Федеральное агентство по образованию РФ
Новгородский государственный университет им. Ярослава Мудрого

Кафедра ИТИС

Численное интегрирование методом трапеций и методом Симпсона

Выполнил студент группы 9311
Лопатин А.С.

Великий Новгород, 2009

1. Цель работы

Реализовать алгоритмы численного интегрирования методом трапеций и методом Симпсона на языке программирования C++.

2. Математическая модель решения

Даны границы интегрирования a и b, точность ε и количество узлов интерполяции n. Требуется найти определенный интеграл $\int\limits_a^b f(x)dx$ (где $f(x)$, например, квадратичная функция $y = 10x - 2x^2$) численными методами: методом трапеций и методом Симпсона.

Метод трапеций заключается в замене площади фигуры площадями прямоугольных трапеций, которые можно вписать в такую фигуру. Возьмем за длину шага между трапециями $h = \frac{b-a}{n}$, которая будет также являться ее высотой. Тогда прямая, проходящая через координаты $(x_i, 0)$ и $(x_i, f(x_i))$ будет одним из оснований этой трапеции. Второе же основание будет таким же, но с x_{i+1}. Отсюда можно найти площадь трапеции с помощью полусуммы оснований, умноженной на высоту: $S_i = \frac{f(x_i)+f(x_{i+1})}{2} \cdot h$. Остается только просуммировать эти площади:

$$S = \sum_{i=0}^{n} \frac{f(x_i) + f(x_{i+1})}{2} \cdot h$$

Проверять точность можно просто сравнивая модуль разности предыдущего результата и текущего с ε.

Метод Симпсона же заключается в замене подынтегральной функции параболой, которая проходит через три точки отрезка интегрирования (в качестве таких точек используют концы отрезка и его среднюю точку). Формула в таком случае выглядит так:

$$S = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Если разбить интервал интегрирования на $2n$ равных частей, то получится:

$$S = \frac{h}{6}(f_0 + 2(f_2 + f_4 + \ldots + f_{2n-2}) + 4(f_1 + f_3 + \ldots + f_{2n-1}) + f_{2n})$$

где $f_i = a + \frac{h \cdot i}{2}$. Проверять точность на соответствие ε можно по следующей формуле:

$$|S_{j-1} - S_j| > \frac{\varepsilon \cdot h \cdot n}{b - a}$$

где S_j и S_{j-1} площади текущей и предыдущей фигуры соответственно.

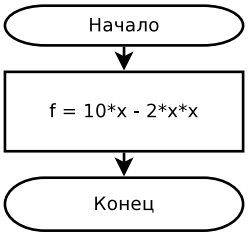
3. Спецификация

Таблица переменных:

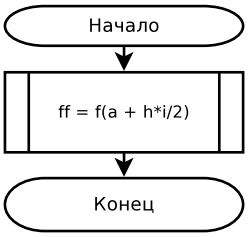
Исходное значение	Идентификатор	Тип	Вид	Размерность	Назначение
a	a	Вещественный	Простой	–	Граница интегрирования
b	b	Вещественный	Простой	–	Граница интегрирования
ε	eps	Вещественный	Простой	–	Точность
n	n	Целый	Простой	–	Количество узлов
h	h	Вещественный	Простой	–	Шаг между узлами
s	s	Вещественный	Простой	–	Сумма
s_old	s_old	Вещественный	Простой	–	Предыдущая сумма
s_1	s1	Вещественный	Простой	–	Первая сумма для метода Симпсона
s_2	s2	Вещественный	Простой	–	Вторая сумма для метода Симпсона
x	x	Вещественный	Простой	–	Аргумент функции $f(x)$
i	i	Целый	Простой	–	Счетчик итераций
done	done	Логический	Простой	–	Признак соответствия точности

4. Алгоритм

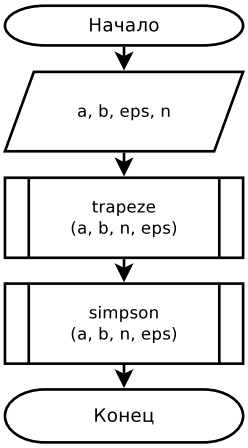
4.1. Функция «$f(x)$»



