Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ-7)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Сравнение эффективности алгоритмов вычисления интегралов методами Гаусса и Ньютона- Котеса на примере решения обратной задачи нахождения аргумента по заданному значению функции Лапласа

Студент ИУ7-78 _____ А.В. Иванников
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ В.М. Градов
(Подпись, дата) (И.О.Фамилия)

Консультант _____ В.М. Градов
(Подпись, дата) (И.О.Фамилия)

2020 г.

Содержание

Введение

Данная курсовая работа посвящена изучению методов численного интегрирования и сравнению эффективности алгоритмов вычисления интегралов на основе этих методов.

Целью курсовой работы является сравнение эффективности алгоритмов вычисления интегралов методами Гаусса и Ньютона-Котеса на примере на примере решения обратной задачи нахождения аргумента по заданному значению функции Лапласа.

Результатом работы является приложение, иллюстрирующее сравнение эффективности алгоритмов вычисления интегралов методами Гаусса и Ньютона-Котеса.

В аналитическом разделе курсовой будет выполнен анализ предметной области решаемой задачи, изложена суть методов Гаусса и Ньютона-Котеса.

В технологическом разделе курсовой работы будет выполнен выбор инструментов разработки приложения, описание процесса его работы.

В заключении приводятся общие выводы о проделанной работе.

В приложении приведены листинг программного кода приложения.

1. Аналитический раздел

1.1 Анализ предметной области решаемой задачи

Аналитическое решение задач вычисления определенного интеграла существует, как правило, только для достаточно ограниченного числа подынтегральных функций *f(x).* В этом случае первообразную можно представить в виде комбинации алгебраических и трансцендентных функций.

Достаточно часто первообразную *F(x)* невозможно выразить через элементарные функции. Кроме этого функция *f(x)* может задаваться не в виде непрерывной функции, а в виде таблицы ее значений на фиксированном конечном множестве точек. В этом случае понятие первообразной теряет смысл, поэтому для вычисления интеграла применяют численные методы.

Задача численного интегрирования состоит в нахождении приближенного значения определенного интеграла с помощью некоторой приближенной формулы через известные значения подынтегральной функции (иногда через значения ее производных) в заданных точках. Методы вычисления однократных интегралов называются квадратурными, кратных интегралов – кубатурными.

В рамках такого подхода искомый интеграл заменяется линейной комбинацией (линейной функцией) значений подынтегральной функции $f(x_i)$ в $(k+1)$ точках интервала $[a, b]$

$$\int_a^b f(x)dx \approx \sum_{i=1}^{k} A_i f(x_i) + R, \text{ где}$$

A_i – весовые коэффициенты,

x_i – узловые точки,

R – погрешность квадратурной формулы.

Принцип построения квадратурных формул состоит в следующем. Подынтегральная функция *f(x)* на интервале *[a, b]* заменяется интерполирующей функцией достаточно простого вида (например, интерполяционным многочленом), от которой легко находится интеграл. Исходный интервал *[a, b]* разделяется на *(k–1)* интервалов с шагом

$$h = \frac{b-a}{k-1}.$$

На каждом из полученных интервалов $[x_i, x_{i+1}]$ строится интерполяционный многочлен. Искомый интеграл вычисляется как сумма k частичных интегралов:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{k} \int_{x_i}^{x_{i+1}} \varphi(x)dx$$

1.2 Метод Гаусса.

Метод Гаусса не предполагает разбиения отрезка интегрирования на равные промежутки. Формулы численного интегрирования интерполяционного типа ищутся таким образом, чтобы они обладали наивысшим порядком точности при заданном числе узлов. Узлы и коэффициенты формул численного интегрирования находятся из условий обращения в нуль их остаточных членов для всех многочленов максимально высокой степени.

Идея метода Гаусса (Гаусса-Лежандра) заключается в следующем. В методе трапеций погрешности аппроксимации суммируются для каждого интервала, причем при заданном шаге уменьшить погрешность невозможно, поскольку краевые точки интервала жестко заданы (рисунок 1а).

