ФГБОУ ВО Национальный Исследовательский Университет «МЭИ»

Институт Автоматики и Вычислительной Техники

Кафедра Прикладной Математики

**Курсовой проект**

по дисциплине «Параллельное программирование»

на тему «Интегрирование функций»

Выполнила: Закладная С.В.

Группа: А-13м-16

Преподаватель: Кутепов В.П.

Москва, 2018

**Содержание**

[Введение 3](#_Toc504100625)

[Постановка задачи 4](#_Toc504100626)

[Алгоритмы и методы 4](#_Toc504100627)

[Инструментальные средства 6](#_Toc504100628)

[Результаты 7](#_Toc504100629)

[*Пример 1* 7](#_Toc504100630)

[*Пример 1а* 9](#_Toc504100631)

[*Пример 2* 10](#_Toc504100632)

[Заключение 12](#_Toc504100633)

[Список литературы 13](#_Toc504100634)

[Приложение 14](#_Toc504100635)

[*Распределение по потокам* 14](#_Toc504100636)

[*Рекурсивный алгоритм интегрирования* 15](#_Toc504100637)

[*Итеративный алгоритм интегрирования* 15](#_Toc504100638)

[*Метод прямоугольников* 15](#_Toc504100639)

[*Метод трапеций* 16](#_Toc504100640)

[*Корректировка разбиения* 16](#_Toc504100641)

# **Введение**

Задача интегрирования функций является одной из фундаментальных задач математического анализа и имеет множество приложений в различных областях науки и техники, в том числе в системах реального времени. По этой причине огромное значение имеют разработка и совершенствование вычислительных методов интегрирования с точки зрения оптимизации временных характеристик.

В данной работе рассматривается параллельная реализация различных методов интегрирования функций.

# **Постановка задачи**

Разработать и исследовать на многоядерных компьютерах оптимальные алгоритмы интегрирования функций.

# **Алгоритмы и методы**

Для исследования были выбраны следующие методы интегрирования:

* Метод прямоугольников:

Площадь элементарной криволинейной трапеции заменяется площадью прямоугольника, основанием которого является отрезок , а высота равна значению . Для случая постоянного шага квадратурная формула имеет вид:

где . [1]

* Метод трапеций:

Площадь элементарной криволинейной трапеции заменяется площадью трапеции, построенной путём соединения отрезком точек и . Для равномерной сетки формула трапеций имеет вид:

На одном потоке значение интеграла с точностью вычисляется по следующему алгоритму:

S1 = I(a, b, N);

N = N \* 2;

S2 = I(a, b, N);

while (|S1 – S2|>)

{

S1 = S2;

N = N \* 2;

S2 = I(a, b, N);

}

Здесь I(a, b, N) – интегральная сумма, вычисленная по формуле (1) или (2). В качестве конечного результата принимается значение S2.

На *K* потоках значение интеграла представляет собой сумму интегралов по отрезкам , вычисленных с точностью :

Разбиение отрезка интегрирования между потоками может осуществляться двумя способами:

* Равномерно

В этом случае отрезок [a, b] делится поровну на K частей.

* Неравномерно

При неравномерном разбиении длины отрезков , полученные в результате равномерного разбиения, корректируются в соответствии с величиной производной подынтегральной функции на данном участке. Производная оценивается через отношение приращения значения функции на отрезке к длине отрезка . Корректировка выполняется по следующему алгоритму:

* + Шаг 1: выполняется поиск отрезка разбиения с максимальным абсолютным значением , длина которого не менее , где – точность вычисления интеграла на данном отрезке.
  + Шаг 2: если такой отрезок найден, уменьшаем его длину вдвое, пока абсолютное значение не станет меньше выбранного пользователем параметра *М*, либо ; иначе – завершаем работу алгоритма.
  + Шаг 3: изменяем границы соседних с изменённым отрезков, чтобы они стыковались друг с другом, после чего переходим к шагу 2.

Код программной реализации описанных методов на языке C# - см. Приложение.

# **Инструментальные средства**

В качестве средств реализации был выбран язык программирования C# и платформа .NET 4.5. Выбор сделан в пользу данных средств поскольку .Net Framework предоставляет достаточно удобный и эффективный инструментарий для создания многопоточных приложений. С подробной документацией средств работы с потоками можно ознакомиться в [2].

# **Результаты**

Тестирование реализованных методов выполнялось на процессоре Intel(R) Core(TM) i5-3570 CPU (тактовая частота 3.40 GHz, ядер: 4).

## *Пример 1*

Рассмотрим задачу вычисления интеграла:

с точностью .

