# Задание

Написать программу, на языке С++ в среде VisualStudio “Вычисление с заданной точностью итерационным численным методом”.

Вычисление производится методом последовательных приближений.

* Входные данные: числа a и b,
* функция f(x) задается в тексте программы,
* точность е.

Однородные вычислительные структуры или среды (ОВС), как правило, относятся к классу ОКМД (согласно классификации Флинна) и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

# OpenMP

Среди специалистов, занимающихся параллельными вычислениями, популярна шутка «Параллельные вычисления — технология будущего... и так будет всегда». Эта шутка не теряет актуальность уже несколько десятилетий. Аналогичные настроения были распространены в сообществе разработчиков архитектур компьютеров, обеспокоенном тем, что скоро будет достигнут предел тактовой частоты процессоров, однако частоты процессоров продолжают повышаться, хотя гораздо медленнее, чем раньше. Сплав оптимизма специалистов по параллельным вычислениям и пессимизма архитекторов систем способствовал появлению революционных многоядерных процессоров.

Главные производители процессоров сместили акцент с повышения тактовых частот на реализацию параллелизма в самих процессорах за счет использования многоядерной архитектуры. Идея проста: интегрировать в один процессор более одного ядра. Система, включающая процессор с двумя ядрами, по сути, не отличается от двухпроцессорного компьютера, а система с четырехядерным процессором — от четырехпроцессорного. Этот подход позволяет избежать многих технологических проблем, связанных с повышением тактовых частот, и создавать при этом более производительные процессоры.

Все это прекрасно, но если ваше приложение не будет использовать несколько ядер, его быстродействие никак не изменится. Именно здесь и вступает в игру технология OpenMP, которая помогает программистам на C++ быстрее создавать многопоточные приложения.

Стандарт OpenMP был разработан в 1997 г. как API, ориентированный на написание портируемых многопоточных приложений. Сначала он был основан на языке Fortran, но позднее включил в себя и C/C++. Последняя версия OpenMP — 2.0;  ее полностью поддерживает Visual C++ 2005. Стандарт OpenMP поддерживается и платформой Xbox 360.

в Visual C++ 2005 параметр компилятора /openmp. (Вы можете активизировать директивы OpenMP на страницах свойств проекта, выбрав Configuration Properties, C/C++, Language и изменив значение свойства OpenMP Support.) Встретив параметр /openmp, компилятор определяет символ \_OPENMP, с помощью которого можно выяснить, включены ли средства OpenMP. Для этого достаточно написать #ifndef \_OPENMP.

OpenMP связывается с приложениями через библиотеку импорта vcomp.lib. Соответствующая библиотека периода выполнения называется vcomp.dll. Отладочные версии библиотек импорта и периода выполнения (vcompd.lib и vcompd.dll соответственно) поддерживают дополнительные сообщения об ошибках, генерируемых при некоторых недопустимых операциях. Имейте в виду, что Visual C++ не поддерживает статическое связывание с библиотекой OpenMP периода выполнения, хотя в версии для Xbox 360 это поддерживается.

