# Лабораторная работа № 4 Оценка сложности и отладка

# математических алгоритмов

**Цель лабораторной работы**

Закрепление теоретических знаний реализации и отладки вычислительных алгоритмов на примере расчета определенных интегралов.

**Постановка задачи**

**Сложность алгоритма** — это функция, описывающая зависимость времени выполнения или объема используемой памяти от размера входных данных. Оценивается в терминах асимптотического поведения при стремлении n → ∞.

**Виды сложности**

1. **Временная сложность (Time Complexity)**
   * Определяет количество **элементарных операций**, выполняемых алгоритмом.
   * Измеряется в **"О-большом"** (асимптотическая верхняя граница).
2. **Пространственная сложность (Space Complexity)**
   * Определяет объем **памяти**, используемой алгоритмом.
   * Учитывает дополнительную память (кроме входных данных).

**Асимптотические обозначения**

Для формализации оценки сложности используются:

* **O(f(n))** — верхняя граница ("не хуже, чем").
* **Ω(f(n))** — нижняя граница ("не лучше, чем").
* **Θ(f(n))** — точная оценка (совпадение O и Ω).

| **Сложность** | **Описание** | **Пример алгоритма** |
| --- | --- | --- |
| **O(1)** | Константное время | Доступ к элементу массива |
| **O(log n)** | Логарифмическое время | Бинарный поиск |
| **O(n)** | Линейное время | Поиск в неотсорт. массиве |
| **O(n log n)** | Линейно-логарифмическое время | Быстрая сортировка |
| **O(n²)** | Квадратичное время | Пузырьковая сортировка |
| **O(2ⁿ)** | Экспоненциальное время | Перебор всех подмножеств |

**Отладка** — это процесс выявления, локализации и устранения ошибок (дефектов) в программном коде с использованием специализированных инструментов и методик.

**Ключевые аспекты:**

* **Диагностика** (анализ поведения программы);
* **Верификация** (проверка соответствия ожидаемому результату);
* **Коррекция** (исправление ошибок).

**Инструменты:**

* **Отладчики** (Debugger, например, в Visual Studio);
* **Профилировщики** (для анализа производительности);
* **Логирование** (запись состояния программы).

**Точка останова** — это маркер в коде, при достижении которого выполнение программы приостанавливается, позволяя разработчику исследовать:

* **значения переменных**,
* **стек вызовов**,
* **состояние памяти**.

**Виды точек останова:**

* **Условные** (срабатывают при выполнении условия);
* **По количеству попаданий** (активируются после N-ного вызова);
* **На данные** (при изменении значения переменной).

**Стек вызовов** — это структура данных, отражающая последовательность вызовов методов, приведших к текущей точке выполнения программы.

**Использование при отладке:**

* Позволяет отследить **путь выполнения**;
* Выявляет **рекурсивные вызовы**;
* Помогает находить **источники исключений**.

**Пошаговое выполнение** — это режим отладки, при котором программа выполняется по одной инструкции за раз.

**Основные команды:**

* **Step Into (F11)** — вход в вызываемый метод;
* **Step Over (F10)** — выполнение текущего метода без входа в вызываемые;
* **Step Out (Shift+F11)** — выход из текущего метода.

**Окно просмотра переменных** — инструмент отладчика, отображающий текущие значения переменных и выражений в режиме реального времени.

**Функциональность:**

* **Изменение значений** переменных во время отладки;
* **Вычисление выражений**;
* **Отслеживание сложных структур данных**.

**Реализация математических алгоритмов**

Для некоторых подынтегральных функций интеграл можно вычислить аналитически или найти в справочниках. Однако в общем случае первообразная функции:

* может быть *неопределенной*;
* может не иметь выражения через элементарные функции.

Кроме того, сами подынтегральные функции в отдельных случаях не являются элементарными. В конечном счёте это приводит к необходимости разработки приближенных методов вычисления определенных интегралов. Наиболее простыми среди них являются так называемые *классические методы численного интегрирования*:

* прямоугольников;
* трапеций;
* парабол.

Каждый из этих методов основан на суммировании элементарных площадей, на которые разбивается вся площадь под функцией. Так в методе прямоугольников площадь под графиком функции (а значит, и определенный интеграл от *a* до *b*) может быть определен по одной из формул:

а) для входящих прямоугольников



б) для выходящих прямоугольников



где *n* – кратность (количество шагов) интегрирования функции  в точке, определяющей либо вписанный, либо описанный по отношению к графику интегрируемой функции прямоугольник.