Напротив, в методе высокоточных квадратур функция на искомом интервале аппроксимируется полиномом, который в зависимости от его параметров пересекает искомую функцию в нескольких точках (например, в двух в случае квадратичной функции), и эти точки возможно подобрать таким образом, чтобы скомпенсировать наилучшим образом погрешности аппроксимации с разными знаками (рисунок 1б).

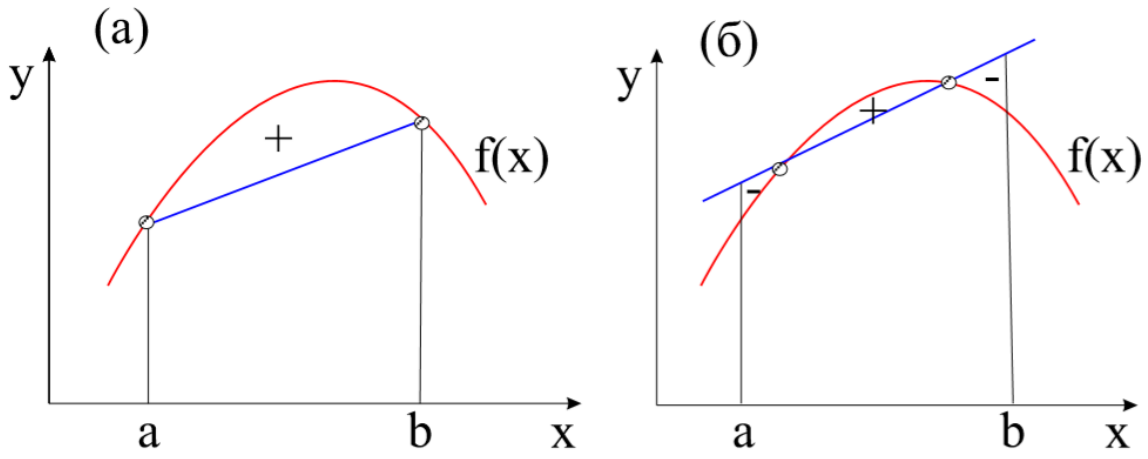

Рисунок 1 - Аппроксимация подынтегральной функции *f(x)* с помощью (а) кусочно-линейной функции на равномерной сетке (метод трапеций) и (б) полиномиальной интерполирующей функции на неравномерной сетке (метод квадратур). Знаками “+” и “–” отмечены погрешности аппроксимации.

Из рисунка видно, что в методе квадратур погрешности могут быть скомпенсированы за счет выбора оптимальных узлов.

Далее значение интеграла вычисляется на рассматриваемом интервале как сумма значений функции в данных точках с соответствующими весами. При этом высокая точность расчетов достигается уже для аппроксимирующих полиномов невысокой степени (2 – 6). В квадратурах Гаусса–Лежандра изначально рассматривается задача вычисления интеграла

$$I = \int_{-1}^{1} f(x)dx \approx I_G = \sum_{i=0}^{n-1} w_i f(x_i)$$

на отрезке [−1,1], гдеw_i— веса, с которыми берутся значения функции в точках x_i. Положения точек и веса функций вычисляются точно для квадратур любого порядка. Однако на практике редко используются квадратуры выше десятого порядка.

Для перехода от вычисления интеграла в интервале интегрирования [−1,1] к вычислению интеграла в произвольном интервале необходимо ввести замену переменных

$$\xi = \frac{1}{2}\,(b+a) + \frac{1}{2}\,(b-a)x,$$

при которой

$$\int_a^b f(\xi)\,d\xi = \left(\frac{b-a}{2}\right)\int_{-1}^{1} f(\xi(x))\,dx.$$

Окончательно получаем квадратуру Гаусса-Лежандра для произвольного интервала интегрирования:

$$I = \int_a^b f(\xi)\,d\xi \approx I_G = \left(\frac{b-a}{2}\right)\sum_{i=0}^{n-1} w_i f(\xi_i),$$

где

$$\xi_i = \left(\frac{b+a}{2}\right) + \left(\frac{b-a}{2}\right)x_i.$$

Квадратуры Гаусса-Лежандра требуют минимум вычислений для достижения заданной точности интегрирования.

