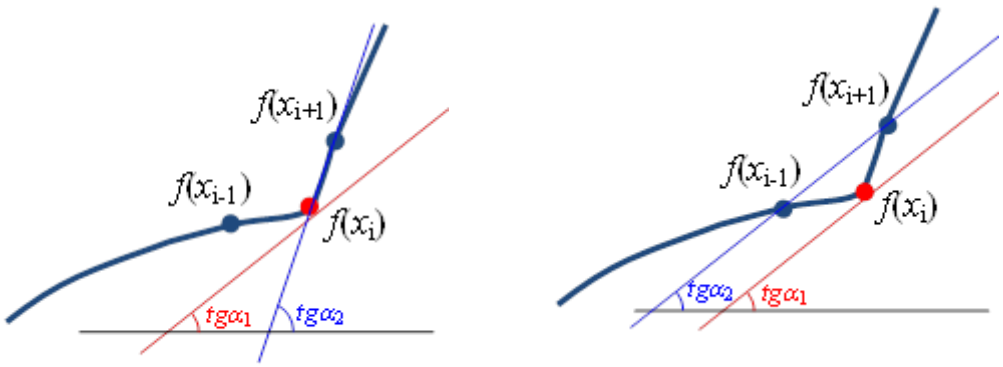
Если *a* имеет постоянное значение, а *b* стремится к *a*, то один из корней, как правило (за исключением случаев, когда вторая производная $f''(a)$ равна нулю или не существует), стремится к середине отрезка, то есть $\lim_{b \to a} \frac{\zeta - a}{b - a} = \frac{1}{2}$. Поэтому более точное приближение к искомому значению производной функции в точке x_0 можно получить, воспользовавшись формулами двусторонней разности:

$$f'(x_0) = \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2 \cdot \Delta x} \qquad (1.8)$$

или, для функций, заданных в виде выборки:

$$f_i' = \frac{f_{i+1} - f_{i-1}}{x_{i+1} - x_{i-1}} \qquad (1.9)$$

Наглядно сравнить одностороннюю и двустороннюю разности можно представив производную, как тангенс угла наклона касательной к функции в точке x_i. На рисунке 1.3 точное значение производной обозначено как $\mathrm{tg}\,\alpha_1$. В методе односторонней разности (рис.1.3, а) вместо касательной проводится прямая через точки x_i и x_{i+1}. Если в окрестностях точки x_i функция не гладкая, то значение производной ($\mathrm{tg}\,\alpha_2$) будет существенно отличаться от точного. В то время как в методе двусторонней разности, проведя прямую через точки x_{i-1} и x_{i+1} (рис.1.3, б), можно получить значение производной практически совпадающее с точным.



а) односторонняя разность *б) двусторонняя разность*

Рис.1.3. Графическое представление производной

c) Частное дифференцирование функции многих переменных

Отдельно следует отметить случай численного определения частных дифференциалов функций многих переменных. В этом случае все аргументы функции становятся константами кроме аргумента, по которому проводится дифференцирование, а требуемый порядок производной получается путем последовательного вычисления производных, вплоть до требуемого порядка:

$$\frac{df}{dx_i} = \frac{f(\ldots, x_i + \Delta x_i, \ldots) - f(\ldots, x_i, \ldots)}{\Delta x_i} \qquad (1.10)$$

d) Производные высоких порядков

При вычислении производных высоких порядков производная (n)-го порядка считается первой производной от (n-1)-го порядка. Так вторая производная функции является первой производной от первой производной:

$$f''(x) = (f'(x))' \text{ или } \frac{d^2 f}{dx^2} = \frac{d}{dx}\left(\frac{df}{dx}\right) \qquad (1.11)$$

Тогда выражение для вычисления производной примет вид:

$$\frac{d^2 f}{dx^2} = \frac{d}{dx}\left(\frac{df}{dx}\right) = \frac{f_1' - f'_{-1}}{2\Delta x} = \frac{\frac{f_2' - f'_0}{2\Delta x} - \frac{f_0' - f'_{-2}}{2\Delta x}}{2\Delta x} = \frac{f_2' - 2f_0' + f'_{-2}}{(2\Delta x)^2} \qquad (1.12)$$

1.2. Описание метода, реализуемого в проекте

a) Вычисление первой производной

Пусть на интервале $[a,b]$ задана непрерывная функция $f(x)$. Данная функция может быть задана в виде некоторого аналитического выражения или алгоритмически, то есть имеется возможность вычислять значения функции при заданном значении аргумента. Разобьем интервал точками $x_i = a + ih$, где $i = 0,1,...N$; $h = (b-a)/N$. В качестве приближенных выражений для первой производной для любой внутренней точки заданного интервала можно взять любую из следующих формул

$$f'(x_i) \approx (f_i - f_{i-1})/h, \qquad (1.13)$$

$$f'(x_i) \approx (f_{i+1} - f_i)/h, \qquad (1.14)$$

$$f'(x_i) \approx (f_{i+1} - f_{i-1})/2h, \qquad (1.15)$$

где $f_i = f(x_i)$, $f_{i-1} = f(x_{i-1})$, $f_{i+1} = f(x_{i+1})$.

