Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования "Уфимский государственный авиационный технический университет"

Кафедра Высокопроизводительных вычислительных технологий и систем

Дисциплина: Численные методы

Отчет по лабораторной работе № 2

Тема: «Численное интегрирование»

Группа ПМ-353	Фамилия И.О.	Подпись	Дата	Оценка
Студент	Шамаев И.Р.			
Принял	Гайнетдинова А. А.			

Уфа 2021

Требования к составу отчета:

Отчет к лабораторной работе оформляется в текстовом процессоре Microsoft Word или OpenOffice (LibreOffice) Write в соответствии с требованиями стандарта СТО УГАТУ 016-2007 и содержать

- титульный лист,
- описание цели работы и краткую теоретическую справку по использованным операторам и функциям языка программирования,
- описание выполнения задания:
  i. формулировка задания, как в методичке,
  ii. блок-схема каждой разработанной функции,
  iii. исходный текст разработанного приложения,
  iv. скриншоты примеров выполнения программы.
- выводы
- список использованной литературы,

Требования к оформлению отчета

- Текст отчета набирается шрифтом Times New Roman, размер шрифта 14pt или 12pt единый во всем документе. Исходный код должен быть набран любым моноширинным шрифтом (например, Courier New), размер символов можно уменьшить до 10 pt. Поля страницы: верхнее и нижнее – 2 см., левое – 2 см, правое – 1.5 см.
- Абзац должен начинаться с красной строки, за исключением тех случаев, когда абзац разорван каким-либо математическим выражением. Выравнивание внутри текстовых абзацев «по ширине», выравнивание в блоке с исходным кодом – «по левому краю».
- Все блок-схемы оформляются в соответствие с ГОСТ 19.701. Описание ГОСТ и пример блок-схемы приведены в разделе Справочник, методических указаний по выполнению лабораторных работ на сайте. Если блок-схемы составляются в сторонних приложениях, следует предусмотреть возможность их правки в учебном классе. В противном случае отчет с некорректно составленной блок-схемой и исправлениями вручную приниматься не будут.
- Описание выполнения каждого задания начинается с новой страницы. Следует использовать заголовок с текстом «**Индивидуальное задание №__**», набранный полужирным шрифтом с выравниванием по левому краю, без красной строки. Текст задания, текст в блок-схеме и описание программы следует набирать шрифтом Times New Roman с прямым начертанием (не курсив!!!).
- Текст на скриншотах должен быть читаемым. Скриншоты должны содержать только окно с результатами выполнения программы, а не весь рабочий стол. Размер шрифта на скриншотах должен соответствовать по размеру окружающему его тексту.

Ниже и выше красным выделены места, которые необходимо изменить.

Цель: получить навык приближенного вычисления определенных интегралов.

Теоретический материал

Задача 1. Вычисление интеграла по квадратурным формулам прямоугольников и трапеций на равномерной сетке.

Квадратурные формулы для прямоугольников и трапеций можно получить из наглядных соображений. Пусть вычисляется интеграл

$$I=\int_a^b f(x)\,dx.$$

Если $f(x)\approx const$ на рассматриваемом отрезке [a, b], то можно положить $I\approx(b-a)f(\zeta),\zeta-¿$ произвольная точка на [a, b]. Естественно, взять в качестве ζ среднюю точку отрезка; тогда получим *формулу прямоугольников*

$$I\approx(b-a)f\left(\frac{a+b}{2}\right).$$

Предположим, что функция $f(x)$ на [a, b] близка к линейной; тогда естественно заменить интеграл площадью трапеции с высотой $(b-a)$ и основаниями $f(a)$ и $f(b)$. Получим *формулу трапеций*

$$I\approx(b-a)\frac{f(a)+f(b)}{2}.$$

Сетка значений x задается равномерно на отрезке [a, b].

Абсолютная погрешность вычисления интеграла от количества узлов сетки находится как

$$\Delta=|J-\bar{J}|.$$

Задача 2. Вычисление интеграла по квадратурной формуле Симпсона.

Введем на [a, b] равномерную сетку с шагом h = (b - a) / N.

Для построения формулы численного интегрирования на всем отрезке [a, b] достаточно построить квадратурную формулу для интеграла

$$\int_{x_{i-1}}^{x_i} f(x)\,dx$$

на частичном отрезке $[x_{i-1}, x_i]$.

При аппроксимации данного интеграла заменим функцию f(x) параболой, проходящей через точки $(x_j, f(x_j)), j = i-1, i-0,5, i,$ т.е. представим приближенно f(x) в виде

$$f(x) \approx L_{2,j}(x), x \in [x_{i-1}, x_i],$$

где $L_{2,j}(x)$ – интерполяционный многочлена Лагранжа второй степени,

$$L_{2,j}(x) = \frac{2}{h^2}\left\{\left(x - x_{i-\frac{1}{2}}\right)(x - x_i) f_{i-1} - 2(x - x_{i-1})(x - x_i) f_{i-\frac{1}{2}} + (x - x_i)\left(x - x_{i-\frac{1}{2}}\right) f_i\right\}.$$

Проводя интегрирование, получим

$$\int_{x_{i-1}}^{x_i} L_{2,i}(x)\,dx = \frac{h}{6}\left(f_{i-1} + 4 f_{i-\frac{1}{2}} + f_i\right), h = x_i - x_{i-1}.$$

Таким образом, приходим к приближенному равенству

$$\int_{x_{i-1}}^{x_i} f(x)\,dx \approx \frac{h}{6}\left(f_{i-1} + 4 f_{i-\frac{1}{2}} + f_i\right),$$

которое называется *формулой Симпсона или формулой парабол.*

Чтобы не использовать дробных индексов, можно обозначить

$x_i = a + 0,5hi, f_i = f(x_i), i = 0, 1, \ldots, 2N, hN = b - a,$

тогда на всем отрезке [a, b] формула Симпсона имеет вид

$$\int_a^b f(x)dx \approx \frac{b-a}{6N}\left[f_0 + f_{2N} + 2\left(f_2 + f_4 + \ldots + f_{2N-2}\right) + 4\left(f_1 + f_3 + \ldots + f_{2N-1}\right)\right].$$

Задача 3. Реализация процесса Эйткена и метода Рунге повышения порядка точности квадратуры, правило Ромберга.

Для применения метода Рунге необходимо знать, каков порядок точности исходной формулы.

Предположим, что порядок точности *p* существует, но неизвестен нам. В этом случае можно уточнить результат, если расчеты проведены на трех (или более) сетках. Метод расчета на нескольких сетках применяется для повышения порядка точности даже в том случае, когда неизвестен порядок главного члена погрешности и называется *процессом Эйткена*.

Пусть заданы три равномерные сетки: $h_1=h, h_2=kh, h_3=k^2 h; k\in(0;1)$

Предположим, что

$$D_N[f]=J[f]-S_N[f]=A_p h^p+O(h^q); q>p.$$

Рассмотрим следующее выражение

$$B=\frac{S_{N_1}[f]-S_{N_2}[f]}{S_{N_2}[f]-S_{N_3}[f]}$$

Применяя формулу оценки главного члена асимптотического разложения, получим

$$B\approx\frac{A_p\left(h_1^p-h_2^p\right)}{A_p\left(h_2^p-h_3^p\right)}=\frac{1-k^p}{k^p\left(1-k^p\right)}=\frac{1}{k^p}$$

$$\ln(B)=-p\ln(k)$$

$$p=\frac{-\ln(B)}{\ln(k)}$$

- порядок главного члена асимптотического разложения погрешности.

