Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Кубанский государственный университет»

Факультет компьютерных технологий и прикладной математики

Кафедра прикладной математики

**ОТЧЕТ О ПРОХОЖДЕНИИ УЧЕБНОЙ ПРАКТИКИ**

**НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА   
(получение первичных навыков научно-исследовательской работы)**

период с 06.07.2025 г. по 19.07.2025 г.

Ахвердян Гурген Артурович

*(Ф.И.О. студента)*

студент(ка) 29 группы 2 курса ОФО

Направление подготовки   
02.03.02 Фундаментальная информатика и информационные технологии

Руководитель учебной практики

доцент кафедры прикладной математики

факультета компьютерных технологий

и прикладной математики, к.ф.-м.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Письменский А.В.

Оценка по итогам защиты практики: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Краснодар 2025 г.

**СОДЕРЖАНИЕ**

1. Постановка задачи…………………………………………………………...... 3

2. Теория………………………………………………………………………... 3-4

2.1. Метод Ньютона (канонический)…………………………………... 3-4

2.2. Метод прямоугольников 1-го порядка……………………………… 4

3. Алгоритм решения задачи……………………………………………………. 5

4. Программная реализация…………………………………………………... 5-6

5. Набор данных для тестирования…………………………………………... 6-7

6. Оценка эффективности параллельных вычислений……………………… 7-8

Заключение…………………………………………………………………….. 8-9

Список использованных источников…………………………………………... 9

Приложение А – Листинг программы…………………………………….. 10-11

**1. Постановка задачи**

Написать программу для решения нелинейного уравнения

методом Ньютона (каноническим) (где подынтегральную функцию f(x), параметры a, b, допустимую погрешность решения ε и начальное приближение x0 задает пользователь (можно в коде программы)) реализовав вычисление связанного с ним определенного интеграла

с помощью формулы прямоугольников 1-го порядка.

**2. Теория**

**2.1 Метод Ньютона (канонический)**

Метод Ньютона – это итерационный численный метод нахождения корня заданной функции, сводящийся к следующей итерационной процедуре вычисления.

Итерационная формула: , где – текущее приближение, – производная функция f(x) в точке

Условия остановки:

1. – достижение заданной точности

. – значение функции близко к нулю

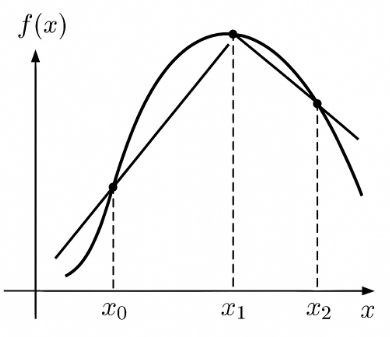


Рисунок 1 – Метод Ньютона

**2.2 Метод прямоугольников 1-го порядка**

Метод прямоугольников 1-го порядка – это простейший способ численного интегрирования, основанный на аппроксимации площади под кривой f(x) прямоугольников.

Интеграл функции f(x) на отрезке вычисляется как:

где – шаг разбиения, – точки разбиения, n – количество отрезков

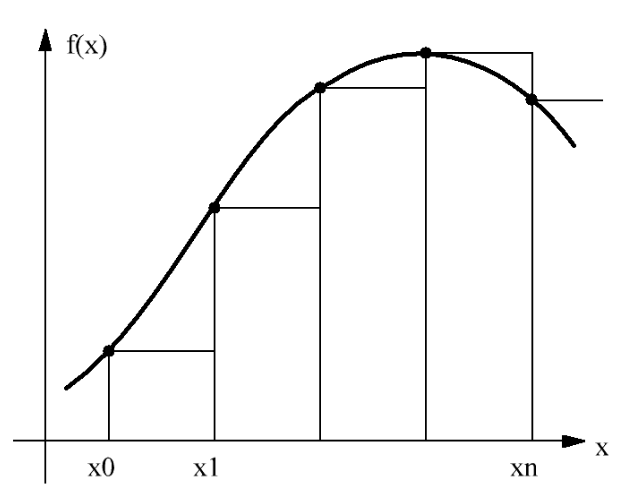


Рисунок 2 – Метод прямоугольников 1-го порядка

**3. Алгоритм решения задачи**

**Цель:** Найти корень уравнения  методом Ньютона с численным интегрированием по формуле прямоугольников 1-го порядка.

**Шаг №1:**

Задать: подынтегральную функцию f(x) (например sin(x)); значения a и b; начальное приближение ; точность (например ); количество разбиений n для метода прямоугольников.

