

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н. И.
Лобачевского»

Институт информационных технологий математики и механики
Кафедра теоретической, компьютерной и экспериментальной механики

Направление подготовки 09.03.04 Программная инженерия
Направленность (профиль) программы бакалавриата: разработка программно-информационных систем

Вычисление многомерных интегралов с использованием многошаговой схемы (метод Симпсона)

Исполнитель: студент **381708-2** группы

Tappa Noukimi Frank

Руководитель: **Волокитин.**

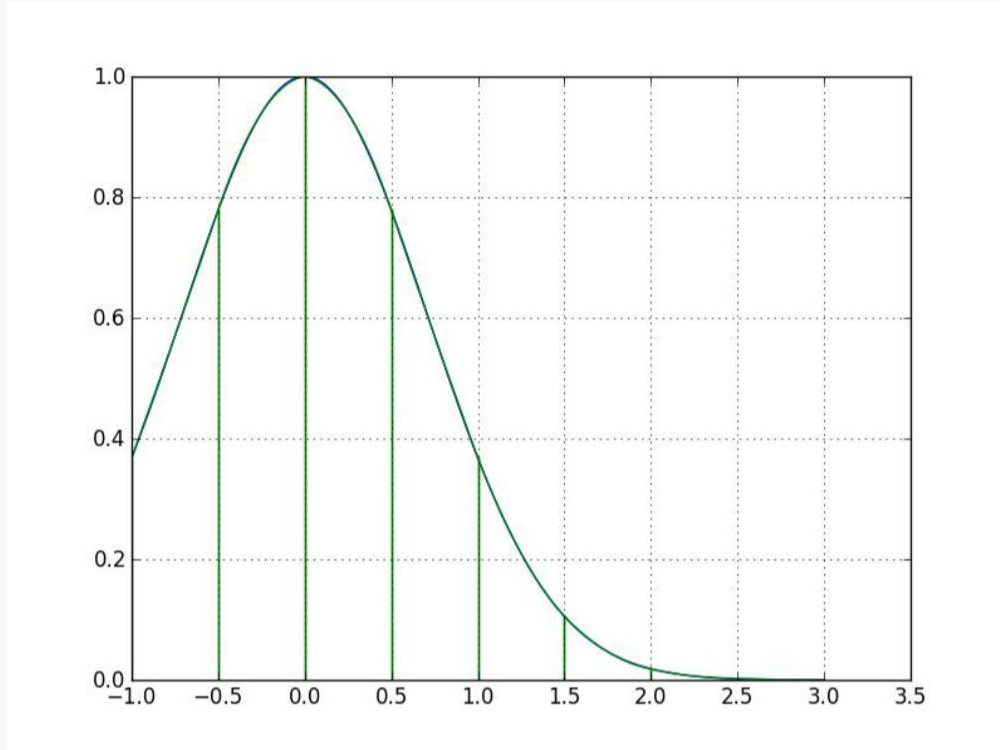
Нижегород, 2020 г.

Оглавление

Принцип метода	3
Расчет интеграла	5
Алгоритм и тесты	8
Приложения	9

Принцип метода

Метод Симпсона состоит в замене функции f на каждом интервале функцией, которая будет триномиальной. Действительно, мы будем приближаться к нему полиномом степени 2, который не проходит точки $M_k(x_k, f(x_k))$, $M_{k+1}(x_{k+1}, f(x_{k+1}))$ и точку в середине $M_{k,5}(x_{k,5}, f(x_{k,5}))$ с $x_{k,5} = \frac{x_k + x_{k+1}}{2}$.



Триномиалы работают.

Кривая функции f практически путается с кривыми функций g_k .

Однако мы все еще не знаем, что наши многочлены. На самом деле существует уникальный многочлен степени 2, который проверяет условия, которые мы ему навязываем. Для его построения мы будем использовать интерполяционные полиномы Лагранжа. Если вы их не знаете, это ничего, мы будем строить наш многочлен шаг за шагом.

Кривая функции f почти перепутана с кривыми функций Q_k .