Для квадратурных формул можно получить асимптотическое разложение вида

$$D_N(f)=J_N(f)-J(f)=\propto_2 h^2+\propto_4 h^4+\propto_6 h^6+\ldots,$$

Если f(x) является достаточно гладкой функцией. При этом ¿ $\propto_{k+2}\vee$¿ значительно меньше $|\propto_k|, (k=2,4)$,поэтому повышение порядка точности квадратурной формулы весьма важно.

Проведем расчеты на двух равномерных сетках с шагами h_1 и h_2 соответственно и найдем выражения $J^{h_1}[f]=J_{N_1}[f]$ и $J^{h_2}[f]=J_{N_2}[f], h_1N_1=h_2N_2=b-a$.

Параметр σ находится как

$$\sigma=\frac{h_2^p}{h_2^p-h_1^p}.$$

В итоге приближенное вычисление интеграла с повышением порядка точности производится по формуле

$$\tilde{J}=\sigma J^{h_1}+(1-\sigma)J^{h_2}.$$

Метод Ромберга заключается в последовательном уточнении значения интеграла при кратном увеличении числа разбиений. В качестве базовой может быть взята формула трапеций с равномерным шагом h.

Обозначим интеграл с числом разбиений n = 1 как

$$R(1;1)=\frac{h}{2}(f(a)-f(b))$$

Уменьшив шаг в два раза, получим

$$R(2;1)=\frac{h}{2}(f(a)+f(b)+hf(a+h))$$

Если последовательно уменьшать шаг в 2^n раз, получим рекуррентное соотношение для расчета

$$R(n+1;1)=\frac{1}{2}R(n;1)+h\sum_{i=1}^{2^n-1}f(a+(2i-1)h)$$

Пусть мы вычислили четыре раза интеграл с *n* от 1 до 4. Представим следующий треугольник:

R(1;1)
R(2;1) R(2;2)
R(3;1) R(3;2) R(3;3)
R(4;1) R(4;2) R(4;3) R(4;4)

В первом столбце стоят значения интеграла, полученные при последовательном удвоении числа интервалов. Следующие столбцы – результаты уточнения значения интеграла по следующей рекуррентной формуле:

$$R(n+1;m+1)=R(n+1;m)+\frac{R(n+1;m)-R(n;m)}{4^m-1}$$

Правое нижнее значение в треугольнике – искомое уточненное значение интеграла.

Задача 4. Квадратурная формула Гаусса

Полиномы вида

$$P_n(x)=\frac{1}{2^n n!}\frac{d}{dx}\left[\left(x^2-1\right)^n\right], n=0,1,2,\ldots$$

называются полиномами Лежандра. Свойства полиномов Лежандра:

1) $P_n(1)=1$, $P_n(-1)=(-1)^n$ (n=0,1,2...)

2) Свойство ортогональности:

$$\int_{-1}^{1} P_n(x)Q_k(x)dx=0, k<n$$

где $Q_k(x)$ - любой полином степени k<n.

3) Полином Лежандра имеет n различных и действительных корней, которые расположены на интервале (-1, 1).

Перейдём теперь к выводу квадратурной формулы Гаусса. Рассмотрим функцию y=f(x) на [-1, 1]. Поставим задачу: как нужно подобрать точки t_1, t_2,..., t_n и коэффициенты A_1, A_2,..., A_n, чтобы квадратурная формула

$$\int_{-1}^{1} f(t)\,dt=\sum_{i=1}^{n} A_i f(t_i) \quad (1)$$

была точной для всех полиномов f(t) наивысшей возможной степени N=2n-1 (т. к. нужно определить 2n коэффициентов $\{A_i, t_i\}$, а полином степени 2n-1 имеет 2n коэффициентов).

Для обеспечения равенства (1) необходимо и достаточно, чтобы оно было верным при

$$f(t)=1,t,t^2,\ldots,t^{2n-1}$$

При этом мы получим следующую систему уравнений относительно A_i, t_i:

$f(t)=1, t, t^2, ..., t^{2n-1}$. (2)

Таким образом, учитывая соотношение

$$\int_{-1}^{1} t^k\,dt=\frac{1-(-1)^{k+1}}{k+1}=\begin{cases}\frac{2}{k+1} \text{ при } k-\text{чётном}\\ 0 \text{ при } k-\text{нечётном}\end{cases}$$

Получим из (2) систему из 2n уравнений для определения коэффициентов A_i и узлов t_i:

$$\begin{cases} \sum_{i=1}^{n} A_i = 2 \\ \sum_{i=1}^{n} A_i t_i = 0 \\ \dots \\ \sum_{i=1}^{n} A_i t_i^{2n-2} = \frac{2}{2n-1} \\ \sum_{i=1}^{n} A_i t_i^{2n-1} = 0 \end{cases}$$

Формула (1), где t_i – нули полиномов Лежандра, а A_i определяются из системы выше, называется квадратурной формулой Гаусса.

Задача 5. Оптимальное распределение узлов квадратурной формулы трапеций

Для вычисления интеграла с узлами интегрирования, наилучшим образом приближенными к оптимальному распределению используется формула, основанная на составной формуле трапеций

$$I_q(f) \approx \frac{a_q - a_{q-1}}{2} \sum_{j=1}^{m} D_j f\left(\frac{a_{q-1} + a_q}{2} + \frac{a_q - a_{q-1}}{2} d_j \right),$$

где $[a_{q-1}, a_q]$ – элементарный отрезок разбиения, D_j – веса для квадратурной формулы Гаусса, d_j – корни многочлена Лежандра.

Задача 6. Вычисление интеграла методом Монте-Карло

Пусть требуется вычислить приближенное значение интеграла

$$I(f)=\int_G f(P)\,dP.$$

Предполагаем, что μ(G) – мера области G равна 1. Как правило, это условие бывает выполнено, поскольку при практической реализации метода Монте-Карло область интегрирования обычно преобразуется в единичный куб. Предположим, что каким-то образом удалось получить N случайных попарно независимых точек $P_1, \ldots, P_N$, равномерно распределенных в G. Случайные величины $s_j = f(P_j)$ попарно независимы и одинаково распределены.

В области G формула Монте-Карло имеет вид

$$S_N(f)=\frac{1}{N}\sum_{j=1}^{N} s_j.$$

Тогда на отрезке [a, b], разбитом на N случайных точек $x_1, x_2, \ldots, x_n$ приближенное значение определенного интеграла можно вычислить как

$$\int_a^b f(x)dx \approx (b-a)\hat{f},$$

где

$$\hat{f}=\frac{1}{N}\sum_{i=1}^{N} f(x_i).$$

Абсолютная погрешность вычисления интеграла от количества узлов сетки находится как

$$\Delta=|J-\bar{J}|.$$

Индивидуальное задание №1

Задание:

1) Написать вычислительную программу на языке программирования C++ для вычисления интеграла J[f] по квадратурным формулам прямоугольников и трапеций на равномерной сетке.