Достоинства метода Гаусса:

- прост в понимании и организации вычислительного процесса;

- оценка погрешности легко определяется;

- разбиение отрезка интегрирования на n равных интервалов в отличие от других квадратурных формул, в которых абсциссы x_i подбираются, исходя из соображений точности и, вообще говоря, являются иррациональными числами;

- значительно большая точность результата для функций высокой гладкости при одинаковом числе узлов.

1.3 Метод Ньютона - Котеса

Среди интерполяционных формул ранее всех были построены широко известные и часто применяемые формулы Ньютона — Котеса. Они относятся к случаю постоянного веса и конечного отрезка интегрирования. Для иллюстрации идеи метода рассмотрим интеграл

$$\int_a^b f(x)dx$$

Отрезок *[a, b]* разделим на n одинаковых частей $h = \frac{b-a}{n}$. Построим интерполяционную квадратурную формулу с узлами *a, a+h, a+2h, ... , a + nh = b*. Чтобы нахождение ее коэффициентов привести к вычислению величин, не зависящих от промежутка *[a, b]*, необходимо записать формулу в виде

$$\int_a^b f(x)dx = (b-a)\sum_{i=1}^{k}\int_{k=0}^{n} B_k^n f(a+kh).$$

Согласно формуле

$$A_k = \int_a^b p(x) \frac{\omega(x)}{(x - x_k)\,\omega'(x_k)}\,dx$$

величины $B_k^n = (b-a)^{-1} A_k$ будут иметь значения

$$B_k^n = (b-a)^{-1} \int_a^b \frac{\omega(x)}{(x - a - kh)\,\omega'(a + kh)}\,dx,$$

где $w(x)=(x - a)(x – a - h)…(x – a - nh)$.

Если ввести новую переменную t, положив $x= a + th$, то будет

$$\omega(x) = h^{n+1} t\,(t-1)\,(t-2)\,\ldots\,(t-n),\quad x - a - kh = h\,(t-k),$$
$$\omega'(a + kh) = (-1)^{n-k} h^n k!\,(n-k)!$$

В итоге

$$B_k^n = \frac{(-1)^{n-k}}{nk!\,(n-k)!} \int_0^n t\,(t-1)\,\ldots\,(t-k+1)\,(t-k-1)\,\ldots\,\ldots\,(t-n)\,dt$$

Котесом были получены коэффициенты B_k^n для n от 1 до 10:

$$n=1;\quad B_0^1=B_1^1=\frac{1}{2};$$

$$n=2;\quad B_0^2=B_2^2=\frac{1}{6},\quad B_1^2=\frac{4}{6};$$

$$n=3;\quad B_0^3=B_3^3=\frac{1}{8},\quad B_1^3=B_2^3=\frac{3}{8};$$

$$n=4;\quad B_0^4=B_4^4=\frac{7}{90},\quad B_1^4=B_3^4=\frac{32}{90},\quad B_2^4=\frac{12}{90};$$

$$n=5;\quad B_0^5=B_5^5=\frac{19}{288},\quad B_1^5=B_4^5=\frac{75}{288},\quad B_2^5=B_3^5=\frac{50}{288};$$

$$n=6;\quad B_0^6=B_6^6=\frac{41}{840},\quad B_1^6=B_5^6=\frac{216}{840},\quad B_2^6=B_4^6=\frac{27}{840};$$

$$B_3^6=\frac{272}{840};$$

$$n=7;\quad B_0^7=B_7^7=\frac{751}{17280},\quad B_1^7=B_6^7=\frac{3577}{17280},\quad B_2^7=B_5^7=\frac{1323}{17280},$$

$$B_3^7=B_4^7=\frac{2989}{17280};$$

$$n=8;\quad B_0^8=B_8^8=\frac{989}{28350},\quad B_1^8=B_7^8=\frac{5888}{28350},\quad B_2^8=B_6^8=\frac{-928}{28350},$$

$$B_3^8=B_5^8=\frac{10496}{28350},\quad B_4^8=\frac{-4540}{28350};$$

$$n=9;\quad B_0^9=B_9^9=\frac{2857}{89600},\quad B_1^9=B_8^9=\frac{15741}{89600},\quad B_2^9=B_7^9=\frac{1080}{89600},$$

$$B_3^9=B_6^9=\frac{19344}{89600}.\quad B_4^9=B_5^9=\frac{5778}{89600};$$

$$n=10;\quad B_0^{10}=B_{10}^{10}=\frac{16067}{598752},\quad B_1^{10}=B_9^{10}=\frac{106300}{598752},\quad B_2^{10}=B_8^{10}=\frac{-48525}{598752},$$

$$B_3^{10}=B_7^{10}=\frac{272400}{598752},\quad B_4^{10}=B_6^{10}=\frac{-260550}{598752},\quad B_5^{10}=\frac{427368}{598752}.$$

1.4 Функция Лапласа

Функция Лапласа имеет вид

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{\frac{-t^2}{2}}\, dt$$

Значения функции при различных значениях x приводятся в специальных таблицах.

График функции Лапласа приведен на рисунке 2.

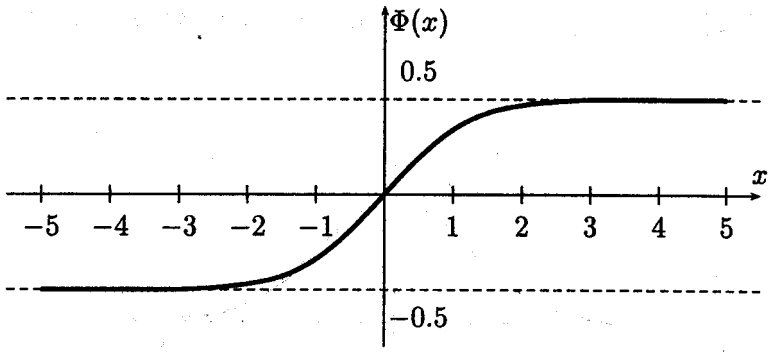

Рисунок 2 – График функции Лапласа

2. Технологический раздел

2.1 Выбор инструментов разработки приложения

В качестве языка программирования для написания приложения был выбран язык C#.

Язык C# — объектно-ориентированный язык программирования. Относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML. Переняв многое от своих предшественников (языков C++, Java, Delphi, Smalltalk) C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем: так, C# не поддерживает множественное наследование классов (в отличие от C++).

В качестве среды разработки приложения была выбрана Visual Studio 2017.

Данная среда обладает рядом достоинств:

— поддержка множества языков при разработке;

— интуитивный стиль кодирования;

— более высокая скорость разработки;

— возможности отладки.

В качестве недостатка можно отметить невозможность отладчика (Microsoft Visual Studio Debugger) отслеживать в коде режима ядра. Отладка в Windows в режиме ядра в общем случае выполняется при использовании Win Dbg, KD или Soft ICE.

2.2 Интерфейс приложения

В качестве входных данных в приложении используются:

- степень (n);

- значение функции Лапласа Ф(x).

Как результат работы приложение должно предоставлять пользователю:

- график функции Лапласа;
- таблицу из двух строк со следующими столбцами:
    - метод;
    - количество итераций;
    - количество тиков;
    - результат вычисления;
- гистограмму.

Рабочее окно приложения приведено на рисунке 3.