Тем не менее, мы до сих пор не знаем, что такое наши полиномы. Фактически, существует один многочлен степени 2, который удовлетворяет условиям, которые мы навязываем ему. Для его построения мы будем использовать интерполяционные полиномы Лагранжа. Если вы их не знаете, это не имеет значения, мы будем строить наш полином шаг за шагом.

Давайте повторим то, что мы просим для полинома Q_k .

- Образ x_k ассоциированной полиномиальной функцией должен быть $f(x_k)$.
- Образ $x_{k,5}$ ассоциированной полиномиальной функцией должен быть $f(x_{k,5})$.
- Образ x_{k+1} ассоциированной полиномиальной функцией должен быть $f(x_{k+1})$.

Другие пункты не имеют значения для нас.

Чтобы построить этот многочлен, мы построим три многочлена:

- L_k такие как $L_k(x_k) = 1$, $L_k(x_{k,5}) = 0$ и $L_k(x_{k+1}) = 0$.
- $L_{k,5}$ такие как $L_{k,5}(x_k) = 0$, $L_{k,5}(x_{k,5}) = 1$ и $L_{k,5}(x_{k+1}) = 0$.
- L_{k+1} такие как $L_{k+1}(x_k) = 0$, $L_{k+1}(x_{k,5}) = 0$ и $L_{k+1}(x_{k+1}) = 1$.

Мы строим их так:

$$L_k = \frac{(X - x_{k+1})(X - x_{k,5})}{(x_k - x_{k+1})(x_k - x_{k,5})}.$$
$$L_{k,5} = \frac{(X - x_{k+1})(X - x_k)}{(x_{k,5} - x_{k+1})(x_{k,5} - x_k)}.$$
$$L_{k+1} = \frac{(X - x_k)(X - x_{k,5})}{(x_{k+1} - x_k)(x_{k+1} - x_{k,5})}.$$

Мы можем проверить, эти многочлены делают то, что мы просим их сделать. Например:

$$L_k(x_k) = \frac{(x_k - x_{k+1})(x_k - x_{k,5})}{(x_k - x_{k+1})(x_k - x_{k,5})} = 1.$$
$$L_k(x_{k+1}) = \frac{(x_{k+1} - x_{k+1})(x_{k+1} - x_{k,5})}{(x_k - x_{k+1})(x_k - x_{k,5})} = 0.$$
$$L_k(x_{k,5}) = \frac{(x_{k,5} - x_{k+1})(x_{k,5} - x_{k,5})}{(x_k - x_{k+1})(x_k - x_{k,5})} = 0.$$

Теперь собрать Q_k очень просто:

$$Q_k = f(x_k)L_k + f(x_{k,5})L_{k,5} + f(x_{k+1})L_{k+1}.$$

Мы получаем многочлен, который удовлетворяет наложенным на него ограничениям.

Расчет интеграла

Теперь у нас есть большая часть работы. Мы должны вычислить интеграл от Q_k . И для этого нам нужно вычислить интегралы L_k , L_k , 5 и $L_k + 1$.

Для облегчения записи мы будем ставить для вычисления этих интегралов $i, 5 = m$.

У нас есть:

$$\int_{x_k}^{x_{k+1}} L_k(t) dt = \int_{x_k}^{x_{k+1}} \frac{(t - x_{k+1})(t - x_m)}{(x_k - x_{k+1})(x_k - x_m)} dt.$$

Мы можем вынести константу:

$$\int_{x_k}^{x_{k+1}} L_k(t) dt = \frac{1}{(x_k - x_{k+1})(x_k - x_m)} \int_{x_k}^{x_{k+1}} (t - x_{k+1})(t - x_m) dt.$$

Интегрируя по частям:

$$\int_{x_k}^{x_{k+1}} L_k(t) dt = \frac{1}{(x_k - x_{k+1})(x_k - x_m)} \left(\left[(t - x_m) \frac{(t - x_{k+1})^2}{2} \right]_{x_k}^{x_{k+1}} - \int_{x_k}^{x_{k+1}} \frac{(t - x_{k+1})^2}{2} dt \right).$$