2) Построить графики зависимости абсолютной погрешности

$$\Delta = ¿ J - \overline{J} \vee ¿$$

вычисления интеграла с использованием обеих формул от количества узлов сетки.

3) Для каждой из квадратурных формул определить минимальное количество узлов равномерной сетки, обеспечивающее вычисление интеграла с указанной в индивидуальном задании величиной абсолютной погрешности $\overline{\Delta}$.

Описание программы**:**

Функция void Task1() находит численное решение определённого интеграла по формулам квадратур прямоугольников и трапеций из теоретической части (задание 1) по узлам от 1 до минимального числа для нужной погрешности (в задании это 10^{-6}). Далее строится график зависимости абсолютной погрешности от числа узлов.

Исходный код программы

Исходный код программы предоставлен в приложении А.

Пример выполнения программы

```
**********Задание 1******************
Реальное значение интеграла: 0.612889
число узлов для прямоугольников: 436
число узлов для трапеций: 616
погрешность для прямоугольников:9.96119e-07
погрешность для трапеций:9.98223e-07
```

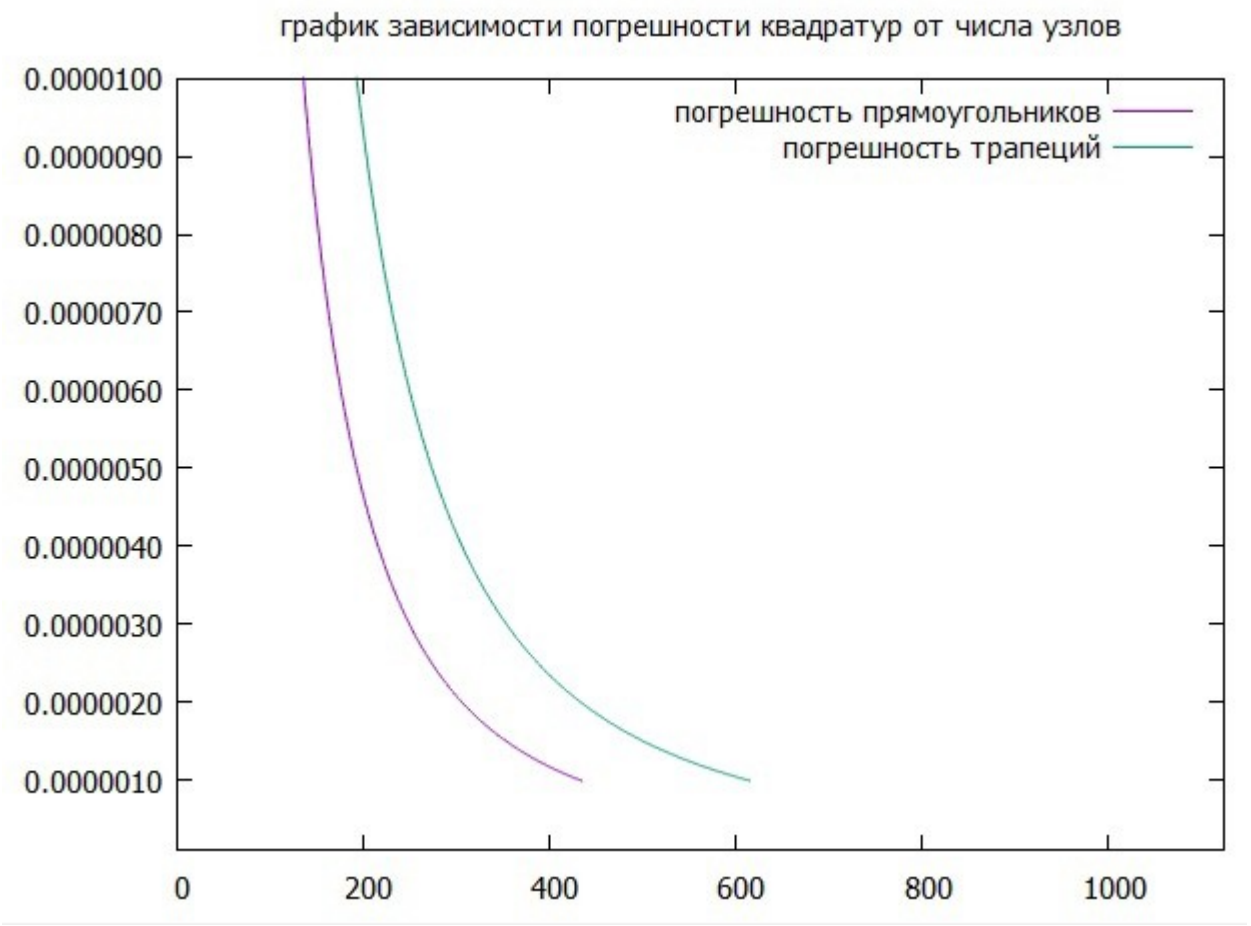


график зависимости погрешности квадратур от числа узлов
погрешность прямоугольников
погрешность трапеций
0.0000100
0.0000090
0.0000080
0.0000070
0.0000060
0.0000050
0.0000040
0.0000030
0.0000020
0.0000010
0
200
400
600
800
1000

Индивидуальное задание №2

Задание: Выполнить п. 1-3 из Задачи 1 для квадратурной формулы Симпсона.