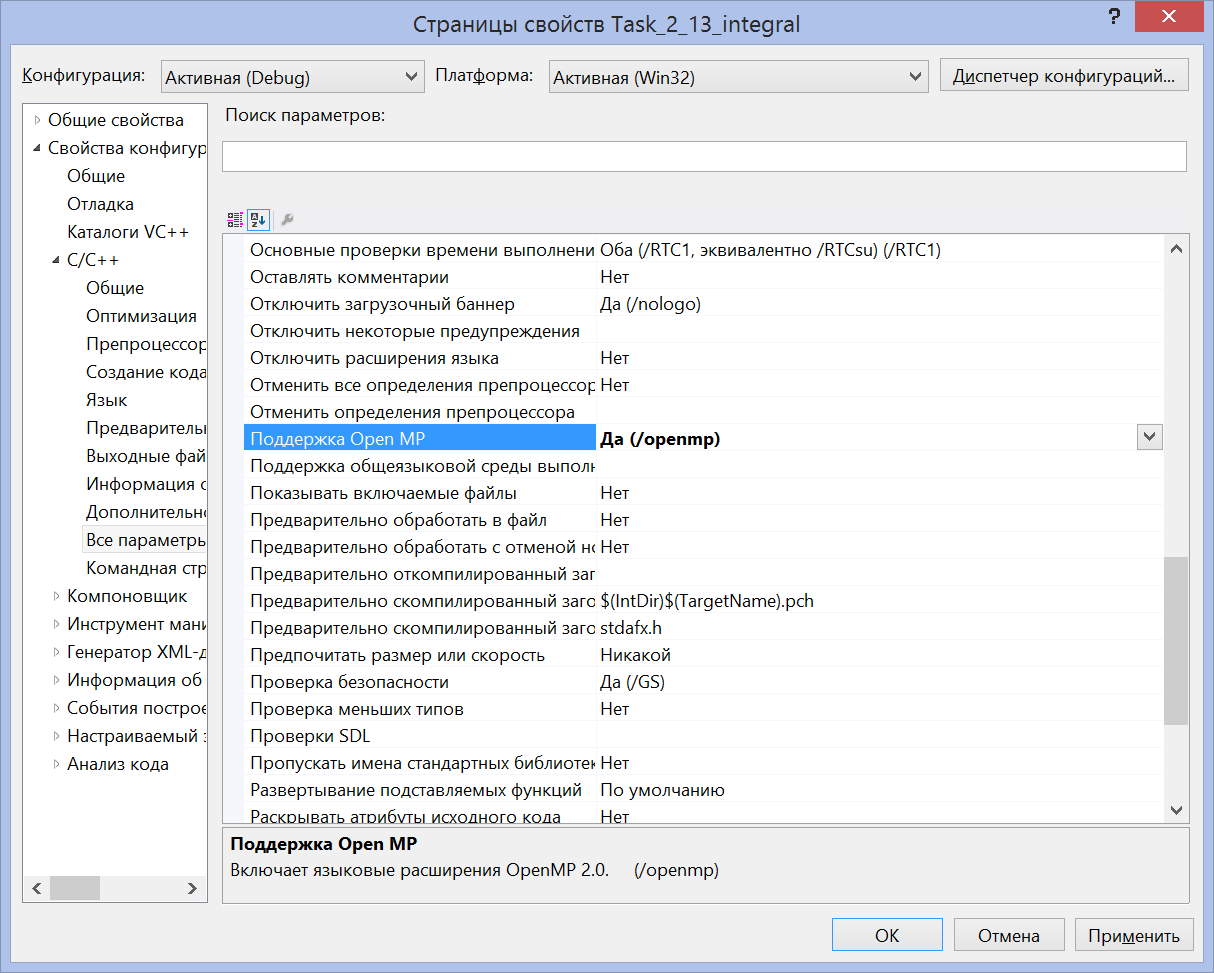
Работа OpenMP-приложения начинается с единственного потока — основного. В приложении могут содержаться параллельные регионы, входя в которые, основной поток создает группы потоков (включающие основной поток). В конце параллельного региона группы потоков останавливаются, а выполнение основного потока продолжается. В параллельный регион могут быть вложены другие параллельные регионы, в которых каждый поток первоначального региона становится основным для своей группы потоков. Вложенные регионы могут в свою очередь включать регионы более глубокого уровня вложенности.

OpenMP прост в использовании и включает лишь два базовых типа конструкций: директивы pragma и функции исполняющей среды OpenMP. Директивы pragma, как правило, указывают компилятору реализовать параллельное выполнение блоков кода. Все эти директивы начинаются с #pragma omp. Как и любые другие директивы pragma, они игнорируются компилятором, не поддерживающим конкретную технологию — в данном случае OpenMP.

Функции OpenMP служат в основном для изменения и получения параметров среды. Кроме того, OpenMP включает API-функции для поддержки некоторых типов синхронизации. Чтобы задействовать эти функции библиотеки OpenMP периода выполнения (исполняющей среды), в программу нужно включить заголовочный файл omp.h. Если вы используете в приложении только OpenMP-директивы pragma, включать этот файл не требуется.

Для реализации параллельного выполнения блоков приложения нужно просто добавить в код директивы pragma и, если нужно, воспользоваться функциями библиотеки OpenMP периода выполнения.

OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic, которые определяют или механизмы разделения работы или конструкции синхронизации.



# Вычисление с заданной точностью итерационным численным методом

**Численное интегрирование** (историческое название: *(численная)*[***квадратура***](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))) — вычисление значения [определённого интеграла](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D0%BB) (как правило, приближённое). Под численным интегрированием понимают набор [численных методов](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B) для нахождения значения определённого интеграла.

Численное интегрирование применяется, когда:

1. Сама подынтегральная функция не задана аналитически. Например, она представлена в виде таблицы (массива) значений в узлах некоторой расчётной сетки.
2. Аналитическое представление подынтегральной функции известно, но её первообразная не выражается через аналитические функции. Например, f(x) = \exp(-x^2).

В этих двух случаях невозможно вычисление интеграла по [формуле Ньютона — Лейбница](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0_%D0%9D%D1%8C%D1%8E%D1%82%D0%BE%D0%BD%D0%B0_%E2%80%94_%D0%9B%D0%B5%D0%B9%D0%B1%D0%BD%D0%B8%D1%86%D0%B0). Также возможна ситуация, когда вид первообразной настолько сложен, что быстрее вычислить значение интеграла численным методом.

Основная идея большинства методов численного интегрирования состоит в замене подынтегральной функции на более простую, интеграл от которой легко вычисляется аналитически. При этом для оценки значения интеграла получаются формулы вида

I \approx \sum_{i=1}^{n} w_i\, f(x_i),

где n\,\! — число точек, в которых вычисляется значение подынтегральной функции. Точки x_i\,\! называются узлами метода, числа w_i\,\! — весами узлов. При замене подынтегральной функции на полином нулевой, первой и второй степени получаются соответственно методы [прямоугольников](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D1%8F%D0%BC%D0%BE%D1%83%D0%B3%D0%BE%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA), [трапеций](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BF%D0%B5%D1%86%D0%B8%D1%8F) и [парабол](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B1%D0%BE%D0%BB%D0%B0) (Симпсона). Часто формулы для оценки значения интеграла называют квадратурными формулами.

### Метод прямоугольников

Пусть требуется определить значение интеграла функции на отрезке \left[ {a},{b} \right]. Этот отрезок делится точками x_0, x_1, \ldots, x_{n-1}, x_n на n\,\! равных отрезков длиной \Delta {x} = \frac{b-a}{n}. Обозначим через y_0, y_1, \ldots, y_{n-1}, y_n значение функции f\left(x\right) в точках x_0, x_1, \ldots, x_{n-1}, x_n. Далее составляем суммы y_0 \,\Delta {x} + y_1 \,\Delta {x} + \ldots + y_{n-1} \,\Delta {x}. Каждая из сумм — интегральная сумма для f\left(x\right) на \left[ {a},{b} \right] и поэтому приближённо выражает интеграл

\int\limits_a^b f(x)\,dx \approx \frac{b-a}{n} (y_0 + y_1 + \ldots + y_{n-1}).

Равномерную сетку можно описать следующим набором формул:

x_i = a + i h, \qquad h = \frac{b - a}{n},

где h\,\! — шаг сетки.

Для равномерных сеток формулы прямоугольников можно записать в виде следующих *формул Котеса*:

1. Составная формула **левых прямоугольников**: \int^b_a f(x)\,dx \approx h \sum_{i=0}^{n-1} f_i = h (f_0 + f_1 + \ldots + f_{n-1}).
2. Составная формула **правых прямоугольников**: \int^b_a f(x)\,dx \approx h \sum_{i=1}^{n} f_i = h (f_1 + f_2 + \ldots + f_{n}).
3. Составная формула **средних прямоугольников** если нет возможности изменять точки в которых вычисляется значение функции выглядит так \int^b_a f(x)\,dx \approx h (\frac{f_0}{2} + f_1 + \ldots + f_{n-1}+\frac{f_{n}}{2}). Т.е. (с точностью до рисунка) превращается в формулу трапеций. Если же есть возможность выбирать точки в которых задано значение функции, то [[1]](http://onosan.narod.ru/gos_exams/17.pdf) \int^b_a f(x)\,dx \approx h \sum_{i=1}^{n} f(x_i-h/2) .