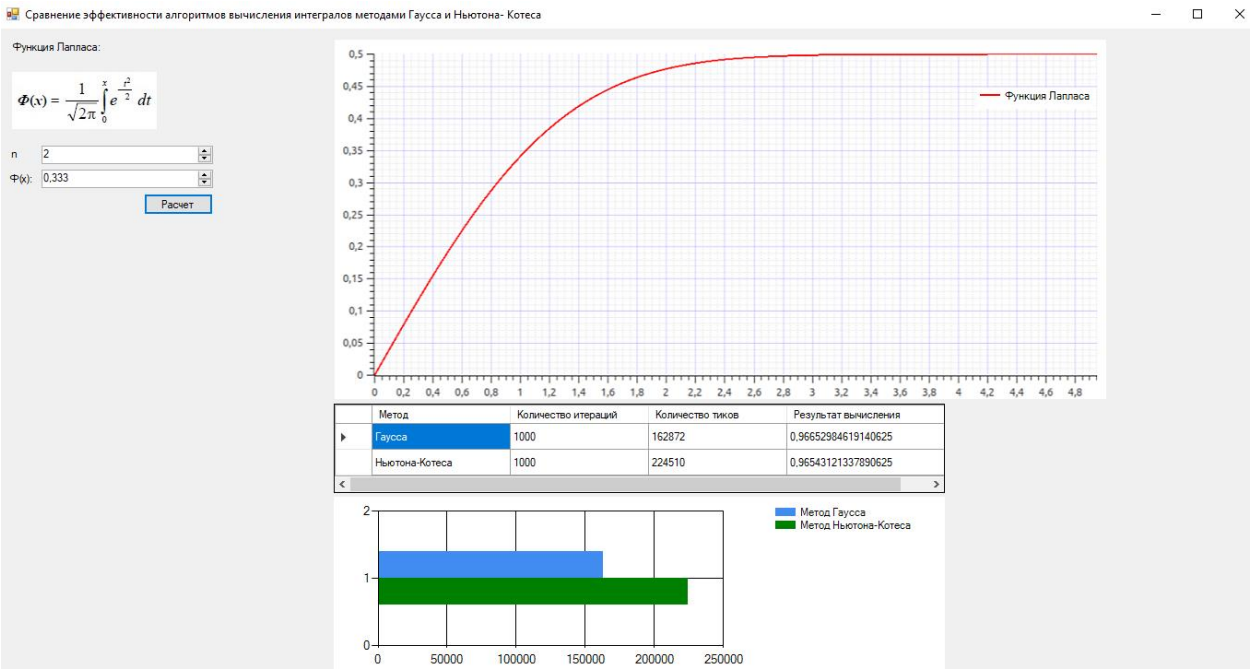

Рисунок 3 – Рабочее окно приложения

Результат работы приложения при n = 2 и Φ(x) = 0,333 приведен в таблице 1.

Таблица 1 – Результат работы приложения при n = 2 и Φ(x) = 0,333

Метод	Количество итераций	Количество тиков	Результат вычисления
Гаусса	1000	249247	0,96652984619140625
Ньютона - Котеса	1000	237967	0,96543121337890625

Заключение

В ходе работы, проведенной в рамках курсовой работы, были изучены два метода численного интегрирования: метод Гаусса и метод Ньютона-Котеса. В результате было разработано приложение, позволяющее сравнить эффективность алгоритмов вычисления интегралов на основе этих методов на примере решения обратной задачи нахождения аргумента по заданному значению функции Лапласа.

По полученным в результате расчета в разработанном приложении данным можно сделать вывод о том, что алгоритм Ньютона-Котеса работает быстрее.

Список использованных источников

1. В.М. Градов. Моделирование: лекции.
2. В.М. Градов. Компьютерные технологии в практике математического моделирования: учебное пособие. Издательство МГТУ им. Н.Э. Баумана, 2006.
3. В.М. Градов, Г.В. Овечкин, П.В. Овечкин, И.В Рудаков. Компьютерное моделирование: учебник. Издательство Инфра-М, Курс, 2017.