Описание программы**:**

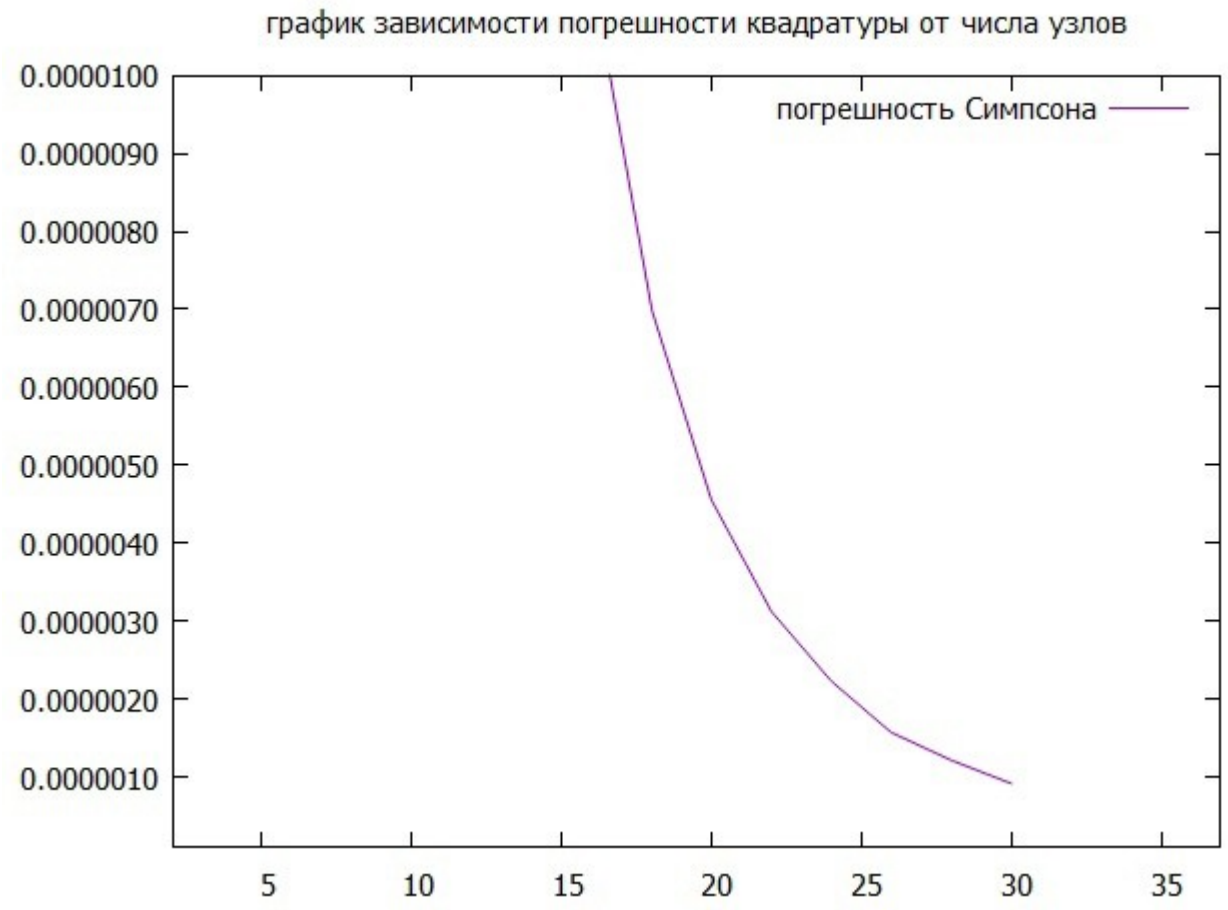
Функция void Task2() находит численное решение определённого интеграла по формуле квадратуры Симпсона из теоретической части (задание 2) по узлам от 1 до минимального числа для нужной погрешности (в задании это 10^{-6}). Далее строится график зависимости абсолютной погрешности от числа узлов.

Исходный код программы

Исходный код программы предоставлен в приложении А.

Пример выполнения программы

```
**********Задание 2******************
погрешность квадратуры Симпсона 9.2306e-07
минимальное число узлов для нужной погрешности 30
```



Индивидуальное задание №3

Задание:

1) С использованием написанной при решении Задачи 1 программы определить порядок главного члена погрешности квадратуры, реализовав программно процесс Эйткена.

2) Зная приближенное значение порядка главного члена погрешности, реализовать метод Рунге повышения порядка точности квадратуры. Определить порядок точности модифицированного метода.

3) С использованием правила Ромберга вычислить значение интеграла с абсолютной погрешностью $10^{-6}\overline{\Delta}$.

Описание программы**:**

Исходный код программы

Исходный код программы предоставлен в приложении А.

Пример выполнения программы

```
главный член погрешности 1.01252
значение интегралапо методу Рунге 0.612885
погрешность 4.02413e-06
Уточненное значение интеграла по Ромбергу = 0.612766 Погрешность = 0.000122995
```

Индивидуальное задание №4

Задание:

1) Заменой переменной интегрирования отобразить отрезок интегрирования [a,b] в [-1,1].

2) Построить квадратурную формулу Гаусса с единичным весом на системе ортогональных многочленов Лежандра.

3) Выполнить программную реализацию построенной квадратуры на языке программирования C++.

4) С использованием написанной программы построить график абсолютной погрешности приближенного вычисления интеграла от числа узлов сетки и определить минимальное количество узлов сетки, обеспечивающее вычисление интеграла с указанной в индивидуальном задании величиной абсолютной погрешности $\bar{\Delta}$.

Описание программы**:**

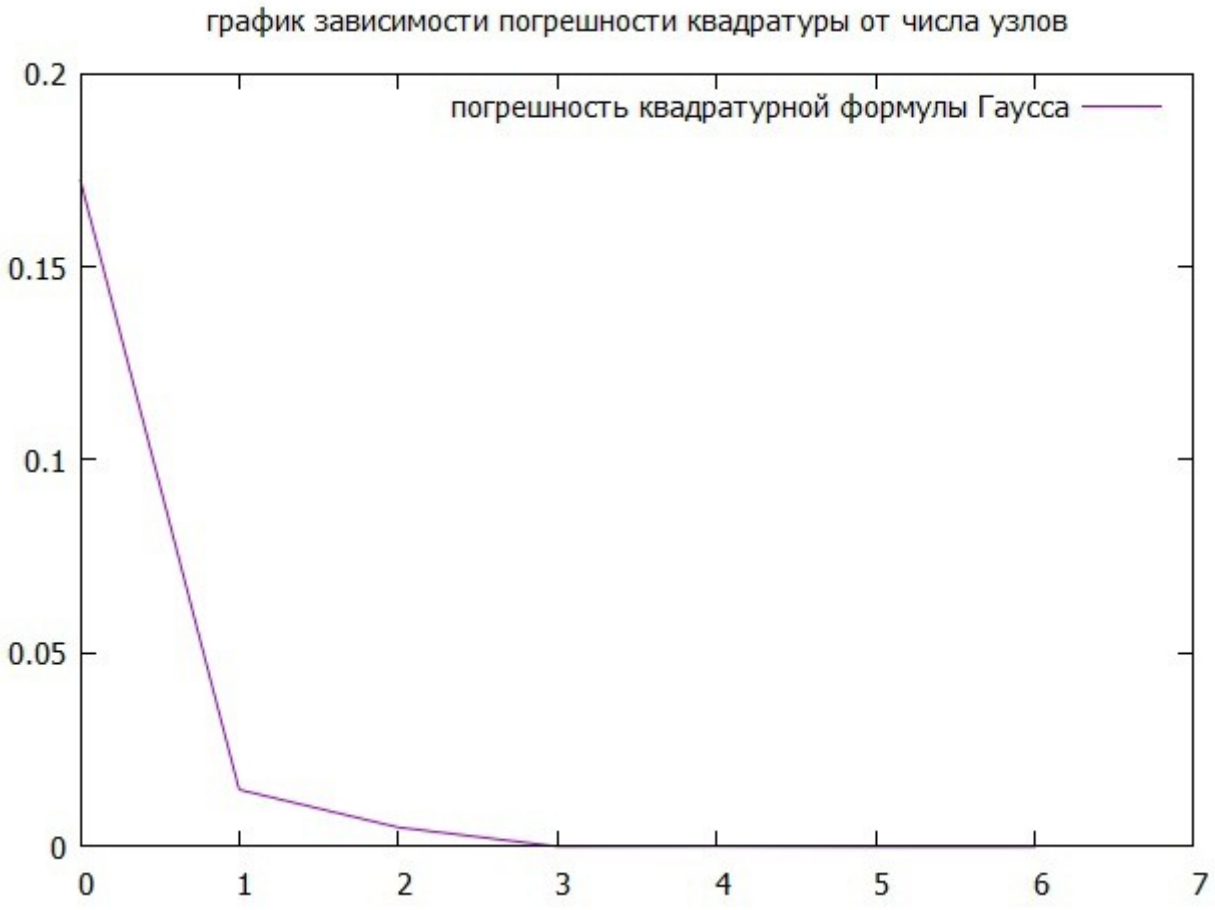
В программе используются уже готовые значения для корней многочленов Лежандра и весов квадратур.

