**Отчет о научно-исследовательской работе**

1. ***Условие задачи***

Написать программу для решения нелинейного уравнения

,

где подынтегральную функцию *f*(*x*), параметры *a*, *b*, допустимую погрешность решения *ε* и начальное приближение *x*0 задает пользователь (можно в коде программы).

Способ численного решения нелинейного уравнения

(1)

и связанного с ним вычисления определенного интеграла

(2)

определяется по варианту.

Вариант 1. Решить нелинейное уравнение (1) методом хорд, при этом осуществить интегрирование (2) с помощью формулы прямоугольников 1-го порядка.

1. ***Описание алгоритма решения и структур хранения данных***

Для решения нелинейного уравнения используется метод хорд, а для интегрирования – метод прямоугольников 1-го порядка.

Метод прямоугольников – метод численного интегрирования функции одной переменной, заключающийся в замене подынтегральной функции на многочлен нулевой степени, то есть константу, на каждом элементарном отрезке [a, b].

При вычислении интеграла методом прямоугольников 1-го порядка, также известного как метод левых прямоугольников, криволинейная трапеция заменяется прямоугольниками, высоты которых равны значению функции в левых точках интервалов.

Формула вычисления интеграла методом левых прямоугольников:

(3)

Метод хорд (секущих) — один из численных методов решения нелинейных уравнений и основан на последовательном сужении интервала, содержащего единственный корень уравнения. Итерационный процесс выполняется до того момента, пока не будет достигнута заданная точность ε.

Основное отличие от других численных методов заключается в том, что для нахождения последующего приближения используются два предыдущих. Порядок сходимости метода секущих ниже, чем у метода касательных и равен в случае однократного корня золотому сечению.

Итерационный процесс имеет вид:

(4)

Из (1) следует:

находится квадратурными методами интегрирования.

Из (3) следует:

Вычисление проводится через квадратурную формулу 1-го порядка.Тогда итоговая формула решения нелинейного уравнения будет иметьследующий вид:

В программе используются следующие переменные и типы:

Для хранения данных: функция типа double f(x), double a, b, double eps, double n0, double x0.

Переменные-константы double π, θ.

Переменные для метода хорд: double x1, x2, double In, In1.

Переменные для проверки правила Рунге: double In2, diff, n.

Переменные для метода квадратов 1-го порядка: double h, sum, xi.

1. ***Описание входных и выходных данных***

На вход поступают функция f(x), параметры a, b, допустимая погрешность Ɛ, начальное приближение x0, начальное число отрезков n0, которые задаются в коде программы через переменные-константы для возможности дальнейшего модифицирования кода.

На выходе получаем вычисленное значение x, количество итераций и время работы программы.

1. ***Текст программы***

#include <iostream>

#include <iomanip>

#include <math.h>

#include <omp.h>

#include <corecrt\_math\_defines.h>

using namespace std;

#define THREADS\_COUNTS omp\_get\_num\_threads()

#define CPU\_COUNTS omp\_get\_num\_procs()

#define CPU\_ID omp\_get\_thread\_num()

using namespace std;

enum verbose { off, medium, full };

verbose DetailedInf =off;

const double eps = 1e-5;

double n0 = 100;

const double π = M\_PI;

const double θ = 1.0 / 3.0;

const double a = 0;

const double b = 5;

const double X0 = 0.1;

double f(double x) {

return 2 \* x + 1;

}

//const double a = 0;

//const double b = 2;

//const double X0 = 3;

//double f(double x) {

// return sin(x);

//}

//const double a = -1;

//const double b = 10;

//const double X0 = 0.7;

//double f(double x) {

// return pow(x, 2) + 3 \* x - 2;

//}

//const double a = 0;

//const double b = PI;

//const double X0 = 0.5;

//double f(double x) {

// return 4 / (x \* x + 1);

//}

double In; double In1; double I2n;

double RectangleMethod1(double b, int n) {

double sum = 0; double h = (b - a) / n;

int i;

#pragma omp parallel for reduction(+:sum)

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

if (DetailedInf == full) {

cout << "Потоки: " << THREADS\_COUNTS << endl;

cout << CPU\_ID << " - " << sum << endl;

}

sum += f(a + i \* h);

}

sum \*= h;

if (DetailedInf) cout << "b = " << fixed << b << "Sum = " << sum << " N = " << n << endl;

return sum;