Формулы (1.13), (1.14) и (1.15) называются соответственно левой, правой и

центральной разностными производными функции $f(x)$ в точке $x = x_i$. Если точка x_i фиксирована, а $h \to 0$ ($N \to \infty$), то каждое из выражений (1.13) - (1.15) в соответствии с определением первой производной стремится к точному значению производной $f'(x_i) \to f'(x_i)$. Поэтому в качестве приближенного выражения для оценки первой производной можно взять любую из этих формул.

b) Вычисление второй производной

Аналогично может быть произведено численное вычисление второй производной функции $f(x)$. Вычитая правые части выражений для правой и левой разностных производных (1.14) и (1.13) друг из друга и разделив на шаг h, получим

$$f''(x_i) \approx (f_{i+1} - 2f_i + f_{i-1}) / h^2 \quad . \qquad (1.16)$$

c) Вычисление третьей и четвёртой производных

Используя формулу вычисления второй производной нетрудно получить выражения для оценки третьей и четвертой производных функции $f(x)$ [2]

$$f'''(x_i) \approx (f_{i+2} - 2f_{i+1} + 2f_{i-1} - f_{i-2}) / 2h^3 \quad , \qquad (1.17)$$

$$f^{IV}(x_i) \approx (f_{i+2} - 4f_{i+1} + 6f_i - 4f_{i-1} + f_{i-2}) / h^4 \quad . \qquad (1.18)$$

1.3. Постановка задачи

Для заданной произвольной функции f(x) с помощью формул численного дифференцирования из ЛР №1 вычислить производные до 4-го порядка включительно. Предусмотреть в программе оценку погрешности. Привести тестовые примеры для всех производных, обеспечивающие вычисление производных с заданной относительной погрешностью. Вывести графики зависимости погрешности от шага дискретизации.

1.4. Описание используемых технологий

Для разработки программы был использован язык C# и среда разработки Visual Studio 2019. Информация о наиболее корректном применении возможностей языка бралась из источника [3].

2. Разработка программы

2.1. Структурная схема системы

Программа представляет собой клиент-серверное приложение. Связь между клиентской и серверной частями реализована с использованием протокола TCP.

Клиентская часть отвечает за пользовательский интерфейс, считывание входных данных и вывод графиков. Пользователь вводит функцию f(x), начальную точку x и пределы изменения шага дискретизации h. После всех вычислений программа выводит шаги дискретизации, производные 1,2,3 и 4 порядка, подсчитанные с помощью формул численного дифференцирования и графики зависимостей погрешностей от шага дискретизации.

На серверной части реализуются непосредственно вычисления. Вычисления самой функции, её производных (как с помощью формул численного дифференцирования, так и обычным способом) и погрешности вычисления производных. Формируются массивы из вычисленных значений и координат точек для графиков и передаются клиентской стороне для вывода результатов.

2.2. Разработка прототипа интерфейса пользователя

В прототипе пользовательского интерфейса предусмотрены поля для ввода начальных значений и поля для вывода конечных значений и графиков.