Исходный код программы

Исходный код программы предоставлен в приложении А.

Пример выполнения программы

```
**********Задание 4**********
n       погрешность
1       0.172509
2       0.0147324
3       0.00499997
4       7.07201e-05
5       8.00184e-05
6       6.11029e-06
7       8.05474e-07
```



Индивидуальное задание №5

Задание:

1) Для заданного интеграла получить приближенное решение задачи об оптимальном распределении узлов квадратурной формулы трапеций.

2) Выполнить программную реализацию квадратуры с оптимальным распределением узлов.

3) С использованием написанной программы определить минимальное оптимальное количество узлов сетки, обеспечивающее вычисление интеграла с указанной в индивидуальном задании величиной абсолютной погрешности $\overline{\Delta}$.

Описание программы**:**

IntegrationMethod- класс Метод итераций

MonteCarlo – класс Монте карло

OptimalNodes – класс оптимального распределения узлов

maxFunctionOnSegment- функция максимума функции на сегменте

minError – функция минимальной ошибки с выводом в файл

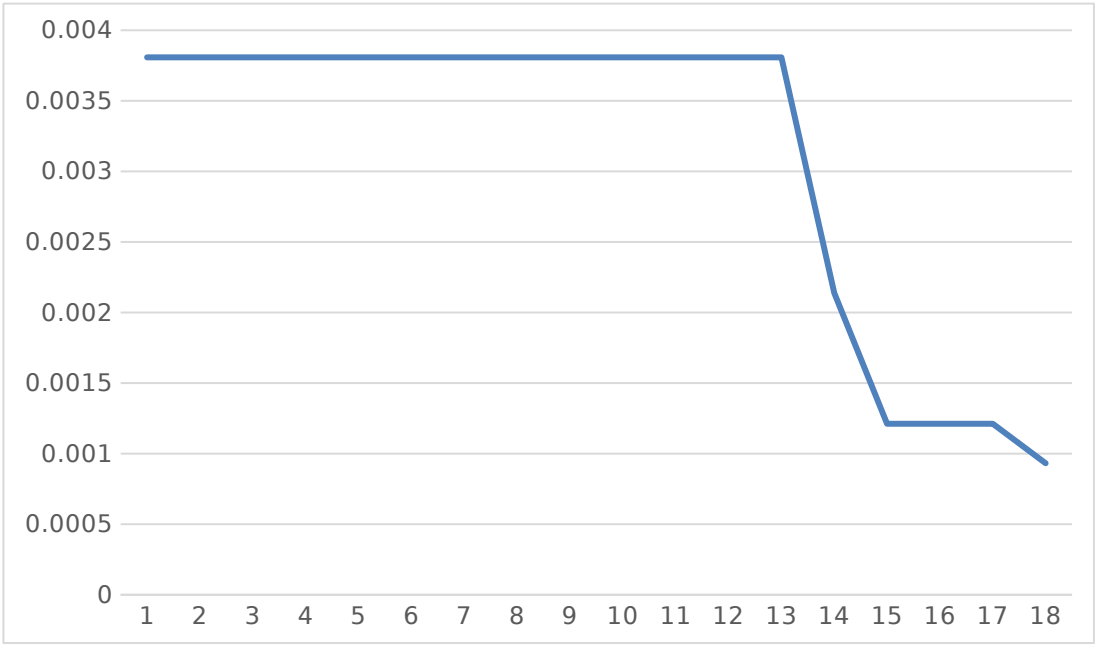
checkExpectationFromN-функция точности оценки мат. ожидания от функции

Исходный код программы

Исходный код программы предоставлен в приложении В.

Пример выполнения программы

```
Метод оптимального распределения узлов трапеции = 0.6128717119  Ошибка: 1.742909351e-05
Минимальное n метода оптимального распределения узлов трапеции = 18
```



Индивидуальное задание №6

Задание:

1) Написать программу на языке программирования C++ для приближенного вычисления интеграла методом Монте-Карло.

2) С использованием написанной программы построить график зависимости оценки математического ожидания абсолютной погрешности приближенного интегрирования от количества случайных точек метода. Размер выборки (количество повторных вычислительных экспериментов) для каждого случая принять равным 100.

Исходный код программы

Исходный код программы предоставлен в приложении В.

Пример выполнения программы

```
Интеграла по Монтекарло = 0.5862681741
Мат ожидание Монтекарло = 0.01751629289
```

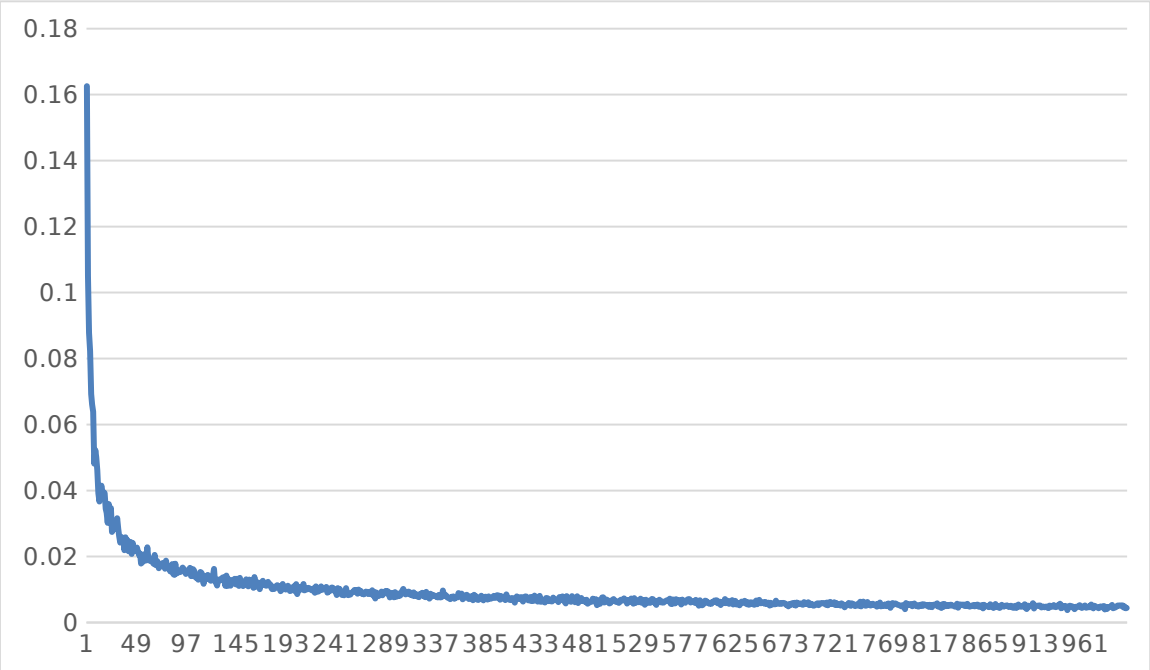


График 6. График зависимости оценки математического ожидания абсолютной погрешности

Вывод

В результате выполнения лабораторной работы был изучен теоретический материал для численного вычисления определённого интеграла. Для каждой задачи было реализовано решение на C++.