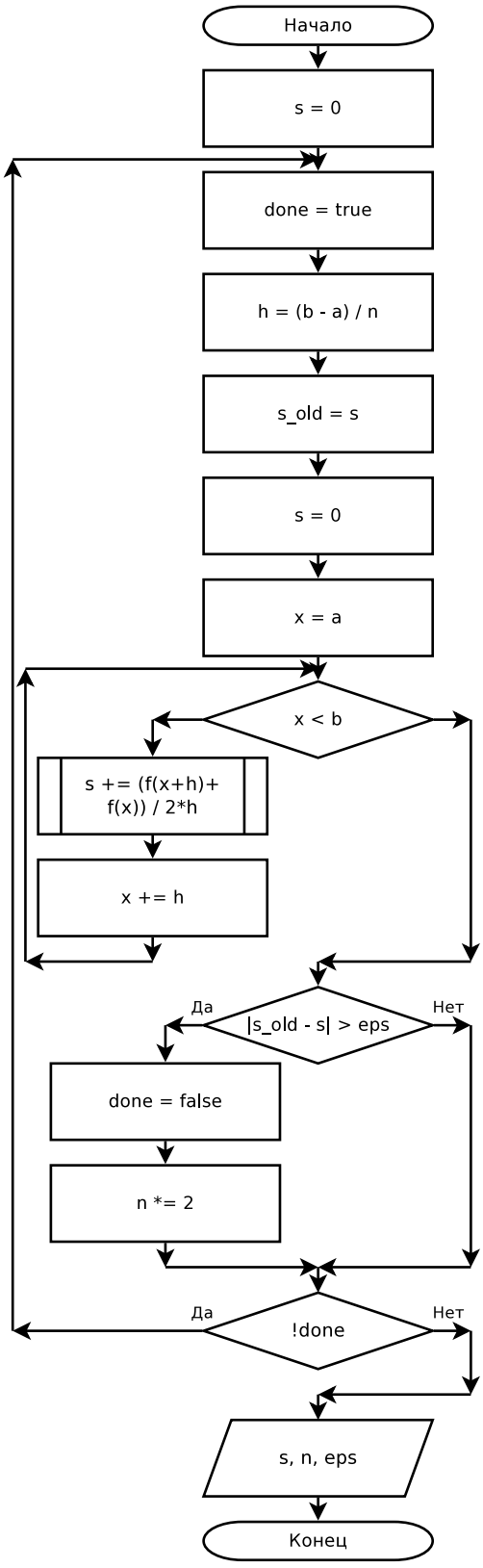
4.2. Функция «f_i»



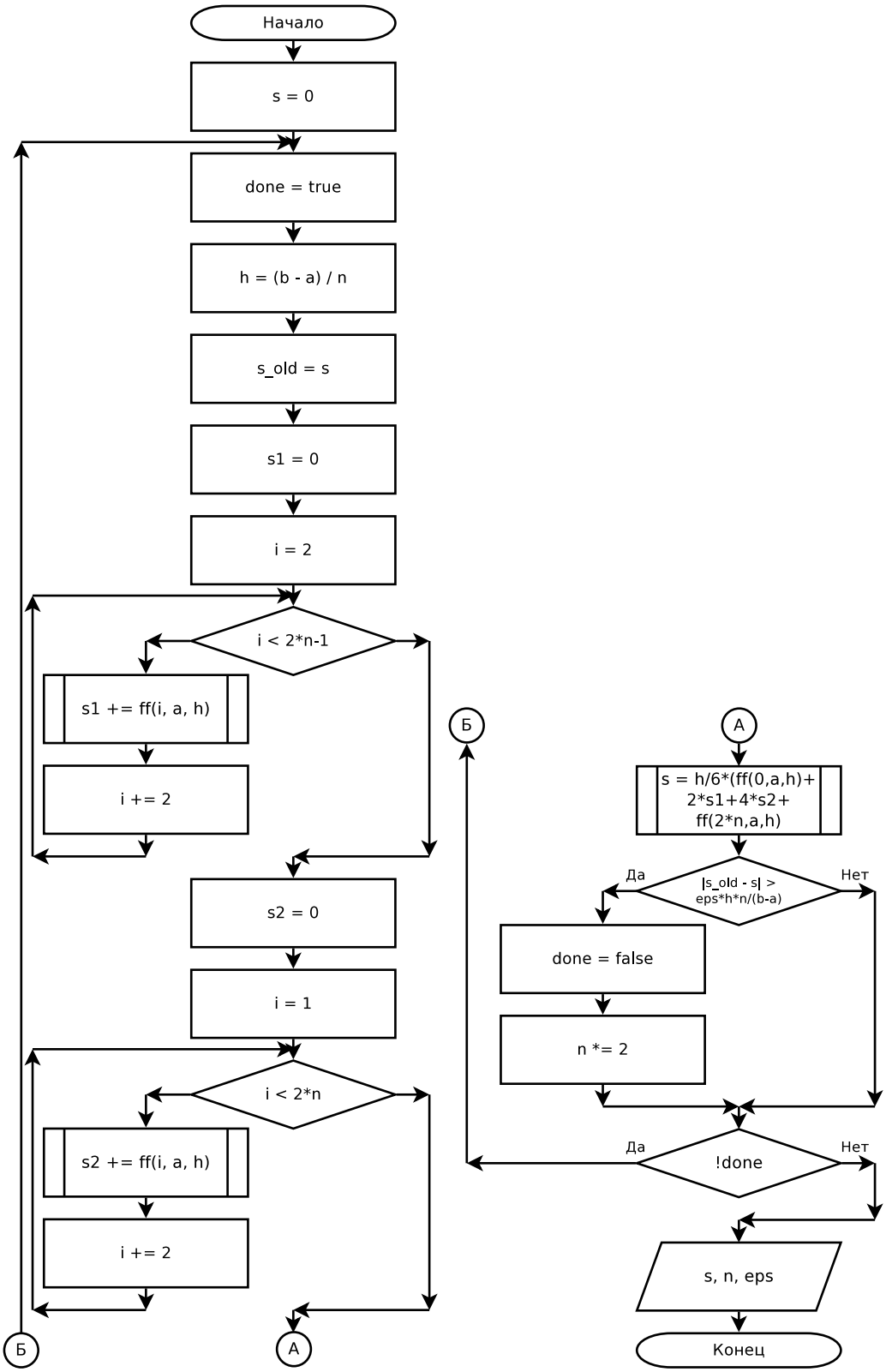
4.3. Функция «main»