**Шаг №2. Реализация метода Ньютона**

1) Вычислить функцию

2) Вычислить итерационный процесс: пока || > :

вернуть

**Шаг №3. Реализация метода прямоугольников 1-го порядка**

1) Разбить отрезок [a,x] на n частей с шагом

2) Вычислить сумму

3) Вернуть значение sum как приближенное значение

**4. Программная реализация**

Задача была написана на языке C++ с использованием технологии OpenMP для параллельных вычислений. В программе также используются стандартные библиотеки для математических операций (cmath), управления выводом (iomanip) и работы с векторами (vector).

**Описание программы:**

**function(double x)**

Возвращает значение функции f(x) которая является подынтегральной функцией в задаче.

**integrirovanie(double a, double b, int n)**

Вычисляет определенный интеграл функции f(x) на интервале [a,b] методом прямоугольников 1-го порядка. Параллелизация выполнена с помощью функции OpenMP  #pragma omp parallel for reduction(+:sum), которая распределяет итерации цикла между потоками и суммирует результаты.

**g(double x, double a, double b, int n)**

Вычисляет значение функции , которая используется в методе Ньютона.

**dg(double x)**

Возвращает производную функции g(x), которая равна f(x).

**newton\_metod(double a, double b, double x0, double eps, int max\_iter, int n\_integral)**

Реализует метод Ньютона для нахождения корня уравнения g(x)=0. На каждой итерации вычисляется значение g(x) и его производной, после чего обновляется приближение корня. Если производная близка к нулю или превышено максимальное число итераций, метод возвращает NAN.

**main()**

Здесь задаются параметры задачи, такие как: пределы интегрирования, начальное приближение, точность, число итераций, число разбиений.

Проводится тестирование программы с разным количеством потоков. Для каждого случая замеряется время выполнения, вычисляется ускорение и эффективность.

Реализован вывод в консоль.

**5. Набор данных для тестирования**

**Тестовые данные для программы**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Подынтегральная функция | Значение a | Значение b | Начальное приближение | Точность | Число итераций | Количество разбиений | Результат |
| sin(x) | 0 | 1 | 1,5 |  | 100 | 1000 | 1,5708 |
| cos(x) | 0 | 0,5 | 1 |  | 100 | 10 | 0.5235 |
| exp(x) | 0 | 1 | 1 |  | 100 | 1000 | 0,693147 |
| x\*x | 0 | 1 | 1 |  | 100 | 1000 | 1,44225 |
|  | 1 | 7 | 3 |  | 100 | 10 | 2,8026 |

Таблица 1 – Набор данных для тестирования

Пример вывода результата работы программы приведен на рисунке 1.

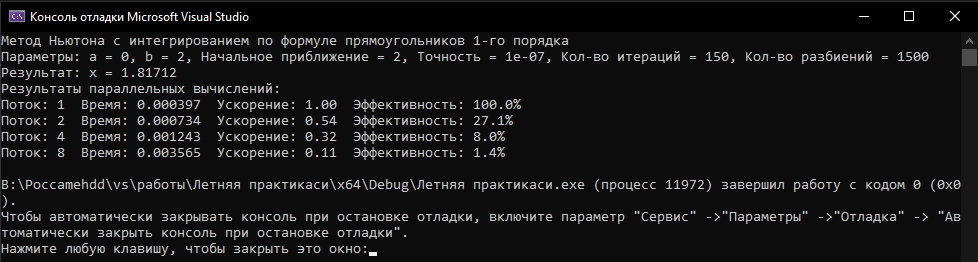


Рисунок 3 – Результат работы программы

**6. Оценка эффективности параллельных вычислений**

Для того, чтобы дать оценку эффективности параллельных вычислений, нам необходимо знать такой параметр как ускорение.

Для вычисления ускорения мы воспользовались формулой , где – это время выполнения на потоке 1, а – это время выполнения на n-потоках.

Для вычисления эффективности мы воспользовались формулой ,где S(n) – это ускорение для n-потоков, n – это количество потоков.

В качестве примера, воспользуемся «Таблица – 1» и возьмем подынтегральную функцию «sin(x)» с заданными для нее параметрами, затем подставим функцию в программу и запустим ее. По завершению программы, мы можем увидеть эффективность параллельных вычислений, как показано на рисунке 2.