Список использованной литературы

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы: Бином, 2018. – 636 с.
2. Калиткин Н.Н. Численные методы, 2-е издание: БХВ-Петербург, 2014. – 592 с.
3. Самарский А.А., Гулин А. В. Численные методы: Учеб, пособие для вузов, — М.: Наука. Гл. ред. физ-мат. лит., 1989.— 432 с.

Приложение А

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <math.h>
#include <vector>
#include <ctime>
#include "gnuplot_iostream.h"
const int n = 15;
const double a = 0, b = 3/2.,s=-0.65169417504713;
const float er = 0.000001;
using namespace std;
double myfanc(double x)
{
  return x * x * cos(M_PI * x);
}
//первое задание
void Task1()
{
  Gnuplot gp1("\"C:\\Program Files\\gnuplot\\bin\\gnuplot.exe\"");
  double p_c=1,p_t=1,J_c=0,J_t=0;//J_c - прямоугольники, J_t - трапеции
  int n_c, n_t;
  vector<float> p1(815);
  vector<float> p2(1185);
  for (int k = 1; k < 1001; k++) {
  double h = (b - a) / k;
  for (int i = 1; i <= k; i++)
  {
            J_c += myfanc(((a + i * h) + (a + (i - 1.) * h)) / 2);
  }
  J_c *= h;
  p1[k - 1] = abs(s - J_c);
  if (abs(s - J_c) < er) { p_c = abs(s - J_c); n_c = k; k = 10000;}
  J_c = 0;
  }
  for (int j = 1; j < 10001; j++) {
  double h = (b - a) / j;
  for (int i = 1; i <= j; i++)
  {
            J_t += myfanc(a + i * h);
  }
  J_t = (J_t - myfanc(j * h)) * h + h * (myfanc(a) + myfanc(b)) / 2;
  p2[j - 1] = abs(s - J_t);
  if (abs(s - J_t) <er) { p_t = abs(s - J_t); n_t = j; j = 10000;}
  J_t = 0;
  }
  cout << "**********Задание 1*****************" << endl;
  cout <<"Реальное значение интеграла: "<<s<<endl<<
  "число узлов для прямоугольников: "<<n_c << endl <<
  "число узлов для трапеций: " << n_t << endl
```

```
<< "погрешность для прямоугольников:" << p_c << endl << "погрешность для трапеций:" << p_t<<endl;
gp1 << "set title 'график зависимости погрешности квадратур от числа узлов'\n";
gp1 << "set xrange[0:1125]\nset yrange[0.0000001:0.00001]\n" << "plot" << gp1.file1d(p1, "n_plot.dat") << "with lines title 'погрешность прямоугольников', "
<< gp1.file1d(p2, "pogreshnosti.dat") << "with lines title 'погрешность трапеций'," << endl;
cin.get();
}
//задание 2
void Task2()//формула Симпсона
{
Gnuplot gp2("\"C:\\Program Files\\gnuplot\\bin\\gnuplot.exe\"");
float p = 1, n, J_s = 0;
float s1 = 0, s2 = 0;
vector<float> p1;
vector<float> nn(36);
for (int k = 2; k < 1001; k+=2) {
float h = (b - a) / k;
for (int i = 1; i <= k/2.; i++)
{
s1 += myfanc(a + h * (2. * i - 1.));
if (i != k/2.) s2 += myfanc(a + h * 2. * i);
}
J_s = (4.*s1 + 2.*s2 + myfanc(a) + myfanc(b)) * (h / 3.);
p1.push_back( abs(s - J_s));
if (abs(s - J_s) < er) { p = abs(s - J_s); n = k; k = 10000; }
J_s = 0; s1 = 0; s2 = 0;
}
cout << "**********Задание 2*****************" << endl;
cout << "погрешность квадратуры Симпсона " << p << "\nминимальное число узлов для нужной погрешности "<<n<<endl;
gp2 << "set title 'график зависимости погрешности квадратуры от числа узлов'\n";
gp2 << "set xrange[2:36]\nset yrange[0.0000001:0.00001]\n" << "plot" << gp2.file1d(p1, "n_plot.dat") << "with lines title 'погрешность Симпсона', "
<< gp2.file1d(nn, "pogreshnosti.dat") << "with lines title "," << endl;
cin.get();
}
float rec(int n)//подсчёт интергала методом прямоугольников
{
float res = 0;
float h = (b - a) / n;
for (int i = 0; i < n; i++)
{
res += myfanc(a + i * h);
}
res = res * h;
return res;
}
float aitken_process(int N,float k)//процесс Эйткена
{
```

```
  srand(time(NULL));
  float S1 = rec(N), S2 = rec(N / k), S3 = rec(N / k / k);
  float B = (S1 - S2) / (S2 - S3);
  float p=-log(abs(B))/log(k);
  return p;
}
float runge( int n_1, int n_2)//метод Рунге повышения порядка точности квадратуры
{
  float k = float(n_1) / float(n_2);
  float p = aitken_process(n_1,k);
  float G = pow(k, p) / (pow(k, p) - 1.);
  float Sn_1;
  float Sn_2;
  Sn_1 = rec(n_1);
  Sn_2 = rec( n_2);
  float res = G * Sn_1 + (1. - G) * Sn_2;
  return res;
}
void romberg(int n, float k)//правило Ромберга
{
  float Sm01, Sm11, Sm21, Sm31;
  float Sm12, Sm22, Sm32;
  float Sm23, Sm33;
  float Sm44;
  Sm01 = rec(n);
  Sm11 = rec( 2 * n);
  Sm21 = rec( 4 * n);
  Sm31 = rec( 8 * n);
  Sm12 = Sm11 + (1. / (2 * 2 - 1)) * (Sm11 - Sm01);
  Sm22 = Sm21 + (1. / (2 * 2 - 1)) * (Sm21 - Sm11);
  Sm32 = Sm31 + (1. / (2 * 2 - 1)) * (Sm31 - Sm21);
  Sm23 = Sm22 + (1. / (2 * 2 * 2 - 1)) * (Sm22 - Sm12);
  Sm33 = Sm32 + (1. / (2 * 2 * 2 - 1)) * (Sm32 - Sm22);
  Sm44 = Sm33 + (1. / (2 * 2 * 2 * 2 - 1)) * (Sm33 - Sm23);
  cout << "Уточненное значение интеграла по Ромбергу = " << Sm44 << " Погрешность = " <<
abs(s - Sm44) <<
  endl;
}
//4 задание
void Task4()//Гаусс
{
  Gnuplot gp3("\"C:\\Program Files\\gnuplot\\bin\\gnuplot.exe\"");
  vector<float> p3(6);
  vector<float> nn(6);
  float J = 0; float x,h;
  //корни многочлена Лежандра
  float t[8][8] = { {0},{0.57735027,-0.57735027},{0.77459667,0,-0.77459667},
  {0.86113631 ,0.33998104 ,-0.33998104 ,-0.86113631},
  {0.90617985,0.53846931 ,0,-0.53846931 ,-0.90617985},
  {0.93246951,0.66120939 ,0.23861919 ,-0.23861919 ,-0.66120939 ,-0.93246951},
```

```
	{0.94910791,0.74153119 ,0.40584515 ,0,-0.40584515 ,-0.74153119 ,-0.94910791},
	{0.96028986,0.79666648 ,0.52553242 ,0.183434464 ,-0.183434464 ,-0.52553242 ,-
0.79666648 ,-0.96028986} };
	//веса квадратуры
	float c[8][8] = { {2},{1,1},{0.55555556,0.88888889,0.55555556},
	{0.34785484,0.65214516 ,0.65214516 ,0.34785484},
	{0.23692688,0.47862868,0.56888889 , 0.47862868,0.23692688},
	{0.17132450,0.36076158,0.46791394,0.46791394,0.36076158,0.17132450},
	{0.12948496,0.27970540,0.38183006,0.41795918,0.38183006,0.27970540,0.12948496},
	{0.10122854, 0.22238104, 0.31370664, 0.36268378, 0.36268378, 0.31370664, 0.22238104,
0.10122854} };
	cout <<"**********Задание 4**********"<< "n\tпогрешность\n";
	for (int i = 0; i < 8; i++)
	{
	h = (b - a) / (i+1);
	for (int j = 0; j < i+1; j++)
	{
		x = 0.5*(b-a)+0.5*(b-a)*t[i][j];
		J += c[i][j] * myfanc(x);
	}
	J *= (b - a) / 2.;
	cout << i + 1 << "\t" << abs(s - J) << endl;
	p3[i] = abs(s - J);
	if (abs(s - J) < er) {
		cout << "минимальное число узлов для нужной погрешности " <<
			i + 1 << endl; i = 10;	}
	J = 0;
	}
	gp3 << "set title 'график зависимости погрешности квадратуры от числа узлов'\n";
	gp3 << "set xrange[0:7]\nset yrange[0.0000001:0.00001]\n" << "plot" << gp3.file1d(nn,
"n_plot.dat") << "with lines title 'погрешность квадратурной формулы Гаусса', "
	<< gp3.file1d(p3, "pogreshnosti.dat") << "with lines title 'погрешность трапеций'," << endl;
	cin.get();
}
int main()
{
  system("chcp 1251");
  system("cls");
  //Task1();
  Task4();
  /*float kk = 0.4;
  int n1 = 120; int n2 = n1 / kk;
  float p = runge(n1, n2);
  cout << "**********Задание 3**********" << endl;
  cout << "главный член погрешности " << aitken_process(n1,kk) << endl;
  cout << "значение интегралапо методу Рунге " << p << endl << "погрешность " <<
  abs(s - p) << endl;
  romberg(n1, kk);
  Task4();*/
}
```

Приложение В

Задание 5,6

IntegrationMethod.h:

```
#pragma once
#include <string>

using namespace std;

class IntegrationMethod
{
public:
	IntegrationMethod(double a, double b, unsigned n, double realResult, double
delta, double(*functionPtr)(double));

	unsigned minError(string fileName);

	inline void setAB(double a, double b) { this->a = a; this->b = b; }

	virtual double calculateMethod() { return 0.0; };
	double calculateMethod(unsigned n);

	void setGrid(unsigned n);

	double orderAitken(unsigned startN);

	double methodAitken(unsigned startN);

	double methodRunge(unsigned N);


protected:
	string firstSchedule;
	double a;
	double b;
	unsigned n;
	double realResult;
	double (*functionPtr)(double);

	double delta;
};
```

MonteCarlo.h:

```
#pragma once
#include "IntegrationMethod.h"

class MonteCarlo : public IntegrationMethod
{
public:
	MonteCarlo(double a, double b, unsigned n, double delta, double realResult,
double(*functionPtr)(double));
	virtual double calculateMethod() override;

	void checkExpectationFromN(unsigned points, unsigned repetitions,
std::string fileName);

	double expectedValue(unsigned k);

private:


};
```

OptimalNodes.h:

```
#pragma once
#include "IntegrationMethod.h"
#include "TrapezoidIntegration.h"
#include <vector>

using namespace std;

class OptimalNodes : public IntegrationMethod
{
public:
	OptimalNodes(double a, double b, unsigned n, unsigned q, double realResult,
double delta, double(*functionPtr)(double),
		double(*secondDerivative)(double), vector<double>&
zerosThirdDerivative);

	virtual double calculateMethod() override;

private:
	double maxFunctionOnSegment(double x_i, double x_j);
	double convertSegmentToGivenInterval(double point);
	double convertSegmentForMethod(double point);
	double(*secondDerivative)(double);
	vector<double> zerosThirdDerivative;
	vector<pair<double, double>> b_l;
	vector<double> A_l;
	unsigned q;

	TrapezoidIntegration trapezoid;
};
```

TrapezoidIntegration.h:

```
#pragma once
#include "IntegrationMethod.h"

class TrapezoidIntegration : public IntegrationMethod
{
public:
	TrapezoidIntegration(double a = 0, double b = 0, unsigned n = 0, double
delta = 0,
		double realResult = 0, double(*functionPtr)(double) = [](double x) {
return 0.0; });
	virtual double calculateMethod();
	double calculateMethod(unsigned n);

private:

};
```

LB2.cpp:

```
#define _USE_MATH_DEFINES
#include <math.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include "MonteCarlo.h"
#include "OptimalNodes.h"
#include "TrapezoidIntegration.h"

#define realResult 0.612889141
#define delta 0.001
#define a 0.
#define b 2
```

```
#define n 100

using namespace std;

double Function(double x);
double FunctionSecondDerivative(double x);

int main() {


	setlocale(0, "Russian");


	MonteCarlo montecarlo{ a, b, n, realResult, delta, Function };
	cout << setprecision(10) << "Интеграла по Монтекарло = " <<
montecarlo.calculateMethod() << "\n";
	cout << setprecision(10) << "Мат ожидание Монтекарло = " <<
montecarlo.expectedValue(100) << "\n";
	montecarlo.checkExpectationFromN(1000, 100, "checkmateWaiting.dat");


	cout << "\n\n";

	vector<double> ZerosThirdDerivative{ 0.37918, 1.84659 };

	OptimalNodes optimalTrapezoid{ a, b, n, 10, realResult, delta, Function,
FunctionSecondDerivative, ZerosThirdDerivative };
	cout << setprecision(10) << "Метод оптимального распределения узлов
трапеции = " << optimalTrapezoid.calculateMethod() << "\tОшибка: " <<
fabs(optimalTrapezoid.calculateMethod() - realResult) << "\n";
	cout << "Минимальное n метода оптимального распределения узлов трапеции = "
<< optimalTrapezoid.minError("optimalTrapezoidError.dat") << "\n";


