ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»

(СПбГУТ)

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ (ИТПИ)

кафедра программной инженерии и вычислительной техники (ПИиВТ)

Дисциплина: «Алгоритмы и структуры данных»

Лабораторная работа №3

Тема: «Численные методы»

Отчёт

Выполнили:

Коньков М. Д.

Семенихин А. Р.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Принял:

Дагаев А. В.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2025г.

# Цель работы:

Целью лабораторной работы является ознакомление с алгоритмами одномерной оптимизации, в частности методами дихотомии, итеративного поиска и золотого сечения, а также изучение методики оценки эффективности данных алгоритмов.

# Описание алгоритмов:

### Метод дихотомии:

Метод дихотомии основан на поиске экстремума функции одной переменной путём последовательного деления интервала неопределённости пополам.  
Алгоритм состоит из двух этапов:

* Поиск начального интервала, содержащего экстремум;
* Итеративное сужение интервала до заданной точности.

На каждом шаге вычисляется середина интервала x0​=(a+b)/2 и оценивается значение функции:

* Если f(x0​)<f(x0​+ε), граница интервала сдвигается влево;
* В противном случае — вправо.

Процесс продолжается до тех пор, пока длина интервала не станет меньше заданной погрешности ε.

### Метод золотого сечения:

Метод золотого сечения представляет собой оптимизированную версию метода дихотомии, минимизирующую количество вычислений функции.  
Идея основана на свойстве золотого сечения: отношение всей длины отрезка к его большей части такое же, как отношение большей части к меньшей.

На каждом шаге выбираются две точки x1​ и x2​ в пределах интервала [a,b] по формулам:

x1​=b−φb−a​,x2​=a+φb−a​

где φ=21+5

# Пошаговое сравнение:

В таблице 1 отображена пошаговая работа рекурсивного варианта дихотомического алгоритма при . В таблице 1: Пошаговая работа рекурсивного дихотомического алгоритма.

*Таблица 1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Границы промежутка /  Номер шага | l | m | r | Примечание |
| 1 | 0 | 4 | 2 |  |
| 2 | 0 | 1 | 2 | Сдвиг правой границы. |
| 3 | 0 | 0,5 | 1 | Сдвиг правой границы. |
| 4 | 0,5 | 0,75 | 1 | Сдвиг левой границы. |
| 5 | 0,5 | 0,625 | 0,75 | Сдвиг правой границы. |
| 6 | 0,625 | 0,6875 | 0,75 | Сдвиг левой границы. |
| 7 | 0,625 | 0,65625 | 0,6875 | Сдвиг правой границы. |
| 8 | 0,65625 | 0,671875 | 0,6875 | Останов. |

В таблице 2 отображена пошаговая работа итеративного варианта дихотомического алгоритма при . В таблице 2: Пошаговая работа итеративного дихотомического алгоритма.

*Таблица 2*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер шага | l | Точка 1 (m1) | Точка 2 (m2) | r | Примечание |
| 1 | 0 | 1.527864 | 2.472136 | 4 | - |
| 2 | 1.527864 | 2.472136 | 3.055728 | 4 | Сдвиг правой границы. |
| 3 | 2.472136 | 3.055728 | 3.416407 | 4 | Сдвиг левой границы |
| 4 | 3.055728 | 3.416407 | 3.639321 | 4 | Сдвиг левой границы. |
| ... | ... | … | ... | ... | До достижения точности |

В таблице 3 отображена пошаговая работа метода золотого сечения при . В таблице 2: Пошаговая работа метода золотого сечения.

### *Таблица 3*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Границы промежутка /  Номер шага | l | m | r | Примечание |
| 1 | 0 | 4 | 2 |  |
| 2 | 0 | 1 | 2 | Сдвиг правой границы. |
| 3 | 0 | 0,5 | 1 | Сдвиг правой границы. |
| 4 | 0,5 | 0,75 | 1 | Сдвиг левой границы. |
| 5 | 0,5 | 0,625 | 0,75 | Сдвиг правой границы. |
| 6 | 0,625 | 0,6875 | 0,75 | Сдвиг левой границы. |
| 7 | 0,625 | 0,65625 | 0,6875 | Сдвиг правой границы. |
| 8 | 0,65625 | 0,671875 | 0,6875 | Останов. |

Примечание: значения точек рассчитываются согласно числу золотого сечения.

# Описание программы:

Программа для выполнения работы реализует три алгоритма поиска экстремума одномерной функции: рекурсивный метод дихотомии, итеративный метод дихотомии и метод золотого сечения.

Функционал программы включает:

* Запрос длины интервала и требуемой точности поиска;
* Запрос количества итераций для усреднения результатов измерения времени;
* Запуск всех трёх алгоритмов последовательно;
* Вывод среднего времени выполнения каждого из алгоритмов.

Программа написана на языке C++ в текстовом редакторе Neovim с использованием конфигурации LazyVim. Работа велась в операционной системе Linux.