И здесь мы знаем, как интегрировать и поэтому:

$$\int_{x_k}^{x_{k+1}} L_k(t) dt = \frac{-(x_k - x_{k+1})^2(x_k - x_m)}{2(x_k - x_{k+1})(x_k - x_m)} - \frac{1}{2(x_k - x_{k+1})(x_k - x_m)} \times \left[\frac{(t - x_{k+1})^3}{3} \right]_{x_k}^{x_{k+1}}$$

Продолжаем расчет для получения:

$$\int_{x_k}^{x_{k+1}} L_k(t) dt = \frac{(x_{k+1} - x_k)}{2} + \frac{(x_k - x_{k+1})^2}{6(x_k - x_m)}.$$

И, наконец, зная, что $x_{k+1} - x_k = \delta$ и $x_k - x_m = -\frac{x_{k+1} - x_k}{2} = -\frac{\delta}{2}$, мы имеем:

$$\int_{x_k}^{x_{k+1}} L_k(t) dt = \left(\frac{1}{2} - \frac{1}{3} \right) \times \delta = \frac{\delta}{6}.$$

Тот же расчет приводит к

$$\int_{x_k}^{x_{k+1}} L_{k+1}(t) dt = \frac{\delta}{6}.$$

Таким же образом мы вычисляем интеграл от L_m :

$$\int_{x_k}^{x_{k+1}} L_m(t) dt = \int_{x_k}^{x_{k+1}} \frac{(t - x_{k+1})(t - x_k)}{(x_m - x_{k+1})(x_m - x_k)} dt.$$

Всегда оставляя постоянную:

$$\int_{x_k}^{x_{k+1}} L_m(t) dt = \frac{1}{(x_m - x_{k+1})(x_m - x_k)} \int_{x_k}^{x_{k+1}} (t - x_{k+1})(t - x_k) dt.$$

И там мы делаем ту же интеграцию по частям, что и раньше:

$$\int_{x_k}^{x_{k+1}} L_m(t) dt = \frac{1}{(x_m - x_{k+1})(x_m - x_k)} \left(\left[(t - x_k) \frac{(t - x_{k+1})^2}{2} \right]_{x_k}^{x_{k+1}} - \int_{x_k}^{x_{k+1}} \frac{(t - x_{k+1})^2}{2} dt \right).$$

Где:

$$\int_{x_k}^{x_{k+1}} L_m(t) dt = \frac{1}{2(x_m - x_{k+1})(x_m - x_k)} \times \left[\frac{(t - x_{k+1})^3}{3} \right]_{x_k}^{x_{k+1}}.$$

Продолжаем расчет для получения:

$$\int_{x_k}^{x_{k+1}} L_m(t) dt = \frac{1}{2(x_m - x_{k+1})(x_m - x_k)} \times \frac{-(x_k - x_{k+1})^3}{3}.$$

И, наконец, зная, что $x_{k+1} - x_k = \delta$, $x_m - x_k = \frac{x_{k+1} - x_k}{2} = \frac{\delta}{2}$ и $x_m - x_{k+1} = -\frac{x_{k+1} - x_k}{2} = -\frac{\delta}{2}$, мы имеем:

$$\int_{x_k}^{x_{k+1}} L_m(t) dt = -\frac{4}{2\delta^2} \times \frac{-\delta^3}{3} = \frac{2\delta}{3}.$$

Благодаря этим двум вычислениям мы можем вычислить интеграл от Q_k :

$$\begin{aligned} \int_{x_k}^{x_{k+1}} Q_k(t) dt &= \int_{x_k}^{x_{k+1}} f(x_k) L_k(t) dt + f(x_m) L_m(t) dt + f(x_{k+1}) L_{k+1}(t) dt \\ &= f(x_k) \int_{x_k}^{x_{k+1}} L_k(t) dt + f(x_m) \int_{x_k}^{x_{k+1}} L_m(t) dt + f(x_{k+1}) \int_{x_k}^{x_{k+1}} L_{k+1}(t) dt \\ &= \frac{\delta f(x_k)}{6} + \frac{2\delta f(x_m)}{3} + \frac{\delta f(x_{k+1})}{6} \\ &= \frac{\delta}{3} \left(\frac{f(x_k) + f(x_{k+1})}{2} + 2f(x_m) \right). \end{aligned}$$

это была долгосрочная работа, но самое сложное сейчас сделано.