## Погрешность метода

Для формул правых и левых прямоугольников погрешность составляет

E(f) = \frac{f'(\xi)}{2n} (b - a)^2.

**Правило Рунге** — правило оценки погрешности [численных методов](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B), было предложено [К. Рунге](https://ru.wikipedia.org/wiki/%D0%A0%D1%83%D0%BD%D0%B3%D0%B5,_%D0%9A%D0%B0%D1%80%D0%BB) в начале 20 века.[[1]](https://ru.wikipedia.org/wiki/%CF%F0%E0%E2%E8%EB%EE_%D0%F3%ED%E3%E5#cite_note-1)

Основная идея (для [методов Рунге-Кутты](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%A0%D1%83%D0%BD%D0%B3%D0%B5_%E2%80%94_%D0%9A%D1%83%D1%82%D1%82%D1%8B) решения [ОДУ](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8B%D0%BA%D0%BD%D0%BE%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5)) состоит в вычислении приближения выбранным методом с шагом h, а затем с шагом h/2, и дальнейшем рассмотрении разностей погрешностей для этих двух вычислений.

## Применение правила Рунге

### Оценка точности вычисления определённого интеграла

Интеграл вычисляется по выбранной формуле (прямоугольников, трапеций, парабол Симпсона) при числе шагов, равном n, а затем при числе шагов, равном 2n. Погрешность вычисления значения интеграла при числе шагов, равном 2n, определяется по формуле Рунге:  
\Delta_{2n} \approx \Theta |I_{2n}-I_{n}|, для формул прямоугольников и трапеций \Theta = \frac{1}{3}, а для формулы Симпсона \Theta = \frac{1}{15}.[[2]](https://ru.wikipedia.org/wiki/%CF%F0%E0%E2%E8%EB%EE_%D0%F3%ED%E3%E5#cite_note-2)

Таким образом, интеграл вычисляется для последовательных значений числа шагов N = n_0,2n_0,4n_0,\dots, где n0 — начальное число шагов. Процесс вычислений заканчивается, когда для очередного значения N будет выполнено условие \Delta_{2n} < \epsilon, где ε — заданная точность.

Программа работает в 2-х режимах:

* с выводом результатов работы на экран (для демонстрации работоспособности);
* без вывода результатов работы на экран, но с определением времени, затрачиваемого на вычисления;
* double e=1e-7; // Точность
* double f(double x) // Функция
* {
* return sin(x);
* }

Листинг кода

#include "Header.hpp"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Метод левых прямоугольников

метод численного интегрирования функции одной переменной,

заключающийся в замене подынтегральной функции на многочлен нулевой степени,

то есть константу, на каждом элементарном отрезке.

Если рассмотреть график подынтегральной функции, то метод будет заключаться в приближённом вычислении

площади под графиком суммированием площадей конечного числа прямоугольников,

ширина которых будет определяться расстоянием между соответствующими соседними узлами

интегрирования, а высота — значением подынтегральной функции в этих узлах.

Для формул правых и левых прямоугольников погрешность составляет

E(f) = f'/2n \* (b - a)^2.

Правило Рунге — правило оценки погрешности численных методов,

было предложено К. Рунге в начале 20 века.[1]

Основная идея (для методов Рунге-Кутты решения ОДУ) состоит в вычислении приближения

выбранным методом с шагом h, а затем с шагом h/2, и дальнейшем рассмотрении разностей

погрешностей для этих двух вычислений.

Интеграл вычисляется по выбранной формуле (прямоугольников, трапеций, парабол Симпсона)

при числе шагов, равном n, а затем при числе шагов, равном 2n.

Погрешность вычисления значения интеграла при числе шагов, равном 2n,

определяется по формуле Рунге:

Delta\_{2n} = Theta \* |I\_{2n}-I\_{n}|, для формул прямоугольников и трапеций Theta = 1/3

\*pf - указатель на функцию

a,b - диапазон интегрирования

е - точность

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double integral(double (\*pf)(double), double a, double b, double e, int demo){

int nmin=10; // Минимальное число разбиения отрезка [a,b]

double prev;

double next;

// Вычисление начального приближения

prev=0;

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

prev+=(\*pf)(a+i\*(b-a)/nmin);

prev /= nmin;

// Итерации

for(int n=nmin;;n++) {

if(demo!=0) cout << "\tШаг итерации:" << n << endl;

if(demo!=0) cout << "\tПредыдущее значение:" << prev << endl;

double \*y=new double[n];

// используется параллельная обработка элементов массива

// Однородные вычислительные структуры или среды (ОВС), как правило,

// относятся к классу ОКМД (согласно классификации Флинна)

// и представляют собой регулярную структуру из однотипных процессорных элементов (ПЭ).