4.4. Процедура «trapeze»



4.5. Процедура «simpson»



5. Программа

```
#include <iostream>
#include <cmath>

using namespace std;

void input(double & a, double & b, double & eps, int & n)
{
    cout << "Input a = "; cin >> a;
    cout << "Input b = "; cin >> b;
    cout << "Input eps = "; cin >> eps;
    cout << "Input n = "; cin >> n;
}

void print(const char *method, const double & s, int n, const double & eps)
{
    cout << method
         << "s = " << s
         << ", n = " << n
         << ", eps = " << eps
         << endl;
}

double f(const double & x)
{
    return 10*x - 2*x*x;
}

void trapeze(const double & a, const double & b, int n, const double & eps)
{
    double h, x, s = 0, s_old;
    bool done;

    do {
        done = true;
        h = (b - a) / n;
        s_old = s;
```

```
        s = 0;
        for (x = a; x < b; x += h)
            s += (f(x+h) + f(x)) / 2*h;

        if (abs(s_old - s) > eps) {
            done = false;
            n *= 2;
        }
    } while (!done);

    print("Trapeze method: ", s, n, eps);
}

double ff(int i, const double & a, const double & h)
{
    return f(a + h*i/2);
}

void simpson(const double & a, const double & b, int n, const double & eps)
{
    double h, s = 0, s_old, s1, s2;
    int i; bool done;

    do {
        h = (b - a) / n;
        done = true;
        s_old = s;

        s1 = 0;
        for (i = 2; i < 2*n-1; i += 2)
            s1 += ff(i, a, h);

        s2 = 0;
        for (i = 1; i < 2*n; i += 2)
            s2 += ff(i, a, h);

        s = h/6 * (ff(0, a, h) + 2*s1 + 4*s2 + ff(2*n, a, h));

        if (abs(s_old - s) > eps*h*n / (b - a)) {
```

```
            done = false;
            n *= 2;
        }
    } while (!done);

    print("Simpson's method: ", s, n, eps);
}

int main()
{
    double a, b, eps; int n;
    input(a, b, eps, n);
    trapeze(a, b, n, eps);
    simpson(a, b, n, eps);
    return 0;
}
```

6. Шаблон ввода исходных данных

```
1
10
0.034
34
```

7. Шаблон вывода результата

```
Trapeze method: s = -171.003, n = 272, eps = 0.034
Simpson's method: s = -171, n = 68, eps = 0.034
```

8. Вывод

Исходя из проделанной работы можно сделать вывод, что метод Симпсона превосходит по своей эффективности метод трапеций, так как требует меньше узлов интерполяции и в то же время имеет большую точность.