Расчет полного интеграла

Нам остается только вычислить сумму интегралов Q_k , чтобы в итоге получить приближение нашего интеграла (мы вернемся к нашему «i, 5»):

$$I(f) \approx \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} Q_k(t) dt = \sum_{k=0}^{n-1} \frac{\delta}{3} \left(\frac{f(x_k) + f(x_{k+1})}{2} + 2f(x_{k,5}) \right).$$

Как обычно, мы получаем постоянную из суммы:

$$I(f) \approx \frac{\delta}{3} \sum_{k=0}^{n-1} \left(\frac{f(x_k) + f(x_{k+1})}{2} + 2f(x_{k,5}) \right).$$

Обратите внимание, что все x_k учитываются дважды, кроме первого (x_0) и последнего (x_n). Таким образом, мы можем получить их из суммы (но это вынуждает нас получить $f(x_{0,5})$ из суммы):

$$I(f) \approx \frac{\delta}{3} \left(\frac{f(x_0) + f(x_n)}{2} + 2f(x_{0,5}) + \sum_{k=1}^{n-1} (f(x_k) + 2f(x_{k,5})) \right).$$

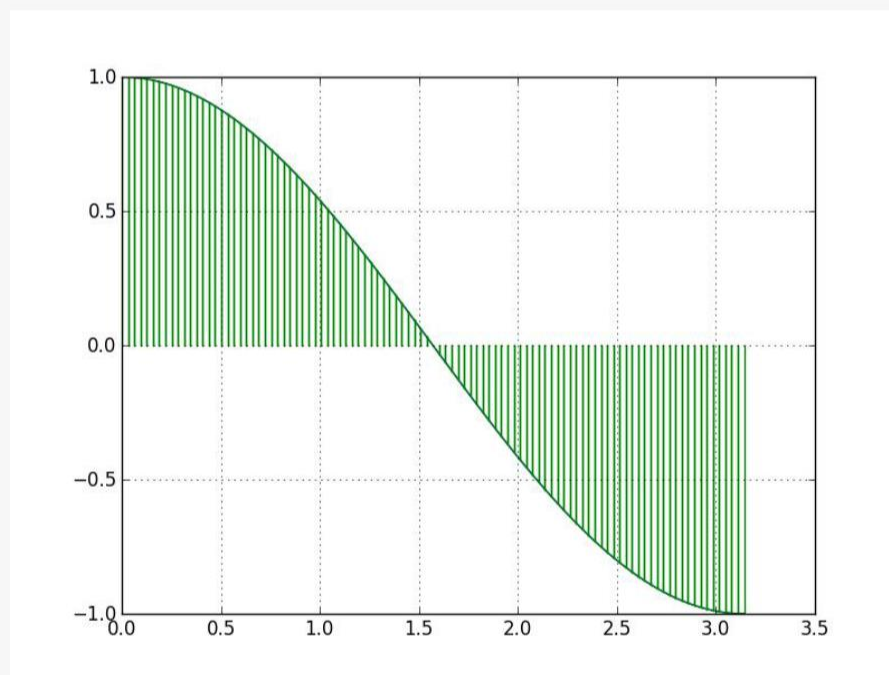
И это письмо, которое мы будем хранить.

Алгоритм и тесты

Давайте напишем этот алгоритм на Python (чтобы получить, 5, нам просто нужно добавить $\delta / 2$)

```
1 def simpson(f, a, b, n):
2     pas = (b - a) / n
3     somme = (f(a) + f(b)) / 2 + 2 * f(a + pas / 2)
4     x = a + pas
5     for i in range(1, n):
6         somme += f(x) + 2 * f(x + pas / 2)
7         x += pas
8     return somme * pas / 3
```

Что касается метода прямоугольников и метода трапеций, давайте попробуем его с функцией косинуса на $[0, \pi]$, разделив интервал на 100. Затем мы получим этот график.



Метод Симпсона о функции косинуса.

График уже дает нам знать, что полученный результат будет близок к истинному значению интеграла, и действительно, мы получаем результат порядка 10^{-15} . Этот результат очень близок к 0.