Приложение

Листинг программного кода приложения

```
//Shell.cs

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFrmShell
{
    public partial class Shell : Form
    {
        Lib.IIntegrals ig;
        Lib.IIntegrals igl;
        Lib.IIntegrals ink;
        Lib.Dichotomy dich;

        List<Model.Result> results;

        uint n = 2;
        double y=0.333;

        public Shell()
        {
            InitializeComponent();
            results = new List<Model.Result>();
            this.bindingSource.DataSource = results;

            ig = new Lib.IntagralGauss();
            igl = new Lib.IntegralGaussLegandr();
            ink = new Lib.NewtonKotes();
            dich = new Lib.Dichotomy();

            this.numericUpDownN.Value = n;
            this.numericUpDownY.Value =(decimal) y;
        }
```

```
        private void btStart_Click(object sender, EventArgs e)
        {
            n = (uint)this.numericUpDownN.Value;
            y = (double)this.numericUpDownY.Value;

            results.Clear();

            System.Diagnostics.Stopwatch sw = new System.Diagnostics.Stopwatch();

            this.GetPointGrafic();

            sw.Start();
            Model.Result r1 = new Model.Result();
            r1.MethodName = "Гаусса";
            for (int i = 0; i < 1000; i++)
            {
                r1.Integral = dich.find(0, 4, n, y, igl.GetIntegral);
                r1.Iterations++;
            }
            sw.Stop();
            r1.Ticks = sw.ElapsedTicks;
            results.Add(r1);

            Model.Result r2 = new Model.Result();
            r2.MethodName = "Ньютона-Котеса";
            System.Diagnostics.Stopwatch sw1 = new System.Diagnostics.Stopwatch();
            sw1.Restart();
            for (int i = 0; i < 1000; i++)
            {
              r2.Integral = dich.find(0, 4, n, y, ink.GetIntegral);
              r2.Iterations++;
            }
            sw1.Stop();
            r2.Ticks = sw1.ElapsedTicks;
            results.Add(r2);

            this.bindingSource.ResetBindings(false);
            this.chart1.Series[0].Points.Add(r1.Ticks);
            this.chart1.Series[1].Points.Add(r2.Ticks);
        }

        private void GetPointGrafic()
        {

           double[] xs = new double[40];
           double[] ys = new double[40];
           double h = 0.1;
           double x = 0;

           for (int i = 0; i < 40; i++)
           {
               xs[i] = x;
               ys[i] = ig.GetIntegral(0, x, 10, y);
               x += h;
           }
        }
    }
}
```

```
//Dichotomy.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFrmShell.Lib
{

    public class Dichotomy
    {
        public double find(double a, double b, uint n, double alpha, Function1 func, double eps =
0.000001)
        {
            double left = a;
            double right = b;
            double x;
            double f;
            int iter = 0;
            do
            {
                x = (left + right) / 2;
                f = func(0, x, n, alpha);
                if (f > 0)
                {
                    right = x;
                }
                else
                {
                    left = x;
                }
                iter++;
            } while (Math.Abs(f) > eps && iter < 20000);
            return x;
        }
    }
}


//Functions.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFrmShell.Lib
{
    public static class Funtions
    {
        public static double fGauss(double t)
        {
            double c = 1 / Math.Sqrt(2 * Math.PI);
            double d = Math.Sqrt(1 / Math.Exp(Math.Pow(t, 2)));
            return c * d;
        }
    }
}
```

```
//IIntegrals.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFrmShell.Lib
{
    interface IIntegrals
    {
        double GetIntegral(double a, double b, uint n, double alpha);
    }
}



//IntagralGauss.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFrmShell.Lib
{
    public delegate double Function1(double a, double b, uint n, double alpha);
    public class IntagralGauss : IIntegrals
    {
        private double GetIntegral(double a, double b, uint n, double alpha)
        {
            double d = 2 * Math.Sqrt(3.0);
            double result = 0.0;