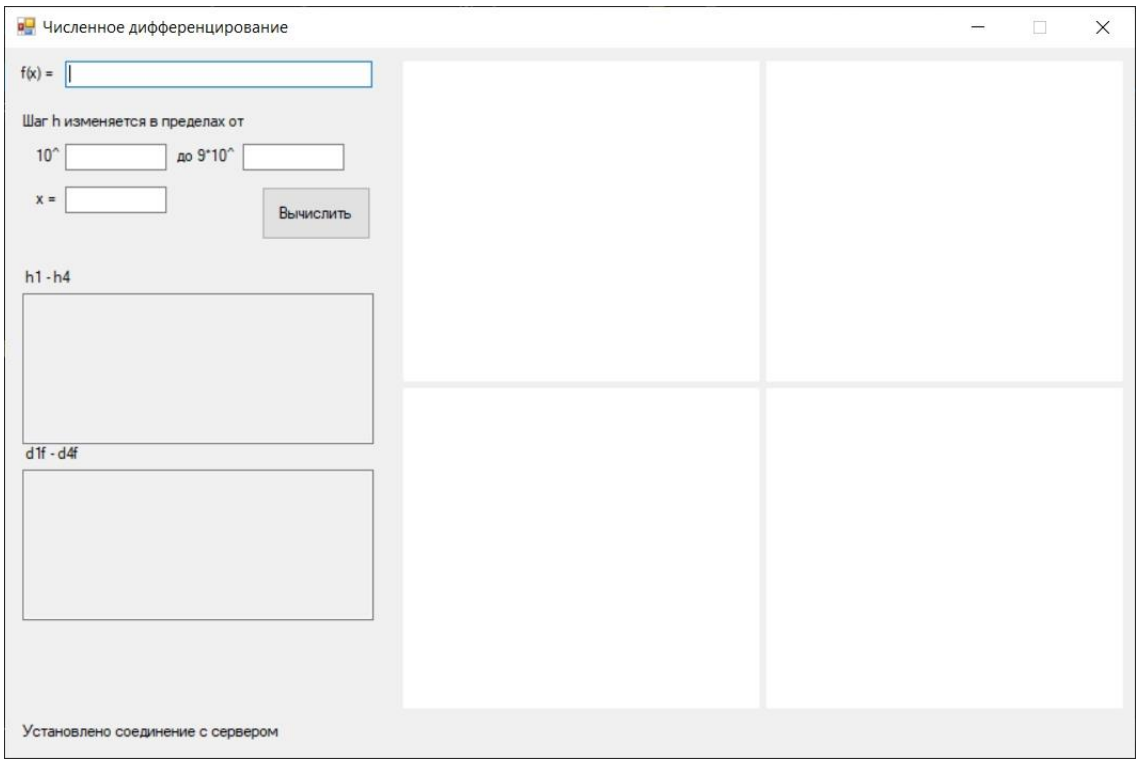


Рис.2.1. Прототип пользовательского интерфейса

2.3. Описание алгоритма вычислений

В процессе выполнения алгоритма производится поиск шага дискретизации, соответствующего наименьшей погрешности вычисления производной численным методом для каждого порядка производной (1-4). Далее полученные значения шагов дискретизации используются для вычисления значений производных по формулам численного дифференцирования. Схема работы алгоритма показана на рисунках 2.2 - 2.4.

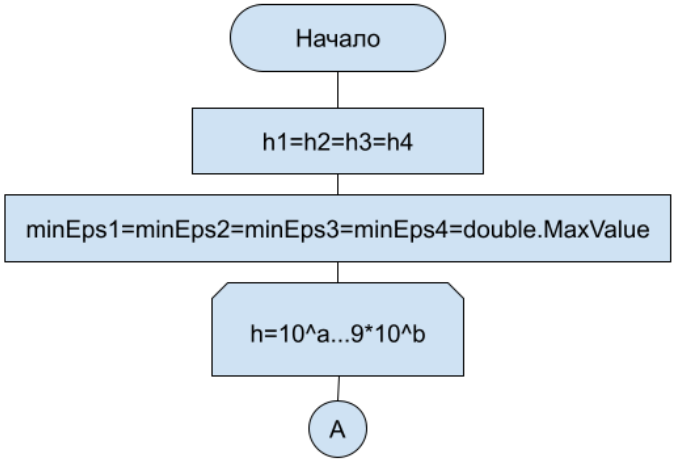


Рис.2.2. Блок-схема (начало)

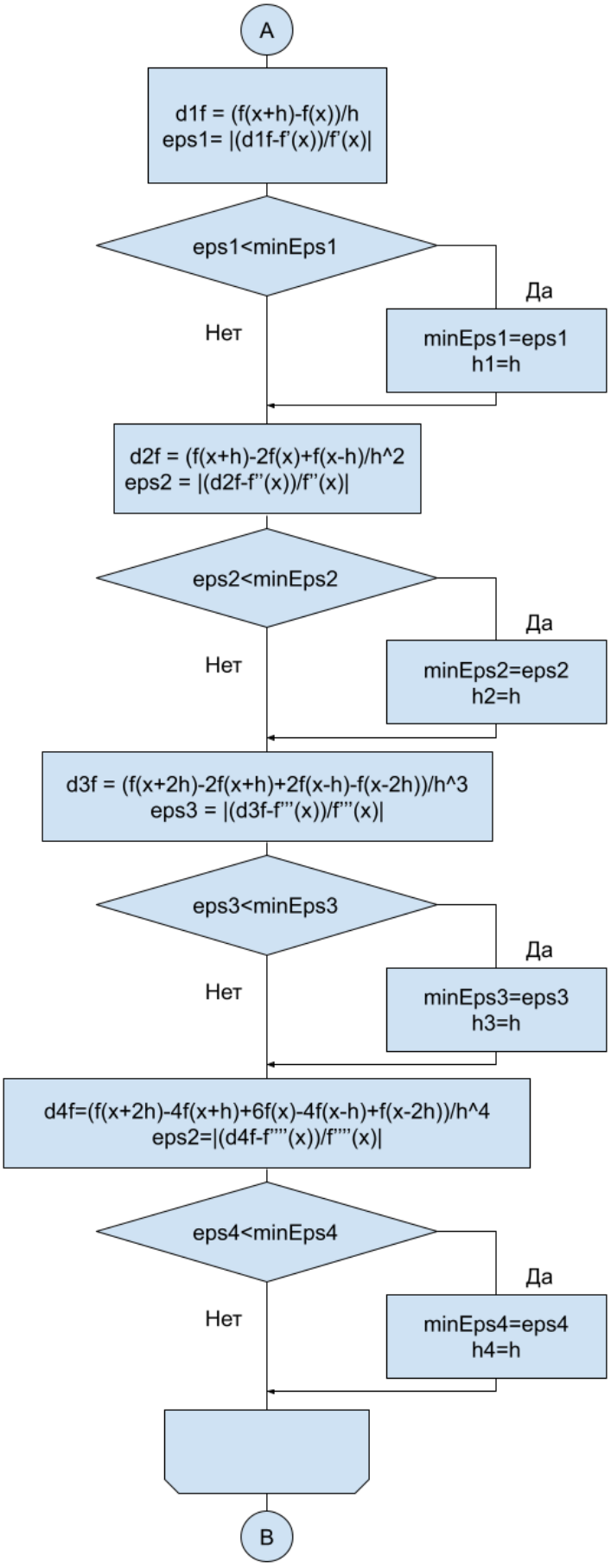


Рис.2.3. Блок-схема (продолжение)

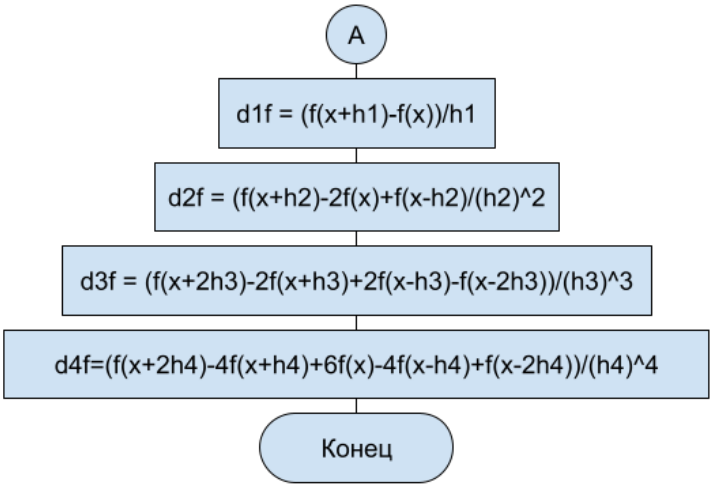


Рис.2.4. Блок-схема (конец)

3. Реализация системы

3.1. Описание интерфейса пользователя

Интерфейс пользователя реализован с помощью Windows Forms. Использовали структуру Point, которая представляет упорядоченную пару целых чисел — координат X и Y, определяющую точку на двумерной плоскости. Для ввода данных используются TextBox компоненты. Отображение графика функции выводится на Chart компоненте данных. Кнопка «Вычислить» запускает программу.

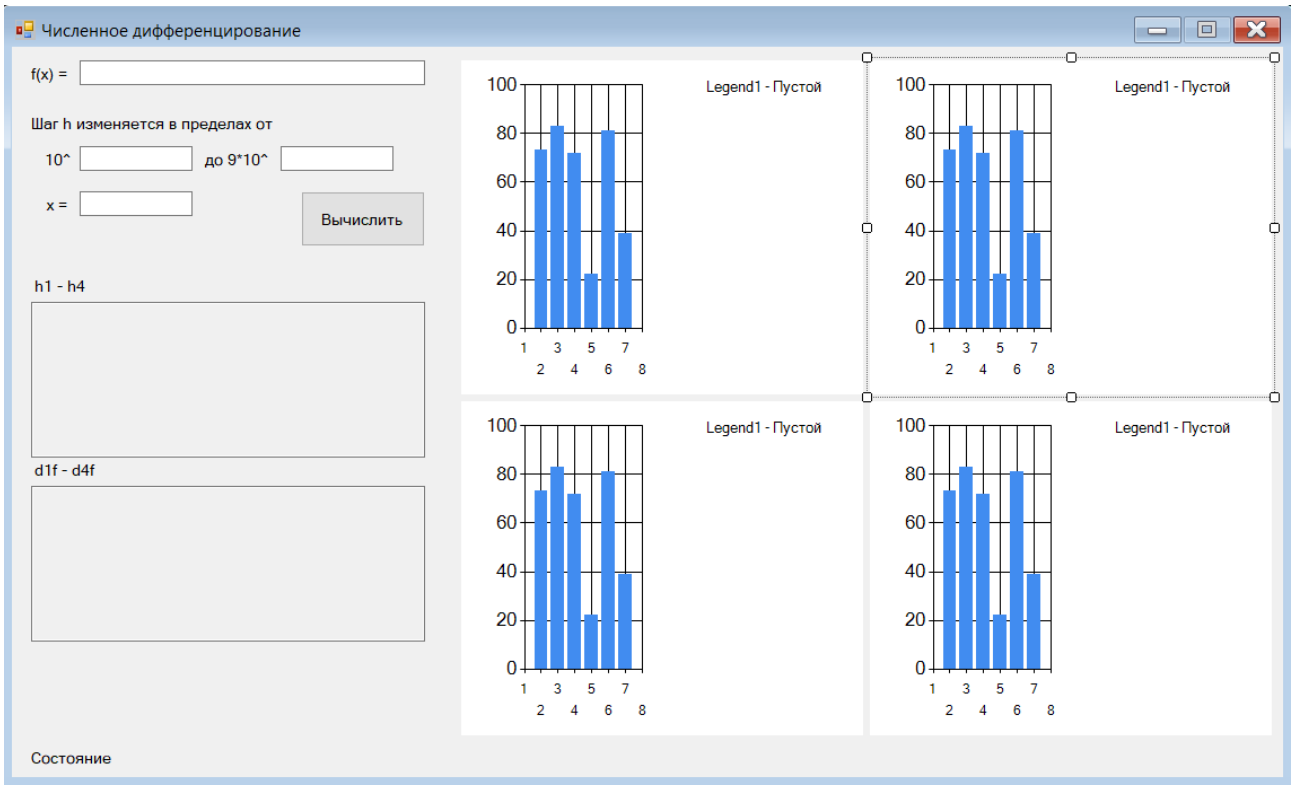


Рис.3.1. Пользовательский интерфейс

3.2. Описание тестовых примеров

a) Пример 1

Дана функция $f(x) = e^{\sin(x)} * \frac{\sin(x)}{x^2+3+\cos(x)}$

x = 1

h изменяется в пределах от 10^{-8} до $9*10^{-3}$

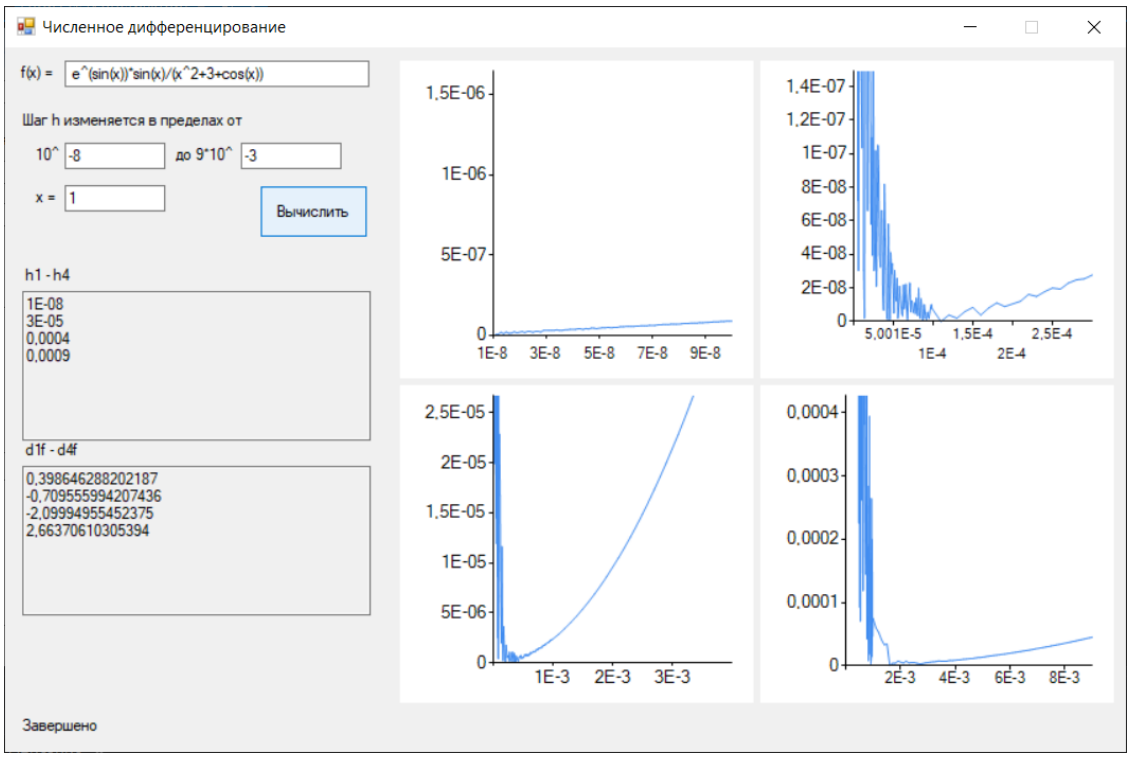


Рис.3.2. Результат работы программы для примера 1

b) Пример 2

Дана функция $f(x) = \frac{x(x+\sin(3x+2))}{x^2+x+0.5}$

x = 1

h изменяется в пределах от 10^{-8} до $9*10^{-3}$

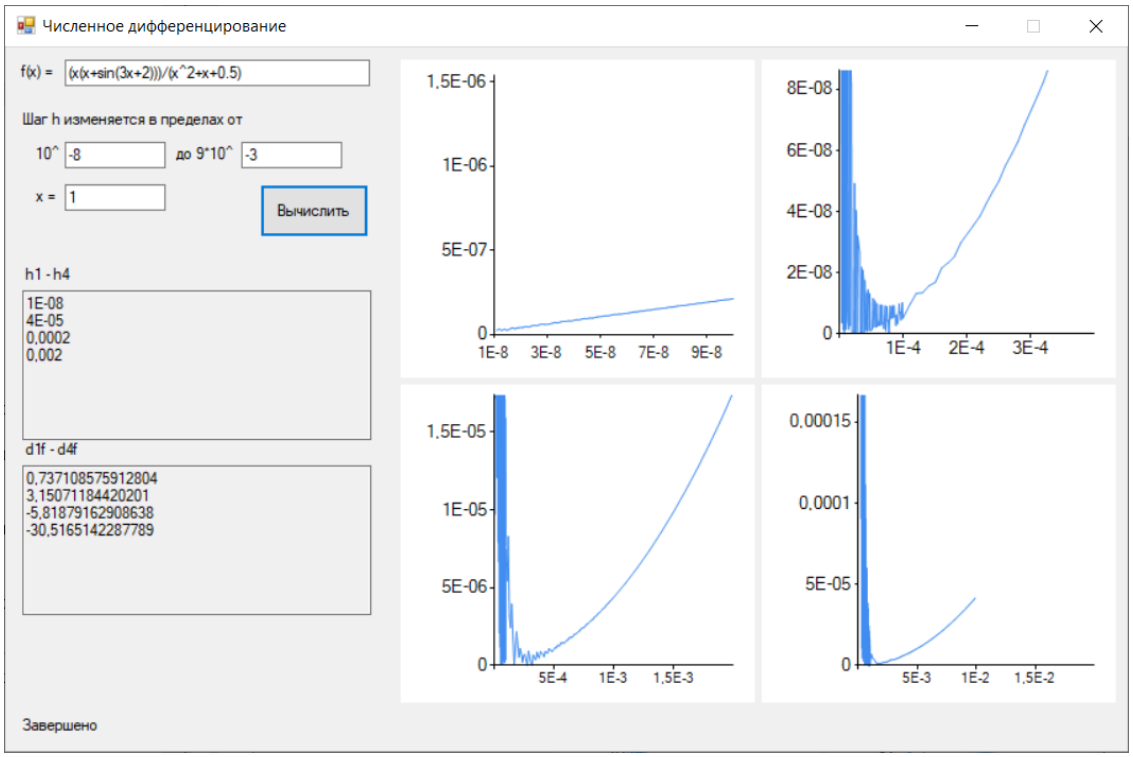


Рис.3.3. Результат работы программы для примера 2

Заключение

В ходе работы была разработана программа для численного интегрирования функций. Для достижения данной цели были изучены методы численного интегрирования функций, разработан пользовательский интерфейс, изучены возможности языка C#, произведена отладка работы программы на различных примерах.

Список использованных источников

1. Коварцев А.Н. Вычислительная математика: учебное пособие / А.Н. Коварцев.-Самара: ООО «Офорт», 2011.-230с.

2. Заболотнов Ю.М. Методические указания к лабораторным работам по вычислительным методам / Ю.М. Заболотнов. Кафедра программных систем. Самарский университет. 2020. 67 с. 17 илл.

3. Программирование на языке высокого уровня C#, Павловская Т.А., 2016

Приложение А Листинг программы

Клиентская часть:

```
using System;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Windows.Forms;
using System.Text.Json;
using System.Windows.Forms.DataVisualization.Charting;

namespace kursach_client
{
    public partial class Form1 : Form
    {
        TcpClient clientSocket = new TcpClient();

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            PrintStatus("Клиент запущен");
            try
            {
                clientSocket.Connect("127.0.0.1", 8888);
                PrintStatus("Установлено соединение с сервером");
            }
            catch (Exception exc)
            {
                PrintError(exc.Message);
            }
        }

        // Обработчик клика на кнопку "вычислить"
        private void Button1_Click(object sender, EventArgs e)
        {
            try
            {
                NetworkStream serverStream = clientSocket.GetStream();
                string functionString = function_textBox.Text;
                string x = x_textBox.Text;
                string a = a_textBox.Text;
                string b = b_textBox.Text;
                if (functionString == "" || x == "" || a == "" || b == "")
                {
                    PrintError("Имеются незаполненные поля.");
                    return;
                }
                PrintStatus("Выполняются вычисления. Пожалуйста, подождите...");
                string outString = string.Join("$$", new[] { functionString, x, a, b, "" });
                byte[] outStream = Encoding.ASCII.GetBytes(outString);
                serverStream.Write(outStream, 0, outStream.Length);

                serverStream.Flush();

                byte[] inStream = new byte[65536];
                serverStream.Read(inStream, 0, clientSocket.ReceiveBufferSize);
                string responseString = Encoding.ASCII.GetString(inStream);
```

```
                int stringEndIndex = responseString.IndexOf('\0');
                if (stringEndIndex >= 0)
                    responseString = responseString.Substring(0, stringEndIndex);

                ResponseData response;
                try
                {
                    response = JsonSerializer.Deserialize<ResponseData>(responseString);
                    Chart[] charts = new[] { chart1, chart2, chart3, chart4 };
                    for (int i = 0; i < charts.Length; i++)
                    {
                        Chart chart = charts[i];
                        chart.Series[0].Points.Clear();
                        chart.Series[0].ChartType = SeriesChartType.Line;
                        chart.ChartAreas[0].AxisX.MajorGrid.Enabled = false;
                        chart.ChartAreas[0].AxisY.MajorGrid.Enabled = false;
                        chart.ChartAreas[0].AxisX.Crossing = 0;
                        chart.ChartAreas[0].AxisY.Crossing = 0;
                        chart.ChartAreas[0].AxisX.Maximum = response.h[i] * 10;
                        chart.ChartAreas[0].AxisY.Maximum = response.e[i] * 100;
                        chart.ChartAreas[0].AxisX.LabelStyle.Format = "{0:0.###E-0}";
                    }
                    for (int i = 0; i < response.x.Length; ++i)
                    {
                        chart1.Series[0].Points.AddXY(response.x[i], response.y1[i]);
                        chart2.Series[0].Points.AddXY(response.x[i], response.y2[i]);
                        chart3.Series[0].Points.AddXY(response.x[i], response.y3[i]);
                        chart4.Series[0].Points.AddXY(response.x[i], response.y4[i]);
                    }
                    hResult_textBox.Lines = response.h.Select(Convert.ToString).ToArray();
                    derivativesResult_textBox.Lines =
response.d.Select(Convert.ToString).ToArray();
                    PrintStatus("Завершено");
                }
                catch
                {
                    PrintError(responseString);
                }
            }
            catch(Exception exc)
            {
                PrintError(exc.Message);
            }
        }

        // Вывод состояния операции
        public void PrintStatus(string message)
        {
            status_label.ForeColor = Color.Black;
            status_label.Text = message;
            status_label.Update();
        }

        // Вывод сообщения об ошибке
        public void PrintError(string message)
        {
            PrintStatus("Ошибка: " + message);
            status_label.ForeColor = Color.Red;
        }
    }

    // Вспомогательный класс для десериализации JSON объекта в объект C#
    public class ResponseData
    {
        public double[] h { get; set; }
```

```
        public double[] e { get; set; }
        public double[] d { get; set; }
        public double[] x { get; set; }
        public double[] y1 { get; set; }
        public double[] y2 { get; set; }
        public double[] y3 { get; set; }
        public double[] y4 { get; set; }
    }
}
```

Серверная часть:

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Net.Sockets;
using System.Text;
using System.Text.Json;
using System.Net;
using AngouriMath;

namespace Kursach_server
{
    class Program
    {
        static void Main(string[] args)
        {
            TcpListener serverSocket = new TcpListener(IPAddress.Parse("127.0.0.1"), 8888);
            TcpClient clientSocket;
            int counter;

            serverSocket.Start();
            Console.WriteLine(" >> " + "Server Started");

            counter = 0;
            while (true)
            {
                counter += 1;
                clientSocket = serverSocket.AcceptTcpClient();
                Console.WriteLine(" >> " + "Client No:" + Convert.ToString(counter) + "
started!");
                ClientHandler client = new ClientHandler();
                client.StartClient(clientSocket, Convert.ToString(counter));
            }
        }
    }

    // Класс методов вычислений (статический)
    public static class Calculation
    {
        // Преобразование AngouriMath.Entity в double
        public static double EntityToDouble(Entity function)
        {
            string strResult = function.ToString();
            int slashIndex = strResult.IndexOf('/');
            if (slashIndex == -1)
            {
                return double.Parse(strResult.Replace('.', ','));
            }
            string first = strResult.Substring(0, slashIndex);
            string second = strResult.Substring(slashIndex + 1);
            double result = double.Parse(first) / double.Parse(second);
            return result;
        }
```

```
        // Вычисление выражения
        public static double Eval(Entity function, double value)
        {
            return EntityToDouble(function.Substitute("x", value).Eval());
        }

        // Вычисление первой производной
        public static double D1f(Entity function, double x, double h)
        {
            return (Eval(function, x + h) - Eval(function, x)) / h;
        }

        // Вычисление второй производной
        public static double D2f(Entity function, double x, double h)
        {
            return (Eval(function, x + h) - 2 * Eval(function, x) + Eval(function, x - h)) /
Math.Pow(h, 2);
        }

        // Вычисление третьей производной
        public static double D3f(Entity function, double x, double h)
        {
            return (Eval(function, x + 2 * h) - 2 * Eval(function, x + h) + 2 *
Eval(function, x - h) - Eval(function, x - 2 * h)) / (2 * Math.Pow(h, 3));
        }

        // Вычисление четвертой производной
        public static double D4f(Entity function, double x, double h)
        {
            return (Eval(function, x + 2 * h) - 4 * Eval(function, x + h) + 6 *
Eval(function, x) - 4 * Eval(function, x - h) + Eval(function, x - 2 * h)) / Math.Pow(h, 4);
        }

        // Вычисление производной обычным способом (power - порядок)
        public static double Df(Entity function, double x, int power)
        {
            Entity derivative = function.Derive("x", power);
            return EntityToDouble(derivative.Substitute("x", x).Eval());
        }

        // Определение погрешности вычисления производной
        public static double EpsDf(Entity function, double x, double h, int power, double
df)
        {
            Func<Entity, double, double, double> func;
            switch (power)
            {
                case 1:
                    func = D1f;
                    break;
                case 2:
                    func = D2f;
                    break;
                case 3:
                    func = D3f;
                    break;
                case 4:
                    func = D4f;
                    break;
                default:
                    func = D1f;
                    break;
            }
            return Math.Abs((func(function, x, h) - df) / df);
```

```
        }
    }

    // Класс - обработчик приложения-клиента
    public class ClientHandler
    {
        TcpClient clientSocket;
        string clNo;

        // Инициализация соединения с клиентом
        public void StartClient(TcpClient inClientSocket, string clineNo)
        {
            clientSocket = inClientSocket;
            clNo = clineNo;
            Thread ctThread = new Thread(Process);
            ctThread.Start();
        }