В методе трапеций площадь криволинейной трапеции и интеграл могут быть вычислены по формуле

.

В методе парабол (Симпсона) определение площади под графиком интегрируемой функции основано на замене двух смежных фрагментов участком параболы второго порядка (параболической трапецией)



**Задание на лабораторную работу**

1. Разработать консольное приложение для расчёта интеграла.
2. Определить сложность алгоритма.
3. Провести отладку алгоритма и выписать в тетрадь значения интегральной суммы на каждой итерации.
4. Завершение работы программы должно осуществляться при явном выборе соответствующего пункта меню (например, 'Выход').
5. Выполнить задание согласно варианту.
6. Подготовить входные данные для проверки работы приложения.

**Варианты заданий на лабораторную работу**

**Задание.**

|  |  |  |
| --- | --- | --- |
| № варианта | Интеграл | Метод |
| 1 |  | Трапеций |
| 2 |  | Входящих прямоугольников |
| 3 |  | Выходящих прямоугольников |
| 4 |  | Трапеций |
| 5 |  | Входящих прямоугольников |
| 6 |  | Выходящих прямоугольников |
| 7 |  | Трапеций |
| 8 |  | Входящих прямоугольников |
| 9 |  | Выходящих прямоугольников |
| 10 |  | Трапеций |
| 11 |  | Входящих прямоугольников |
| 12 |  | Выходящих прямоугольников |
| 13 |  | Трапеций |
| 14 |  | Входящих прямоугольников |
| 15 |  | Выходящих прямоугольников |
| 16 |  | Трапеций |
| 17 |  | Входящих прямоугольников |
| 18 |  | Выходящих прямоугольников |
| 19 |  | Трапеций |
| 20 |  | Входящих прямоугольников |
| 21 |  | Выходящих прямоугольников |
| 22 |  | Трапеций |
| 23 |  | Входящих прямоугольников |
| 24 |  | Выходящих прямоугольников |
| 25 |  | Трапеций |
| 26 |  | Входящих прямоугольников |
| 27 |  | Выходящих прямоугольников |
| 28 |  | Трапеций |
| 29 |  | Входящих прямоугольников |
| 30 |  | Выходящих прямоугольников |

**Пример разработки приложения**

**Задание.**

Метод трапеции

.

Запишем метод для расчета подынтегральной функции:

using System;

double Function(double x)

{

return x \* x; // f(x) = x^2

}

Запишем метод для расчета первообразной функции:

double Antiderivative(double x)

{

return (x \* x \* x) / 3; // F(x) = x^3 / 3

}

Запишем метод трапеций для численного интегрирования:

double TrapezoidalIntegration(double a, double b, int n)

{

double h = (b - a) / n; // Шаг

double sum = 0.5 \* (Function(a) + Function(b)); // Начальная сумма

// Считаем сумму значений функции в промежуточных точках

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

{

double x = a + i \* h;

sum += Function(x);

}

return sum \* h; // Возвращаем результат

}

Запишем основную часть программы:

// Определяем границы интегрирования и количество подынтервалов

double a = 0; // Начало интервала

double b = 1; // Конец интервала

int n = 100; // Количество подынтервалов

// Численное интегрирование методом трапеций

double numericalResult = TrapezoidalIntegration(a, b, n);

// Аналитическое решение через первообразные

double analyticalResult = Antiderivative(b) - Antiderivative(a);

// Вывод результатов

Console.WriteLine($"Численное интегрирование (метод трапеций): {numericalResult}");

Console.WriteLine($"Аналитическое решение (формула Ньютона-Лейбница): {analyticalResult}");

Console.WriteLine($"Ошибка: {Math.Abs(numericalResult - analyticalResult)}");

**Контрольные вопросы**

1. Что такое метод в C# и как его объявить?
2. Как передать параметры в метод и вернуть значение из метода?
3. Что такое перегрузка методов? Приведите пример.
4. Какова формула для численного интегрирования методом трапеций?
5. Что такое первообразная функции и как она связана с интегрированием?
6. Как можно оценить ошибку численного интегрирования?