// Каждый ПЭ, в зависимости от типа ОВС, может как обладать алгоритмически полным набором

// операций, так и реализовывать один вид операций, жестко заданный в структуре микросхемы

// на этапе проектирования, а также выполнять операции обмена или взаимодействия с другими ПЭ.

// OpenMP поддерживает директивы parallel, for, parallel for, section, sections, single, master, critical, flush, ordered и atomic,

// которые определяют или механизмы разделения работы или конструкции синхронизации.

// Общие переменные доступны всем потокам из группы, поэтому изменения таких переменных в одном потоке видимы другим потокам

// в параллельном регионе. Что касается частных переменных, то каждый поток из группы располагает их отдельными экземплярами,

// поэтому изменения таких переменных в одном потоке никак не сказываются на их экземплярах,

// принадлежащих другим потокам.

// частными являются индексы параллельных циклов for.

#pragma omp parallel for

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

y[i]=(\*pf)(a+(2\*i+1)\*(b-a)/(2\*n));

// Суммирование элементов требует дополнительных обработок для параллельных вычислений

// Поэтому используется непараллельная версия

next=0;

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

next+=y[i];

delete y;

next = (next / (2\*n)) + (prev/2);

if(demo!=0) cout << "\tСледующее значение:" << next << endl;

// Правило Рунге — правило оценки погрешности численных методов

double delta=abs(next-prev)/3;

if(demo!=0) cout << "\tИзменение значения:" << delta << endl;

if(demo!=0) {

cout << "\tНажмите ВВОД для продолжения" << endl;

getchar();

getchar();

}

if(delta < e) break;

prev=next;

}

return next;

}

#include "Header.hpp"

using namespace std;

// 13. Вычислить интеграл с заданной точностью любым итерационным численным методом (указать, каким именно).

// Вычисление производится методом последовательных приближений.

// Входные данные: числа a и b, функция f(x) задается в тексте программы, точность е.

double e=1e-7; // Точность

double f(double x) // Функция

{

return sin(x);

}

int main()

{

double a,b;

double s;

int demo; // режим программы

setlocale(LC\_ALL, "Russian");

cout << "\tЗадача вычислить интеграл итерационным численным методом" << endl;

cout << "\tВведите диапазон интегрирования:" << endl; cin >> a >> b;

cout << "\tДемонстрационный режим програмы (0 - замер времени, 1 - демонстрация работоспособности):" << endl; cin >> demo;

clock\_t t = clock();

s=integral(f, a, b, e, demo);

t = clock()-t;

cout << "\tЗначение интеграла:" << s << endl;

double seconds = ((double)t)/CLOCKS\_PER\_SEC;

cout << "\tВремя выполнения: " << seconds << "sec" << endl;

getchar();

getchar();

return 0;

}

# Контрольные примеры работы программы

Пример 1.