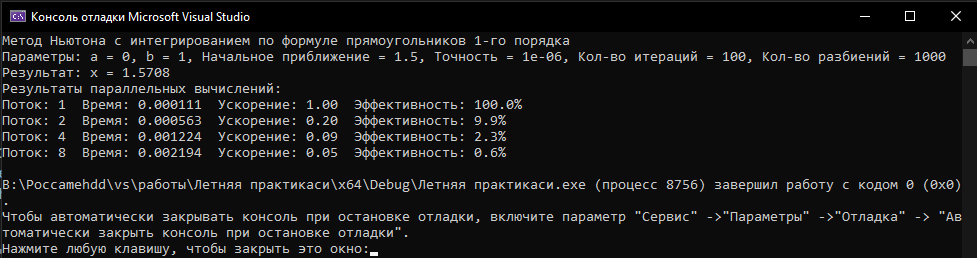


Рисунок 4 – Эффективность параллельных вычислений

В результате тестирования, минимальное время работы, максимальное ускорение и максимальная эффективность достигнута при запуске процесса из 1-го потока.

**ЗАКЛЮЧЕНИЕ**

За время прохождения практики приобретен опыт в реализации программных продуктов, основанных на знаниях. Реализована программа для решения нелинейного уравнения. Способ численного решения нелинейного уравнения и связанного с ним вычисления определенного интеграла. Также была написана параллельная OpenMP программа и показана ее эффективность.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Амосов А.А., Дубинский Ю.А., Копченова Н.В. "Вычислительные методы". — СПб.: Лань, 2014. — С. 150–155.

2. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. "Численные методы". — М.: Лаборатория знаний, 2015. — С. 200–205.

3. Лапчик М.П., Рагулина М.И., Хеннер Е.К. "Численные методы". — М.: Академия, 2004. — С. 180–185.

4. Амосов А.А., Дубинский Ю.А., Копченова Н.В. "Вычислительные методы" — СПб.: Лань, 2014. — С. 282–283.

5. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. "Численные методы" — М.: Лаборатория знаний, 2015. — С. 320–325.

6. Лапчик М.П., Рагулина М.И., Хеннер Е.К. "Численные методы" — М.: Академия, 2004. — С. 190–195.

**Приложение А**

Листинг программы

#include <iostream>

#include <cmath>

#include <omp.h>

#include <vector>

#include <iomanip>

using namespace std;

double function(double x) {

return sin(x);

}

double integrirovanie(double a, double b, int n) {

double h = (b - a) / n;

double sum = 0.0;

#pragma omp parallel for reduction(+:sum)

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

double x = a + (i + 0.5) \* h;

sum += function(x);

}

return sum \* h;

}

double g(double x, double a, double b, int n) {

return integrirovanie(a, x, n) - b;

}

double dg(double x) {

return function(x);

}

double newton\_metod(double a, double b, double x0, double eps, int max\_iter, int n\_integral) {

double x = x0;

double x\_prev;

int iter = 0;

do {

x\_prev = x;

double g\_val = g(x\_prev, a, b, n\_integral);

double dg\_val = dg(x\_prev);

if (fabs(dg\_val) < 1e-10) {

cout << "Производная близка к нулю. Метод Ньютона не сходится." << endl;

return NAN;

}

x = x\_prev - g\_val / dg\_val;

iter++;

if (iter > max\_iter) {

cout << "Превышено максимальное число итераций." << endl;

return NAN;

}

} while (fabs(x - x\_prev) > eps);

return x;

}

int main() {

setlocale(0, "");

double a = 0.0;

double b = 1.0;

double x0 = 1.5;

double eps = 1e-6;

int iter = 100;

int n = 1000;

vector<int> threads = { 1, 2, 4, 8 };

vector<double> times(threads.size());

vector<double> results(threads.size());

cout << "Метод Ньютона с интегрированием по формуле прямоугольников 1-го порядка" << endl;

cout << "Параметры: a = " << a << ", b = " << b << ", Начальное приближение = " << x0 << ", Точность = " << eps << ", Кол-во итераций = " << iter << ", Кол-во разбиений = " << n << endl;

for (size\_t i = 0; i < threads.size(); i++) {

int num\_threads = threads[i];

double start\_time = omp\_get\_wtime();

omp\_set\_num\_threads(num\_threads);

results[i] = newton\_metod(a, b, x0, eps, iter, n);

double end\_time = omp\_get\_wtime();

times[i] = end\_time - start\_time;

}

cout << "Результат: x = " << results[0] << endl;

cout << "Результаты параллельных вычислений:" << endl;

for (size\_t i = 0; i < threads.size(); i++) {

double speedup = times[0] / times[i];

double efficiency = speedup / threads[i] \* 100;

cout << "Поток: " << threads[i] << " Время: " << fixed << setprecision(6) << times[i] << " Ускорение: " << fixed << setprecision(2) << speedup << " Эффективность: " << fixed << setprecision(1) << efficiency << "%" << endl;

}

return 0;

}