        // Подключение и обработка данных (основной метод)
        private void Process()
        {
            byte[] bytesFrom = new byte[65536];
            int requestCount = 0;

            while (true)
            {
                try
                {
                    requestCount++;
                    NetworkStream networkStream = clientSocket.GetStream();
                    Console.WriteLine(" >> ReceiveBufferSize: " +
clientSocket.ReceiveBufferSize);
                    networkStream.Read(bytesFrom, 0, clientSocket.ReceiveBufferSize);
                    string dataFromClient = Encoding.ASCII.GetString(bytesFrom);
                    string[] values = dataFromClient.Split(new[] { "$$" },
StringSplitOptions.RemoveEmptyEntries);
                    string function = values[0];
                    double x = double.Parse(values[1]);
                    int a = int.Parse(values[2]);
                    int b = int.Parse(values[3]);
                    Console.WriteLine(" >> " + "From client-" + clNo + ": " + function + ",
" + x + ", " + a + ", " + b);

                    string json;

                    try
                    {
                        Entity fn = MathS.FromString(function);
                        double df1 = Calculation.Df(fn, x, 1);
                        double df2 = Calculation.Df(fn, x, 2);
                        double df3 = Calculation.Df(fn, x, 3);
                        double df4 = Calculation.Df(fn, x, 4);
                        double h1 = 0, h2 = 0, h3 = 0, h4 = 0;
                        double minEps1 = double.MaxValue, minEps2 = minEps1, minEps3 =
minEps1, minEps4 = minEps1;
                        List<double> _h = new List<double>();
                        List<double> graph1 = new List<double>();
                        List<double> graph2 = new List<double>();
                        List<double> graph3 = new List<double>();
                        List<double> graph4 = new List<double>();
                        int hMinPower = a, hMaxPower = b;
                        for (int hPower = hMinPower; hPower <= hMaxPower; hPower++)
                        {
                            for (int i = 1; i < 10; i++)
                            {
```

```
                                double h = i * Math.Pow(10, hPower);
                                double eps1 = Calculation.EpsDf(fn, x, h, 1, df1);
                                if (eps1 < minEps1)
                                {
                                    minEps1 = eps1;
                                    h1 = h;
                                }
                                double eps2 = Calculation.EpsDf(fn, x, h, 2, df2);
                                if (eps2 < minEps2)
                                {
                                    minEps2 = eps2;
                                    h2 = h;
                                }
                                double eps3 = Calculation.EpsDf(fn, x, h, 3, df3);
                                if (eps3 < minEps3)
                                {
                                    minEps3 = eps3;
                                    h3 = h;
                                }
                                double eps4 = Calculation.EpsDf(fn, x, h, 4, df4);
                                if (eps4 < minEps4)
                                {
                                    minEps4 = eps4;
                                    h4 = h;
                                }
                                for (int j = 1; j < 10; j++)
                                {
                                    double hTmp = h + 0.1 * j * Math.Pow(10, hPower);
                                    _h.Add(hTmp);
                                    graph1.Add(Calculation.EpsDf(fn, x, hTmp, 1, df1));
                                    graph2.Add(Calculation.EpsDf(fn, x, hTmp, 2, df2));
                                    graph3.Add(Calculation.EpsDf(fn, x, hTmp, 3, df3));
                                    graph4.Add(Calculation.EpsDf(fn, x, hTmp, 4, df4));
                                }
                            }
                        }
                        double[] hArray = new[] { h1, h2, h3, h4 };
                        double[] eArray = new[] { minEps1, minEps2, minEps3, minEps4 };
                        double[] derivatives = new[] { Calculation.D1f(fn, x, h1),
Calculation.D2f(fn, x, h2), Calculation.D3f(fn, x, h3), Calculation.D4f(fn, x, h4) }; //
вычисляем производные с заданным шагом дискретизации с помощью формул численного
дифференцирования

                        json = "{\"h\":" + JsonSerializer.Serialize(hArray) + ",";
                        json += "\"d\":" + JsonSerializer.Serialize(derivatives) + ",";
                        json += "\"e\":" + JsonSerializer.Serialize(eArray) + ",";
                        json += "\"x\":" + JsonSerializer.Serialize(_h) + ",";
                        json += "\"y1\":" + JsonSerializer.Serialize(graph1.ToArray()) +
",";
                        json += "\"y2\":" + JsonSerializer.Serialize(graph2.ToArray()) +
",";
                        json += "\"y3\":" + JsonSerializer.Serialize(graph3.ToArray()) +
",";
                        json += "\"y4\":" + JsonSerializer.Serialize(graph4.ToArray());
                        json += "}";

                    }
                    catch (Exception e)
                    {
                        json = e.Message;
                    }
                    byte[] sendBytes = Encoding.ASCII.GetBytes(json);
                    networkStream.Write(sendBytes, 0, sendBytes.Length);
                    networkStream.Flush();
                }
```

```
                catch (Exception ex)
                {
                    Console.WriteLine(" >> " + ex.ToString());
                }
            }
        }
    }
}
```