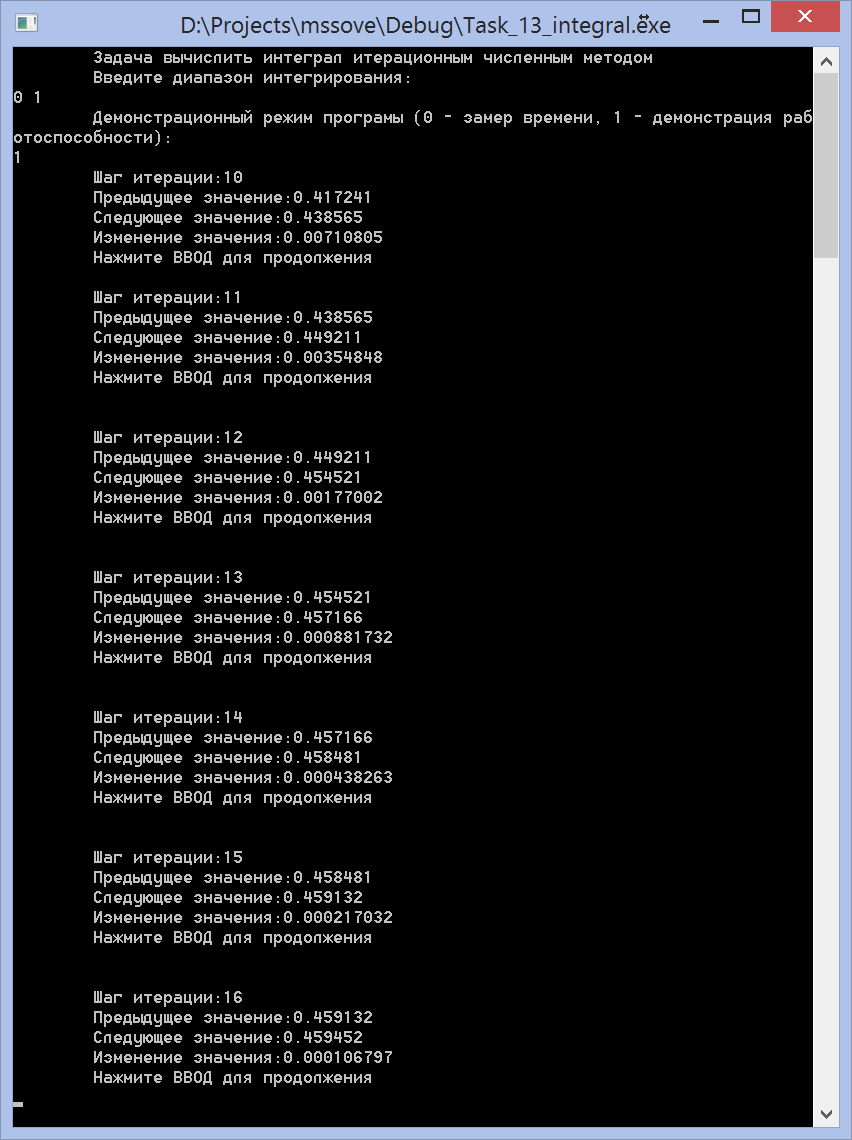
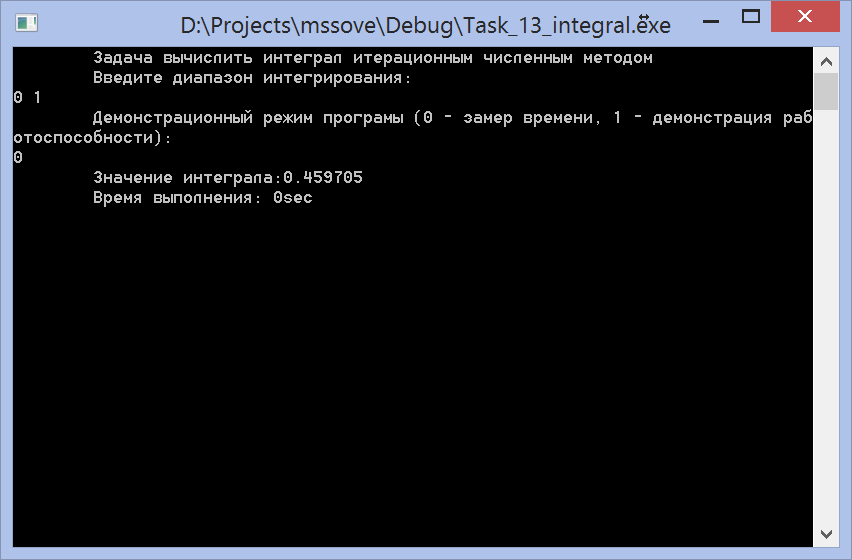


Рис.1..

Пример 2.



# Заключение

В ходе данной лабораторной был изучены методов численного интегрирования. Была сделана программная реализация данного алгоритма. Для данной программной реализации были проведены ряд тестов, показывающие правильность работы алгоритма.