}

double Function(double x) {
	return atan(x)/(1+x*x);
}
double FunctionSecondDerivative(double x) {

	return (-6*x-2*atan(x)+6*atan(x)*x*x)/pow(1+x*x,3);
}
```

IntegrationMethod.cpp:

```
#include "IntegrationMethod.h"
#include <math.h>
#include <fstream>
#include <iostream>

using namespace std;

IntegrationMethod::IntegrationMethod(double a, double b, unsigned n, double
realResult, double delta, double(*functionPtr)(double))
{
	this->a = a;
	this->b = b;
	this->n = n;
	this->realResult = realResult;
	this->delta = delta;
	this->functionPtr = functionPtr;
}

unsigned IntegrationMethod::minError(string fileName)
{
```

```
	ofstream out(fileName);
	double error = 0.0;
	n = 0;
	do
	{
		n++;
		double temp = calculateMethod();
		//error = fabs(realResult - calculateMethod());
		double result = fabs(realResult - temp);
		out << n << " " << result << "\n";
		error = result;

	} while (error > delta);
	out.close();
	return n;
}

double IntegrationMethod::calculateMethod(unsigned n)
{
	unsigned lastN = this->n;
	setGrid(n);

	double result = calculateMethod();
	setGrid(lastN);

	return result;
}

void IntegrationMethod::setGrid(unsigned n)
{
	this->n = n;

}
```

MonteCarlo.cpp:

```
#include "MonteCarlo.h"
#include <time.h>
#include <random>
#include <fstream>

using namespace std;

MonteCarlo::MonteCarlo(double a, double b, unsigned n, double delta, double
realResult, double(*functionPtr)(double)) : IntegrationMethod(a, b, n, delta,
realResult, functionPtr)
{
	srand(time(0));
}

double MonteCarlo::calculateMethod()
{
	//Обычный алгоритм Монте-Карло интегрирования
	double u = 0.;
	double sum = 0.;
	double x = 0.;
	double value = 0.;
	for (int i = 0; i < n; i++) {
		u = double(rand()) / double(RAND_MAX);//случайная величина
		x = a + (b - a) * u;
		sum += functionPtr(x);
		value = sum / n;//выборочное среднее для нахождения оценки интеграла
```

```
	}
	return ((b - a) * value);
}

void MonteCarlo::checkExpectationFromN(unsigned points, unsigned repetitions,
string fileName)//Мат.ожижание от количества N(Грубо говоря точность оценки)
{
	ofstream outFile(fileName);

	for (unsigned i = 1; i <= points; ++i) {
		setGrid(i);
		outFile << i << " " << expectedValue(repetitions) << "\n";
	}
	outFile.close();

}


double MonteCarlo::expectedValue(unsigned k)//Ожидаемое значение
{
	double expectRes = 0;
	for (int i = 0; i < k; i++)
	{
		expectRes += fabs(realResult - calculateMethod());
	}

	return expectRes / k;

}
```

OptimalNodes.cpp:

```
#include "OptimalNodes.h"
#include <iostream>

using namespace std;

OptimalNodes::OptimalNodes(double a, double b, unsigned n, unsigned q, double
realResult, double delta, double(*functionPtr)(double), double(*secondDerivative)
(double),
	vector<double>& zerosThirdDerivative) : IntegrationMethod(a, b, n,
realResult, delta, functionPtr)
{
	trapezoid = { a, b, n, delta, realResult, functionPtr };
	this->secondDerivative = secondDerivative;
	this->q = q;

	for (int i = 0; i < zerosThirdDerivative.size(); i++)
	{
		this-
>zerosThirdDerivative.push_back(convertSegmentForMethod(zerosThirdDerivative[i]))
;
	}
}

double OptimalNodes::calculateMethod()
{
	double h = 1. / q;//шаг

	double x_i = 0.;
	double x_j = 0.;
	double sum = 0.;
	double six_lambda = 0.;//оценка погрешности по элементраному отрезку
интегрирования bl/Nl
	double answer = 0.;
	double checkN;
```

```
	double tempRes;


	for (int i = 0; i < q; ++i) {
		x_i = h * i;
		x_j = h * (i + 1);
		b_l.push_back(make_pair(x_i, x_j));//создаем пару(интервал)
		A_l.push_back(maxFunctionOnSegment(x_i, x_j));


		sum += (x_j - x_i) * pow(A_l.back(), 1. / 3.);
	}
	six_lambda = sum / n;
	double tPow;
	for (int i = 0; i < q; ++i) {
		trapezoid.setAB(convertSegmentToGivenInterval(b_l[i].first),
convertSegmentToGivenInterval(b_l[i].second));

		tPow = pow(A_l[i], 1. / 3.);

		tempRes = (int)(((b_l[i].second - b_l[i].first)) * tPow /
six_lambda);

		checkN = tempRes != 0 ? tempRes : 1;


		trapezoid.setGrid(checkN);

		answer += trapezoid.calculateMethod();

	}

	return answer;

}

double OptimalNodes::maxFunctionOnSegment(double x_i, double x_j)
{
	double functionZero;
	double functionA =
fabs(secondDerivative(convertSegmentToGivenInterval(x_i)));
	double functionB =
fabs(secondDerivative(convertSegmentToGivenInterval(x_j)));

	for (int i = 0; i < zerosThirdDerivative.size(); ++i) {

		if (x_i <= zerosThirdDerivative[i] && x_j >= zerosThirdDerivative[i])
//находит интервал ф, где 3 производная обращается в 0
		{
			functionZero =
fabs(secondDerivative(convertSegmentToGivenInterval(zerosThirdDerivative[i])));
			return ((functionZero >= functionA && functionZero >=
functionB) ?
				functionZero : ((functionA >= functionB) ? functionA :
functionB));
		}
	}
	return ((functionA >= functionB) ? functionA : functionB);
}

double OptimalNodes::convertSegmentToGivenInterval(double point)
{
	return ((b - a) * point + a);
}
```

```
double OptimalNodes::convertSegmentForMethod(double point)
{
	return ((point - a) / (b - a));
}
```

TrapezoidIntegration.cpp:

```
#include "TrapezoidIntegration.h"
#include <math.h>
#include <iostream>

TrapezoidIntegration::TrapezoidIntegration(double a, double b, unsigned n, double
delta, double realResult,
	double(*functionPtr)(double)) : IntegrationMethod(a, b, n, delta,
realResult, functionPtr)
{
}

double TrapezoidIntegration::calculateMethod()
{
	double sum = 0.0;
	double h = (b - a) / n;
	double x_i = 0.0;
	double x_j = 0.0;

	for (unsigned i = 0; i < n; i++)
	{
		x_i = a + h * i;
		x_j = a + h * (i + 1);
		sum += (functionPtr(x_i) + functionPtr(x_j));
	}
	sum = sum * 0.5 * h;//формула трапеции
	return sum;
}

double TrapezoidIntegration::calculateMethod(unsigned n)
{
	setGrid(n);
	return calculateMethod();
}
```