График подынтегральной функции изображен на рисунке 1:

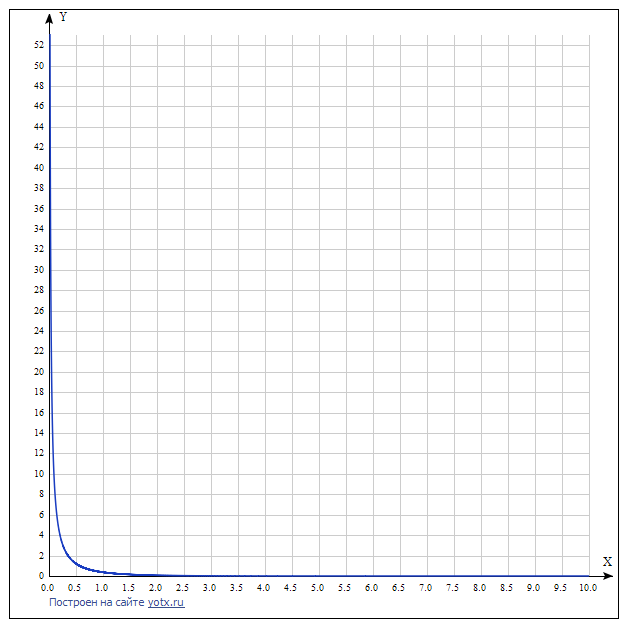


Рисунок 1. График подынтегральной функции (пример 1)

Найдём значение интеграла (4) методом прямоугольников и методом трапеций, используя для каждого из них как равномерное, так и неравномерное разбиение отрезка интегрирования. Зависимость времени вычисления от количества потоков представлена на рисунках 2 и 3.

Рисунок 2. Пример 1 (метод прямоугольников)

Рисунок 3. Пример 1 (метод трапеций)

## *Пример 1а*

Повторим эксперимент из примера 1 с равномерной сеткой для рекурсивной версии алгоритма. Зависимость времени вычисления от количества потоков представлена на рисунках 4 и 5.

Рисунок 4. Пример 1а (метод прямоугольников)

Рисунок 5. Пример 1а (метод трапеций)

## *Пример 2*

Рассмотрим задачу вычисления интеграла:

с точностью .

График подынтегральной функции изображен на рисунке 6:

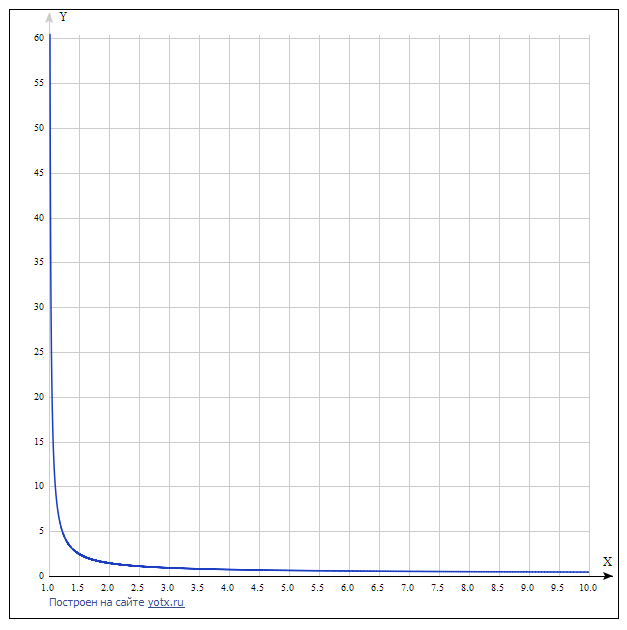


Рисунок 6. График подынтегральной функции (пример 2)

Найдём значение интеграла (5) методом прямоугольников и методом трапеций, используя для каждого из них как равномерное, так и неравномерное разбиение отрезка интегрирования. Зависимость времени вычисления от количества потоков представлена на рисунках 7 и 8.

Рисунок 7. Пример 2 (метод прямоугольников)

Рисунок 8. Пример 2 (метод трапеций)

# **Заключение**

В данной работе была рассмотрена параллельная реализация двух методов численного интегрирования и исследованы их временные характеристики при выполнении на многоядерном процессоре.

Задача интегрирования при наличии у подынтегральной функции различных особенностей (точки разрыва, участки резкого возрастания/убывания и т.д.) становится весьма сложной. Распараллеливание вычисления интеграла даёт значительное снижение времени даже при равномерном разбиении отрезка. Введение переменного шага позволяет дополнительно улучшить эти результаты. Однако отрезок интегрирования и оптимальное количество потоков необходимо выбирать в зависимости от задачи, т.к. методы численного интегрирования весьма чувствительны к особенностям подынтегральной функции.

# **Список литературы**

|  |  |
| --- | --- |
| 1. | Амосов А.А., Дубинский Ю.А., Копченова Н.В. Вычислительные методы для инженеров, Москва: Высшая школа, 1994. |
| 2. | «System.Threading Пространство имен,» [В Интернете]. Available: https://msdn.microsoft.com/ru-ru/library/system.threading(v=vs.110).aspx. |

# **Приложение**

## *Распределение по потокам*

public void Distribute(Integral I,ref List<Integral> SubIntegrals,int threadscount, bool regular)

{

if (threadscount < 1)

{

throw new Exception("Неверное число потоков");

}

//Равномерная сетка

if (regular)

{

Threads = new List<Thread>();

//Делим отрезок интегрирования между потоками

double h = (I.UpperLimit - I.LowerLimit) / threadscount;

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

{

Integral intg = new Integral(I.LowerLimit + i \* h, I.LowerLimit + (i + 1) \* h,I.F,I.Eps/threadscount,I.Method);

SubIntegrals.Add(intg);

//Создаём поток и добавляем в список

ThreadStart solver = new ThreadStart(SubIntegrals[i].Solve);

Thread thread = new Thread(solver);

Threads.Add(thread);

}

}

else

{

Threads = new List<Thread>();

double h = (I.UpperLimit - I.LowerLimit) / threadscount;

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

{

Integral intg = new Integral(I.LowerLimit + i \* h, I.LowerLimit + (i + 1) \* h, I.F, I.Eps / threadscount, I.Method);

SubIntegrals.Add(intg);

}

if (threadscount > 1)

{

I.CorrectGrid(ref SubIntegrals, K);

}

foreach(Integral sub in SubIntegrals)

{

ThreadStart solver = new ThreadStart(sub.Solve);

Thread thread = new Thread(solver);

Threads.Add(thread);

}

}

}

## *Рекурсивный алгоритм интегрирования*

public void SolveR()

{

if(k<1)

{

Value = 0;

return;

}

double h = (UpperLimit - LowerLimit) / k;

Integral I = new Integral(LowerLimit + h, UpperLimit, F, Eps, Method,n1,k - 1);

ThreadStart solver = new ThreadStart(I.SolveR);

Thread thread = new Thread(solver);

thread.Start();

Integral I0 = new Integral(LowerLimit, LowerLimit + h, F, Eps/n1, Method);

I0.Solve();

thread.Join();

Value = I.Value + I0.Value;

}

## *Итеративный алгоритм интегрирования*

public void Solve()

{

double S = Method(LowerLimit,UpperLimit,N);

N \*= 2;

Value = Method(LowerLimit, UpperLimit, N);

while (Math.Abs(Value - S)>Eps)

{

S = Value;

N \*= 2;

Value = Method(LowerLimit, UpperLimit, N);

}

}

## *Метод прямоугольников*

private double Rectangle(double a, double b, int n)

{

double S = 0; //результат

double h = (b - a) / n; //шаг

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

{

double xi = x(a,i, h);

double xi1 = x(a,i + 1, h);

S += F((xi + xi1) / 2) \* (xi1 - xi);

}

return S;

}

## *Метод трапеций*

private double Trapezoid(double a, double b, int n)

{

double S = 0; //результат

double h = (b - a) / n; //шаг

S = (F(x(a,0,h))+ F(x(a, n, h))) / 2;

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

{

S += F(x(a, i, h));

}

S \*= h;

return S;

}

## *Корректировка разбиения*

public void CorrectGrid(ref List<Integral> Subs, double K0)

{

if(Subs.Count<2)

{

return;

}

bool found = true; //Найден отрезок с большой производной

while(found)

{

found = false;

//поиск отрезка с максимальной производной

double maxTg = 0;

int imax = -1;

for (int i = 0; i < Subs.Count; i++)

{

double atg = Math.Abs(Tg(Subs[i].LowerLimit, Subs[i].UpperLimit));

if ((atg > maxTg) && ((Subs[i].UpperLimit - Subs[i].LowerLimit) > Subs[i].Eps))

{

maxTg = atg;

imax = i;

}

}

//Сравниваем максимум с параметром корректировки

if (imax >= 0)

{

if ((maxTg > K0) && (Subs[imax].UpperLimit - Subs[imax].LowerLimit) > Subs[imax].Eps\*10)

{

found = true;

bool changed = false;

//Сжимаем отрезок с максимальной производной

while ((Math.Abs(Tg(Subs[imax].LowerLimit, Subs[imax].UpperLimit)) > K0) && (Subs[imax].UpperLimit - Subs[imax].LowerLimit) > Subs[imax].Eps\*10)

{

double middle = (Subs[imax].UpperLimit + Subs[imax].LowerLimit) / 2;

if (Math.Abs(Tg(Subs[imax].LowerLimit, middle)) > Math.Abs(Tg(middle, Subs[imax].UpperLimit)) && (imax < Subs.Count-1))

{

changed = true;

Subs[imax].UpperLimit = middle;

}

else if(imax > 0)

{

changed = true;

Subs[imax].LowerLimit = middle;

}

}

if (changed)

{

//Пристыковываем соседние отрезки

if (imax > 0)

{

Subs[imax - 1].UpperLimit = Subs[imax].LowerLimit;

}

if (imax < Subs.Count - 1)

{

Subs[imax + 1].LowerLimit = Subs[imax].UpperLimit;

}

}

}

}

}

}