            double step = (b - a) / n;
            double a1, b1;
            a1 = 0; b1 = 0;
            for (int i = 0; i < n; i++)
            {
                if (i == 0)
                {
                    a1 = a;
                    b1 = a1 + step;
                }
                else
                {
                    if (i == n - 1)
                    {
                        a1 = b1;
                        b1 = b;
                    }
                    else
                    {
                        a1 = b1;
                        b1 = a1 + step;
                    }
                }
                result += ((b1 - a1) / 2) * (Lib.Funtions.fGauss((a1 + b1) / 2 - (b1 - a1) / d) +
(Lib.Funtions.fGauss((a1 + b1) / 2 + (b1 - a1) / d)));

            }
            return Math.Round(result-alpha, 7);

        }
```

```
        double IIntegrals.GetIntegral(double a, double b, uint n, double alpha)
        {
            return this.GetIntegral(a, b, n, alpha);
        }
    }
}


//IntagralGauss.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFrmShell.Lib
{
    public class IntegralGaussLegandr:IIntegrals
    {
        private double Get(double a, double b, uint n, double alpha)
        {
            double I = 0;
            double[] A = null;
            double[] ts = null;
            this.GetPolinomLegandra(n, ref ts, ref A);
            double[] xs = this.GetXs(ts, a, b);
            double mn = (b - a) / 2.0;
            for (int i = 0; i < n; i++)
            {
                I += A[i] *Lib.Funtions.fGauss(xs[i]);
            }
            I = I * mn;
            return I-alpha;

        }

        private void GetPolinomLegandra(uint n, ref double[] ts, ref double[] a)
        {

            if (n == 1)
            {
                a = new double[] { 2.0 };
                ts = new double[] { 0.0 };
                return;
            }
            if (n == 2)
            {
                a = new double[] { 1.0, 1.0 };
                ts = new double[] { -0.577350269189626, 0.577350269189626 };
                return;
            }
            if (n == 3)
            {
                a = new double[] { 0.555555555555555, 0.888888888888889, 0.555555555555555 };
                ts = new double[] { -0.774596669241483, 0, 0.774596669241483 };
                return;
            }
            if (n == 4)
            {
                a = new double[] { 0.347854845137454, 0.652145154862546, 0.652145154862546,
0.347854845137454 };
                ts = new double[] { -0.861136311594052, -0.339981043584856, 0.339981043584856,
0.861136311594052 };
                return;
            }
```

```
        if (n == 5)
            {
                a = new double[] { 0.236926885056189, 0.478628670499366, 0.568888888888889,
0.478628670499368, 0.236926885056189 };
                ts = new double[] { -0.906179845938664, -0.538469310105683, 0.0, 0.538469310105683,
0.906179845938664 };
                return;
            }
            if (n == 6)
            {
                a = new double[] { 0.17132449237917, 0.360761573048139, 0.467913934572691,
0.467913934572691, 0.360761573048138, 0.171324492379171 };
                ts = new double[] { -0.932469514203152, -0.661209386466265, -0.238619186083197,
0.238619186083197, 0.661209386466265, 0.932469514203152 };
                return;
            }
            if (n == 7)
            {
                a = new double[] { 0.12948496616887, 0.279705391489277, 0.381830050505119,
0.417959183673469, 0.38183005050512, 0.279705391489277, 0.129484966168869 };
                ts = new double[] { -0.949107912342758, -0.741531185599394, -0.405845151377397, 0.0,
0.405845151377397, 0.741531185599394, 0.949107912342758 };
                return;
            }

            if (n == 8)
            {
                a = new double[] { 0.101228536290377, 0.222381034453375, 0.313706645877887,
0.362683783378362, 0.362683783378362, 0.313706645877888, 0.222381034453374, 0.101228536290376 };
                ts = new double[] { -0, 960289856497536, -0, 796666477413627, -0, 525532409916329, -0,
18343464249565, 0, 18343464249565, 0, 525532409916329, 0, 796666477413627, 0, 960289856497536 };
                return;
            }
            if (n == 8)
            {
                a = new double[] { 0.101228536290377, 0.222381034453375, 0.313706645877887,
0.362683783378362, 0.362683783378362, 0.313706645877888, 0.222381034453374, 0.101228536290376 };
                ts = new double[] { -0, 960289856497536, -0, 796666477413627, -0, 525532409916329, -0,
18343464249565, 0, 18343464249565, 0, 525532409916329, 0, 796666477413627, 0, 960289856497536 };
                return;
            }
            if (n == 9)
            {
                a = new double[] { 0.081274388361574, 0.180648160694858, 0.260610696402936,
0.312347077040002, 0.33023935500126, 0.312347077040002, 0.260610696402936, 0.180648160694857,
0.0812743883615747 };
                ts = new double[] { -0.968160239507626, -0.836031107326636, -0.61337143270059, -
0.324253423403809, 0, 0.324253423403809, 0.613371432700591, 0.836031107326636, 0.968160239507626 };
                return;
            }
            throw new ArgumentException("n>9 или n<1");
        }
        private double[] GetXs(double[] ts, double a, double b)
        {
            double[] xs = new double[ts.Length];
            for (int i = 0; i < xs.Length; i++)
            {
                xs[i] = (b + a) / 2 + (((b - a) / 2) * ts[i]);
            }
            return xs;
        }
```

```
        double IIntegrals.GetIntegral(double a, double b, uint n, double alpha)
        {
            return this.Get(a, b, n, alpha);
        }
    }
}


//NewtonKotes.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFrmShell.Lib
{
    public class NewtonKotes:IIntegrals
    {
        double IIntegrals.GetIntegral(double a, double b, uint n, double alpha/*, Function f*/)
        {
            return this.Calc(a, b, n, alpha/*, f*/);
        }

        uint n = 0;
        double a;
        double b;
        private double[] h;
        public double[] H
        {
            get { return this.h; }
        }
        public double Calc(double a, double b, uint n, double alpha)
        {
            if (n < 1)
                throw new ArgumentException("n<1)");
            if (n > 9)
                throw new ArgumentException("n>8)");
            this.n = n;
            this.a = a;
            this.b = b;
            this.h = this.GetH(this.n);

            double step = (b - a) / n;
            double[] xs = new double[n + 1];
            xs[0] = a;
            xs[n] = b;
            for (int i = 1; i < n; i++)
            {
                xs[i] = a + (step * i);
            }
            double mn = (b - a) / 2;
            double I = 0;
            for (int i = 0; i <= this.n; i++)
            {
                I += H[i] * Lib.Funtions.fGauss(xs[i]);
            }
            I = I * mn-alpha;
            return I;
        }
```

```
        private double[] GetH(uint n)
        {
            if (n == 1)
                return new double[] { 1d, 1d };
            if (n == 2)
                return new double[] { 1d / 3d, 4d / 3d, 1d / 3d };
            if (n == 3)
                return new double[] { 1d / 4d, 3d / 4d, 3d / 4d, 1d / 4d };
            if (n == 4)
                return new double[] { (7d / 45d), (32d / 45d), (12d / 45d), (32d / 45d), (7d / 45d) };
            if (n == 5)
                return new double[] { 19d / 144d, 75d / 144d, 50d / 144d, 50d / 144d, 75d / 144d, 19d /
144d };
            if (n == 6)
                return new double[] { 41d / 420d, 216d / 420d, 37d / 420d, 275d / 420d, 27d / 420d,
216d / 420d, 41d / 420d };
            if (n == 7)
                return new double[] { 751d / 8640d, 3577d / 8640d, 1323d / 8640d, 2989d / 8640d, 2989d
/ 8640d, 1323d / 8640d, 3577d / 8640d, 751d / 8640d };
            if (n == 8)
                return new double[] { 989d / 14175d, 5888d / 14175d, -928d / 14175d, 10496d / 14175d, -
4540d / 14175d, 10496d / 14175d, -928d / 14175d, 5888d / 14175d, 989d / 14175d };
            if (n == 9)
                return new double[] { 2857d / 44800d, 15741d / 44800d, 1080d / 44800d, 19344d / 44800d,
5778d / 44800d, 5778d / 44800d, 19344d / 44800d, 1080d / 44800d, 15741d / 44800d, 2857d / 44800d, };
            throw new ArgumentException();
        }
    }
}
```