}

double RungeRule(double b) {

double In = RectangleMethod1(b, n0); double In2 = RectangleMethod1(b, 2 \* n0);

double diff = In2 - In;

while (θ \* abs(diff) >= eps) {

n0 \*= 2;

In = RectangleMethod1(b, n0); In2 = RectangleMethod1(b, 2 \* n0);

diff = In2 - In;

}

if (DetailedInf) cout << "N1 = " << n0 << " Diff = " << diff << endl;

return n0;

}

double ChordMethod() {

double start = omp\_get\_wtime();

double x0 = X0;

double x1 = b;

double N0 = RungeRule(x0);

In = RectangleMethod1(x0, N0);

In1 = RectangleMethod1(x1, N0);

double x2 = x1 - ((In1 - b) \* (x1 - x0)) / ((In1 - b) - (In - b));

int iter = 1;

while (abs(x2 - x1) >= eps) {

iter++;

x0 = x1;

x1 = x2;

In = In1;

In1 = RectangleMethod1(x1, N0);

x2 = x1 - ((In1 - b) \* (x1 - x0)) / ((In1 - b) - (In - b));

}

double end = omp\_get\_wtime();

cout << "Результат: x1 = " << fixed << x0 << endl;

cout << "Количество итераций: " << iter << endl;

return end - start;

}

void InputData() {

cout.precision(10);

cout << "Входные данные:" << endl;

cout << "Нижний предел интегрирования a = " << a << endl;

cout << "Верхний предел интегрирования b = " << b << endl;

cout << "Допустимая погрешность решения eps = " << eps << endl;

cout << "Количество отрезков n0 = " << n0 << endl;

cout << "Начальное приближение x0 = " << X0 << endl;

cout << "Общее количество процессоров: " << CPU\_COUNTS << endl;

}

void SetCPUcounts(int N) {

omp\_set\_num\_threads(N);

cout << "Количество потоков: " << N << endl;

}

void Result() {

cout << endl;

cout.precision(10);

cout << "Время вычисления: " << ChordMethod() << " секунд" << endl;

}

int main() {

setlocale(0, "");

InputData();

SetCPUcounts(8); // Количество потоков

Result();

return 0;

}

1. ***Тестовый запуск программы и вывод результатов работы***

Пример 1

Заданные параметры:

Подынтегральная функция:

Нелинейное уравнение:

Вычисленный определенный интеграл:

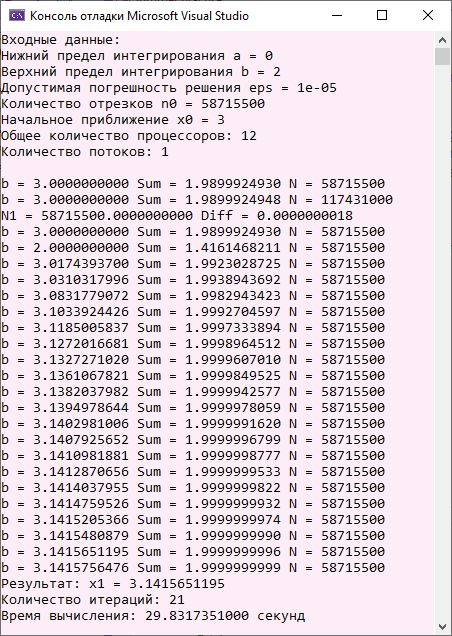
**

Рисунок 1 – пример при использовании одного потока, число отрезков вычислено по правилу Рунге

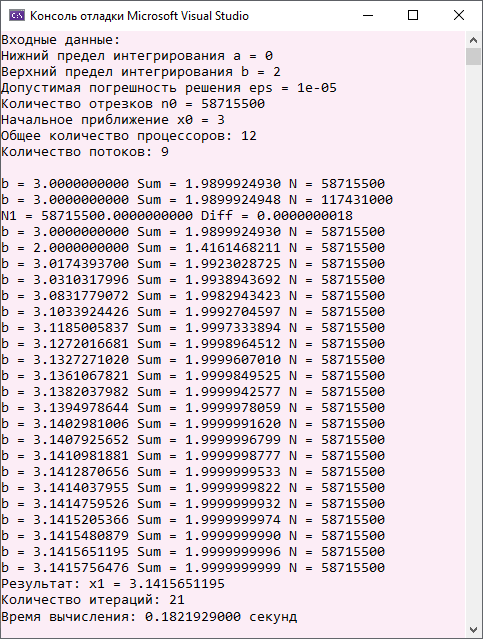


Рисунок 2 – пример при использовании девяти потоков, число отрезков вычислено по правилу Рунге

