Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Факультет информационных технологий математики и механики

**Отчет по лабораторной работе**

**Интегрирование методом трапеций**

**с использованием параллельных вычислений**

**Выполнил**:студент группы 381606-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кутовой В.Н.

Подпись

**Научный руководитель**:

к.т.н.,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кустикова В.Д

Подпись

Нижний Новгород

2018

Содержание

[Введение 3](#_Toc532824248)

[Постановка задачи 4](#_Toc532824249)

[Метод решения 5](#_Toc532824250)

[Схема распараллеливания 7](#_Toc532824251)

[Проверка корректности 8](#_Toc532824252)

[Эксперименты 9](#_Toc532824253)

[Руководство пользователя 11](#_Toc532824254)

[Заключение 12](#_Toc532824255)

[Приложение 13](#_Toc532824256)

# Введение

**Интеграл** — одно из важнейших понятий математического анализа, которое возникает при решении задач о нахождении площади под кривой, пройденного пути при неравномерном движении, массы неоднородного тела, и тому подобных, а также в задаче о восстановлении функции по её производной (неопределённый интеграл). Упрощённо интеграл можно представить как аналог суммы для бесконечного числа бесконечно малых слагаемых. В зависимости от пространства, на котором задана подынтегральная функция, интеграл может быть — двойной, тройной, криволинейный, поверхностный и так далее; также существуют разные подходы к определению интеграла — различают интегралы Римана, Лебега, Стилтьеса и другие.

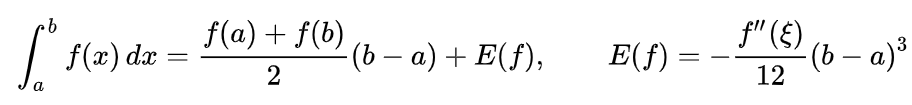
# Постановка задачи

Необходимо разработать параллельную программу, которая будет вычислять многомерные интегралы, используя метод трапеций. Данная программа должна использовать одну из библиотек MPI. Также необходимо провести сравнение параллельной и последовательной реализации метода интегрирования и проанализировать полученные результаты, проверить корректность вычислений реализованной программы.

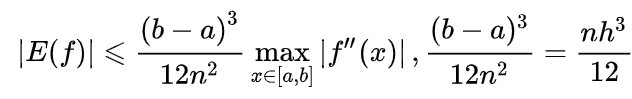
# Метод решения

**Метод трапеций** — метод численного интегрирования функции одной переменной, заключающийся в замене на каждом элементарном отрезке подынтегральной функции на многочлен первой степени, то есть линейную функцию. Площадь под графиком функции аппроксимируется прямоугольными трапециями. Алгебраический порядок точности равен 1.

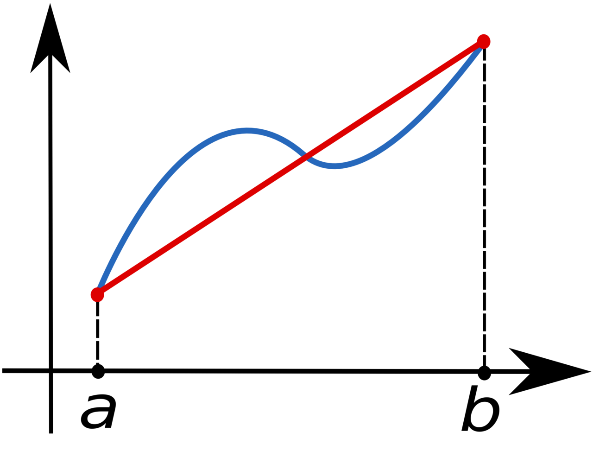
Если отрезок {\displaystyle \left[a,b\right]} является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле

{\displaystyle \int \_{a}^{b}f(x)\,dx={\frac {f(a)+f(b)}{2}}(b-a)+E(f),\qquad E(f)=-{\frac {f''(\xi )}{12}}\left(b-a\right)^{3}.} 

Это простое применение формулы для площади трапеции — произведение полусуммы оснований, которыми в данном случае являются значения функции в крайних точках отрезка, на высоту (длину отрезка интегрирования). Погрешность аппроксимации можно оценить через максимум второй производной

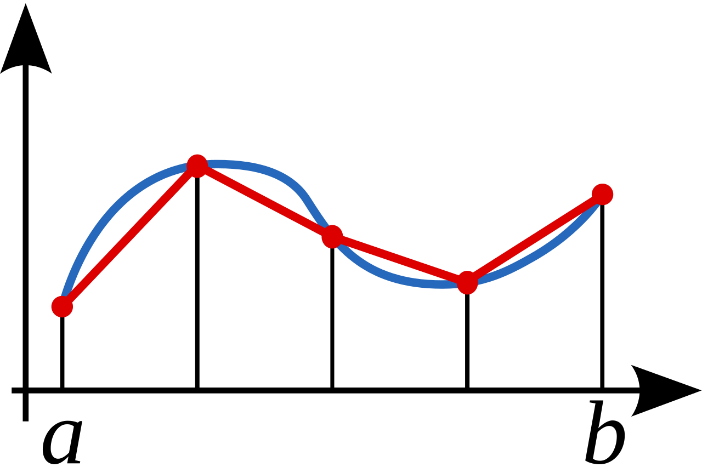
{\displaystyle \left|E(f)\right|\leqslant {\frac {\left(b-a\right)^{3}}{12n^{2}}}\max \_{x\in [a,b]}\left|f''(x)\right|,{\frac {(b-a)^{3}}{12n^{2}}}={\frac {nh^{3}}{12}}.} 

На рисунке показана аппроксимация функции линейной зависимостью при интегрировании методом трапеций.

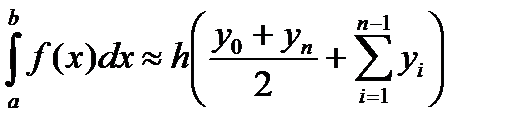


Для реализации программы имеет смысл использовать сетку с равномерным шагом, так как вычисления будут производиться с относительно небольшим шагом и погрешность аппроксимации будет незначительной.

Так будет выглядеть разбиение функции одной переменной в реализации программы:



Для вычислений можно использовать **формулу Котеса:**



Для двумерного случая вычисления интеграла будут производится аналогичным способом, только строится будет “трехмерная трапеция”, объем которой будет вычислен.

# Схема распараллеливания

Очевидно, что для распараллеливания вычислений и увеличении скорости работы программы нужно разбить отрезки интегрирования между процессами, затем собрать полученные результаты и просуммировать. Очевидно, что необходимо разделить длину отрезка интегрирования на количество процессов, соответственно, если в вычислениях задействован один процесс, то получим результат выполнения последовательной программы.

Алгоритм:

1. Определение отрезка интегрирования каждого процесса :

(a – левый предел интегрирования, b – правый предел интегрирования, size – количество процессов )

1. Вычисление каждым процессом локальных границ интегрирования:

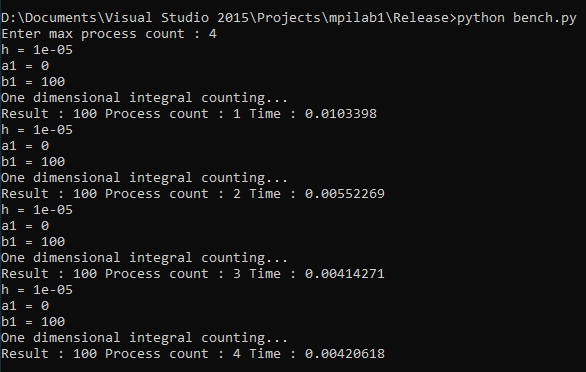
(rank – ранг процесса)

1. Вычисление локальных интегралов на каждом процессе
2. Сбор результатов на нулевом процессе и их суммирование.

При вычислении двумерного интеграла, разбиение производится также и каждый процесс будет вычислять объемы не только от левой до правой локальной границы интегрирования по оси X, но и от левой до правой границы интегрирования по Y.

# Проверка корректности

Для проверки корректности вычислений были проведены тестовые запуски вычисления значений площади прямоугольника со сторонами 1 и 100.



Полученные результаты совпадают с ожидаемыми результатами для разного количества используемых процессов.

# Эксперименты

Тестовые запуски программы производились на четырехъядерном процессоре Intel Core i5 без технологии Hyper Threading.

Шаг интегрирования , , :

Шаг интегрирования **0.01** , , :

Из полученных результатов видно, что производительность программы перестает расти при числе процессов больше четырех. Очевидно, что это связано с количеством ядер процессора, но в тоже время мы можем заметить на графике, что каждое увеличение числа процессов дает все меньший и меньший прирост. Если обратить внимание на время выполнения программы для трех и четырех процессов, то уже можно заметить относительно близкое по меркам человека время выполнения. Связано это с тем, что при увеличении числа процессов требуется больше времени на передачу и сбор данных, но стоит отметить, что увеличение вычислительных объемов будет увеличивать разрыв между разным числом используемых процессов. Это хорошо заметно при вычислении двумерного интеграла. Стоит отметить, что разница времени выполнения между последовательной реализацией и параллельной, запущенной на четырех процессах довольно велика. В эксперименте с вычислением одномерного интеграла получаем уменьшение времени выполнения почти в 3 раза, а в эксперименте с двумерным интегралом более чем в 3.5 раза.

# Руководство пользователя

mpiintegrate.exe [ОПЦИИ]

**Опции:**

**-a1**

левая граница интегрирования по оси X, если не указано, то устанавливается значение по умолчанию.

**-b1**

правая граница интегрирования по оси X, если не указано, то устанавливается значение по умолчанию.

**-a2**

левая граница интегрирования по оси X, если не указано, то вычисляется одномерный интеграл.

**-b2**

правая граница интегрирования по оси X, если не указано, то вычисляется одномерный интеграл.

**-h, --height**

шаг интегрирования.

**-v, --verbose**

вывод в консоль дополнительной информации в ходе выполнения программы.

**-log**

запись результатов в лог.

# Заключение

По результатам экспериментов можно сделать вывод о эффективной параллельной реализации программы для вычисления интегралов методом трапеций. Использование большого числа процессов может значительно ускорить вычисления при больших объемах вычислений. Существенным ограничением является только колчиство ядер процессора. Мне успешно удалось выполнить поставленную задачу.

# Приложение

#include "mpi.h"

#include <iostream>

#include <fstream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <Windows.h>

#include <algorithm>

#include <math.h>

using namespace std;

// sin(x) \* cos (y)

double integral\_2D(const double a1, const double b1, const double a2, const double b2, const double h)

{

double sum;

double midval;

sum = 0.0;

for (double i = a1; i < b1; i += h)

{

for (double j = a2; j < b2; j += h)

{

midval = ((sin(i \* j) \* cos(i \* j)) +

(sin((i + h) \* j) \* cos((i + h) \* j)) +

(sin(i \* (j + h)) \* cos(i \* (j + h))) +

(sin((i + h) \* (j + h)) \* cos((i + h) \* (j + h)))) / 4;

sum += midval \* h \* h;

}

}

return sum;

}

double f(double x)

{

return sin(x);

}

double integral\_1D(const double a1, const double b1, const double h)

{

double sum = 0;

double hl = 0, hr = 0;

for (double i = a1; i < b1; i += h)

{

hl = f(i);

hr = f(i + h);

sum += (hl + hr) / 2 \* h;

}

return sum;

}

char\* getCmdOption(char \*\* begin, char \*\* end, const std::string & option)

{

char \*\* itr = std::find(begin, end, option);

if (itr != end && ++itr != end)

{

return \*itr;

}

return 0;

}

bool cmdOptionExists(char\*\* begin, char\*\* end, const std::string& option)

{

return std::find(begin, end, option) != end;

}

int main(int argc, char \*argv[])

{

int procRank, procNum;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &procNum);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &procRank);

double starttime, endtime;

double a1 = 0.0;

double b1 = 16.0;

double a2 = INFINITY;

double b2 = INFINITY;

double h = 0.1;

double step;

double res = 0.0;

double \*ressivedres = NULL;

bool verbose = false;

bool log = false;

if (cmdOptionExists(argv, argv + argc, "-a1"))

{

char \* wcount = getCmdOption(argv, argv + argc, "-a1");

a1 = atof(wcount);

}

if (cmdOptionExists(argv, argv + argc, "-b1"))

{

char \* wcount = getCmdOption(argv, argv + argc, "-b1");

b1 = atof(wcount);

}

if (cmdOptionExists(argv, argv + argc, "-a2"))

{

char \* wcount = getCmdOption(argv, argv + argc, "-a2");

a2 = atof(wcount);

}

if (cmdOptionExists(argv, argv + argc, "-b2"))

{

char \* wcount = getCmdOption(argv, argv + argc, "-b2");

b2 = atof(wcount);

}

if (cmdOptionExists(argv, argv + argc, "-h"))

{

char \* wcount = getCmdOption(argv, argv + argc, "-h");

h = atof(wcount);

}

if (cmdOptionExists(argv, argv + argc, "-v") || cmdOptionExists(argv, argv + argc, "--verbose"))

{

verbose = true;

}

if (cmdOptionExists(argv, argv + argc, "-log"))

{

log = true;

}

if (procRank == 0 && verbose)

{

if (log) cout << "Logging into log.txt" << endl;

cout << "h = " << h << endl;

cout << "a1 = " << a1 << endl;

cout << "b1 = " << b1 << endl;

if (a2 != INFINITY && b2 != INFINITY) {

cout << "a2 = " << a2 << endl;

cout << "b2 = " << b2 << endl;

cout << "Two dimensional integral counting..." << endl;

}

else

cout << "One dimensional integral counting..." << endl;

}

MPI\_Barrier(MPI\_COMM\_WORLD);

if (procRank == 0)

starttime = MPI\_Wtime();

step = (b1 - a1) / (double)procNum;

double \*X;

X = new double[2 \* procNum];

if (a2 == INFINITY || b2 == INFINITY)

{

res = integral\_1D(procRank \* step, procRank \* step + step, h);

}

else

{

if (procRank == 0)

{

// double tmp = (b1 - a1) / (double)procNum;

// tmp = round(tmp \* 1000) / 1000;

// X[2] = a1;

// X[3] = a1 + tmp;

// for (int i = 4; i < procNum \* 2; i += 2)

// {

// X[i] = X[i - 1];

// X[i + 1] = X[i] + tmp;

// }

// X[0] = X[procNum \* 2 - 1];

// X[1] = b1;

//}

//double X\_loc[2];

//

//MPI\_Scatter(X, 2, MPI\_DOUBLE, X\_loc, 2, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

res = integral\_2D(procRank \* step, procRank \* step + step, a2, b2, h);

}

}

if (procRank == 0) {

ressivedres = new double[procNum];

}

MPI\_Gather(&res, 1, MPI\_DOUBLE, ressivedres, 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

if (procRank == 0)

{

endtime = MPI\_Wtime();

for (int i = 1; i < procNum; i++)

res += ressivedres[i];

res -= h;

double estimated\_time = endtime - starttime;

cout << "Result : " << res << " Process count : " << procNum << " Time : " << estimated\_time << endl;

if (verbose) {

fstream log;

log.open("log.txt", ios::out | ios::app);

//log << "Result : " << res << " Process count : " << procNum << " Time : " << estimated\_time << endl;

log << procNum << " " << estimated\_time << endl;

log.close();

}

}

MPI\_Finalize();

return 0;

}