# Полученные результаты:

Результаты оценки временной сложности алгоритмов (в миллисекундах) представлены в таблице 4 и отображены на рисунке 1. Далее показана **зависимость времени работы алгоритмов одномерной оптимизации от заданной точности вычислений (в миллисекундах).**

*Таблица 4*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Точность / Алгоритм | **10⁻⁴** | **10⁻⁵** | **10⁻⁶** | **10⁻⁷** | **10⁻⁸** | **10⁻⁹** |
| **Рекурсивный алгоритм** | 6,39E-07 | 6,45E-07 | 6,86E-07 | 7,24E-07 | 7,81E-07 | 8,79E-07 |
| **Итеративный алгоритм** | 6,02E-07 | 6,20E-07 | 6,54E-07 | 6,90E-07 | 7,32E-07 | 7,78E-07 |
| **Золотое сечение** | 7,558E-07 | 7,717E-07 | 8,38E-07 | 8,89E-07 | 1,02E-06 | 1,18E-06 |

Рисунок 1. Результаты работы двух алгоритмов

точности вычислений.

Данные, отображенные в таблице и на рисунке, показывают, что рекурсивный и итеративный дихотомические алгоритмы имеют одинаковую асимптотическую сложность; тем не менее, итеративный вариант работает в среднем быстрее, а вот алгоритм золотого сечения оказался самым медленным, его сложность тоже асимптотическая. Сходятся все эти методы линейно, то есть в постоянной пропорции на каждом шаге.

# Вывод:

В ходе проведённой лабораторной работы было написана программа на языке C++ для реализации трёх алгоритмов одномерной оптимизации: итеративного, рекурсивного и методом золотого сечения. Также были получены следующие результаты:

 **Итеративный алгоритм** стабильно показывает **самое лучшее время выполнения** по сравнению с остальными двумя методами при всех уровнях точности. Его стоит использовать, когда требуется быстрое выполнение и точность не запредельная.

 **Рекурсивный алгоритм** медленнее итеративного на всём диапазоне точностей, но он всё же быстрее **метода золотого сечения**. Он может быть полезен, когда нужно сохранить структуру решения, основанную на рекурсии (например, для удобства реализации или теоретического анализа), но не при максимальной оптимизации по скорости.

 **Метод золотого сечения** оказывается **самым медленным** из всех представленных алгоритмов, причём его время выполнения растёт значительно быстрее при увеличении точности. Это объясняется тем, что метод золотого сечения требует дополнительных вычислений (например, вычислений с иррациональными числами) и более сложных процедур сравнения.

# Листинг кода:

#include <iostream>

#include <vector>

#include <cmath>

#include <ctime>

double f(double x)

{

return 2.755 \* x \* x - 3.7743 \* x + 1.2311;

}

// Рекурсивный метод дихотомии

double f1(double l, double r, double pre)

{

double m = (r - l) / 2.0 + l;

if (std::abs(f(r) - f(l)) <= pre)

{

return m;

}

if (f(m - pre) < f(m + pre))

{

return f1(l, m, pre);

}

else

{

return f1(m, r, pre);

}

}

// Итеративный метод дихотомии

double f2(double l, double r, double pre)

{

double m = (r - l) / 2.0 + l;

while (std::abs(f(r) - f(l)) > pre)

{

if (f(m - pre) < f(m + pre))

{

r = m;

}

else

{

l = m;

}

m = (r - l) / 2.0 + l;

}

return m;

}

// Метод золотого сечения

double f3(double l, double r, double pre)

{

const double phi = (1 + std::sqrt(5)) / 2.0; // Число золотого сечения

double x1 = r - (r - l) / phi;

double x2 = l + (r - l) / phi;

while (std::abs(r - l) > pre)

{

if (f(x1) < f(x2))

{

r = x2;

x2 = x1;

x1 = r - (r - l) / phi;

}

else

{

l = x1;

x1 = x2;

x2 = l + (r - l) / phi;

}

}

return (l + r) / 2.0;

}

int main()

{

srand(static\_cast<unsigned int>(time(nullptr)));

char c = ' ';

while (c != 'q')

{

int r = 0;

double eps, rg;

std::cout << "Precision as power of 10:" << std::endl;

std::cin >> rg;

eps = std::pow(10.0, -1.0 \* rg);

std::cout << "Iterations:" << std::endl;

std::cin >> r;

double time\_f1\_sum = 0, time\_f2\_sum = 0, time\_golden\_sum = 0;

const double a = 0, b = 4;

for (int k = 0; k < r; k++)

{

clock\_t start = clock();

f1(a, b, eps);

clock\_t end = clock();

double seconds = static\_cast<double>(end - start) / CLOCKS\_PER\_SEC;

time\_f1\_sum += seconds;

start = clock();

f2(a, b, eps);

end = clock();

seconds = static\_cast<double>(end - start) / CLOCKS\_PER\_SEC;

time\_f2\_sum += seconds;

start = clock();

f3(a, b, eps);

end = clock();

seconds = static\_cast<double>(end - start) / CLOCKS\_PER\_SEC;

time\_golden\_sum += seconds;

}

double time\_1 = time\_f1\_sum / static\_cast<double>(r);

double time\_2 = time\_f2\_sum / static\_cast<double>(r);

double time\_3 = time\_golden\_sum / static\_cast<double>(r);

std::cout << "The time (recursion): " << time\_1 << " seconds" << std::endl;

std::cout << "The time (iteration): " << time\_2 << " seconds" << std::endl;

std::cout << "The time (golden section): " << time\_3 << " seconds" << std::endl;

std::cout << "Enter any char to continue; enter 'q' to exit." << std::endl;

std::cin >> c;

}

return 0;

}