Фактически, метод Симпсона является менее точным, чем метод трапеции в частном случае функции косинуса. В следующем секретном блоке мы увидим почему. Это только частный случай, но в целом для достаточно регулярной функции (см. Следующую главу) метод Симпсона является более точным.

Приложения

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mpi.h"

#define MASTER 0
#define REPETITIONS 10

double Function(double x);

int main(int argc, char *argv[]){
    int myrank, nprocs, a, b, i;
    long n, j;
    double times[REPETITIONS];
    double time, slowest, startInt, endInt, result, h, x, endresult;

    MPI_Status recv_status;
    MPI_Request request;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

    if (argc != 4){
        printf("Usage: mpiexec -n nodecount simpson1c a b n\n\n");
        return 1;
    }

    a = atoi(argv[1]);
    b = atoi(argv[2]);
    n = atol(argv[3]);

    startInt = (double)(b - a) / nprocs * myrank + a; endInt
    = (double)(b - a) / nprocs * (myrank + 1.0) + a;

    for (i = 0; i < REPETITIONS; ++i){
        MPI_Barrier(MPI_COMM_WORLD);
        time = MPI_Wtime();

        h = ((endInt - startInt) / n);
        x = startInt;
        result = Function(x);
        x = x + h;
        for (j = 1; x < endInt; ++j){
            result = (j % 2 == 0) ? result + 2 * Function(x) : result + 4 * Function(x);
            x = x + h;
        }
    }
```

```

    result += Function(x);
    result = result / (3.0) * h;

    MPI_Reduce(&result, &endresult, 1, MPI_DOUBLE, MPI_SUM, MASTER, MPI_COMM_WORLD);

    time = MPI_Wtime() - time;
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Reduce(&time, &slowest, 1, MPI_DOUBLE, MPI_MAX, MASTER, MPI_COMM_WORLD);

    if (myrank == MASTER) {
        times[i] = slowest;
    }
}

if (myrank == MASTER){
    double min = times[0];
    for (int i = 1; i < REPETITIONS; i++) {
        min = (times[i] < min) ? times[i] : min;
    }
    //printf("\n%d; %ld; %f; %f\n", nprocs, n, min, endresult);
    printf("\nnprocs= %d; n= %ld; min= %f; endresult= %f\n\n", nprocs, n, min, endresult);
}

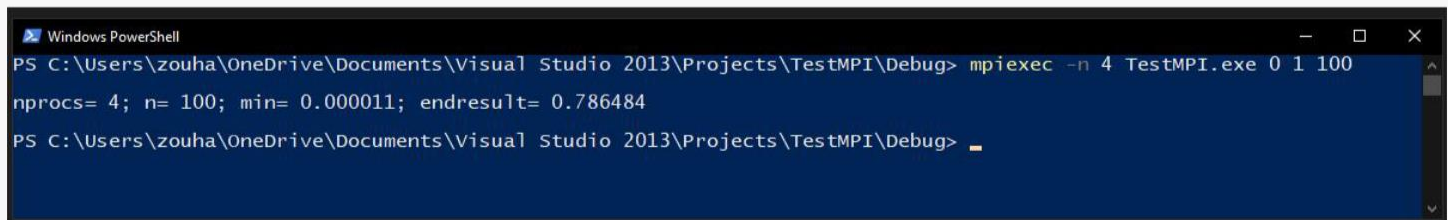
MPI_Finalize();
}

double Function(double x){
    double result = sqrt(1.0 - x*x);
    if (isnan(result)){ // sometimes encountered a problem with x=1.0 which resulted in minus
        result = 0.0f;
    }
    return result;
}

```

Теперь пришло время посмотреть, что этот метод дает нам на нашем примере.

mpiexec -n 4 Simpson.exe 0 1 100



```

Windows PowerShell
PS C:\Users\zouha\OneDrive\Documents\Visual Studio 2013\Projects\TestMPI\Debug> mpiexec -n 4 TestMPI.exe 0 1 100
nprocs= 4; n= 100; min= 0.000011; endresult= 0.786484
PS C:\Users\zouha\OneDrive\Documents\Visual Studio 2013\Projects\TestMPI\Debug>

```

С этим мы получаем около **0,746824**.