Пример 2

Заданные параметры:

Подынтегральная функция:

Нелинейное уравнение:

Вычисленный определенный интеграл:

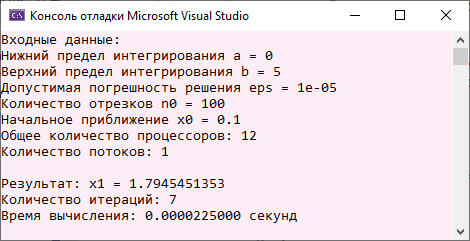


Рисунок 3 – пример при использовании одного потока

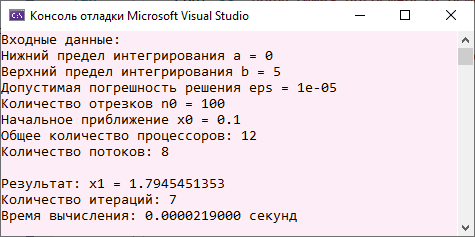


Рисунок 4 – пример при использовании восьми потоков

Пример 3

Заданные параметры:

Подынтегральная функция:

Нелинейное уравнение:

Вычисленный определенный интеграл:

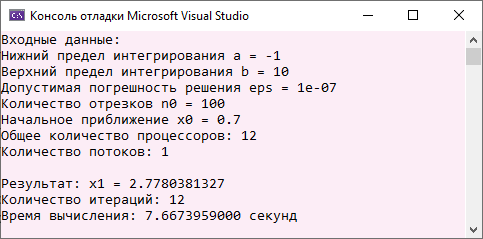


Рисунок 5 – пример при использовании одного потока

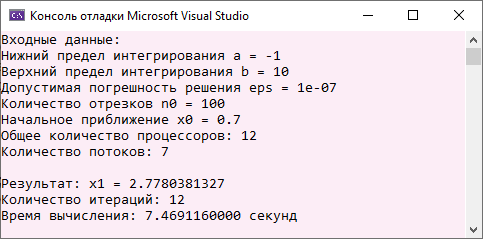


Рисунок 6 – пример при использовании семи потоков

Пример 4

Заданные параметры:

Подынтегральная функция:

Нелинейное уравнение:

Вычисленный определенный интеграл:

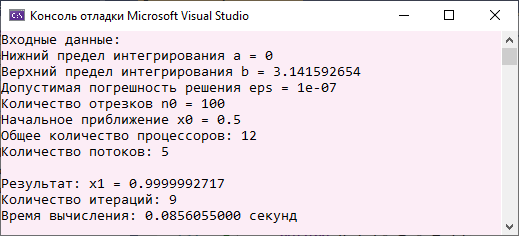


Рисунок 7 – пример при использовании пяти потоков

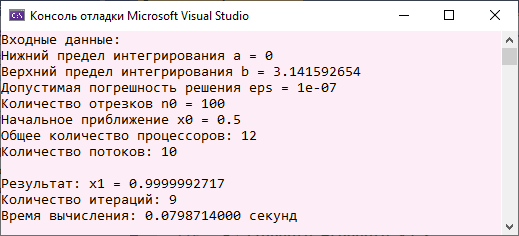


Рисунок 7 – пример при использовании десяти потоков

1. ***Эффективность распараллеливания кода***

Характеристики компьютера:

Процессор: Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz 2.59 GHz

Оперативная память: 32,0 ГБ

Для оценки эффективности распараллеливания взят пример 1, с погрешностью .

Таблица 1 — Зависимость времени работы программы от числа параллельных потоков

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Число потоков i | 1 | 3 | 5 | 7 | 9 | 11 |
| Время работы  T(i), с. | 27,95 | 19,61 | 12,97 | 7,11 | 1,18 | 1,35 |

Таблица 2 — Зависимость ускорения работы программы от числа параллельных потоков.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Число потоков i | 1 | 3 | 5 | 7 | 9 | 11 |
| Ускорение работы S(i) | 1 | 1,43 | 2,15 | 3,93 | 23,69 | 20,7 |

Оптимальное число потоков 9, потому что время работы при большем количестве потоков увеличивается.

Ускорением параллельного алгоритма называют отношение времени выполнения лучшего последовательного алгоритмам к времени выполнения параллельного алгоритма: