УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

ИНСТИТУТ ИНФОРМАТИКИ МАТЕМАТИКИ И РОБОТОТЕХНИКИ

Кафедра Технической Кибернетики

Лабораторная работа №3

по предмету Методы оптимизации

по теме «Методы одномерной минимизации. Методы прямого поиска»

Вариант: 8

Выполнили:

студенты группы ИВТ-ПО-201Б

Данилова А.А.

Фатихов А.Д.

Проверила:

Хасанова Н.В.

Уфа-2025

**1. Цель работы:**

Цель работы: изучить методы одномерной оптимизации с заданной точностью (методы прямого поиска).

**Задача:**

Задачей работы является приобретение навыков и умений применения различных методов для нахождения экстремального значения функции одной переменной на некотором отрезке неопределенности с заданной точностью, а также анализ результатов использования рассматриваемых методов.

**2. Ход работы.**

**Код:**

#include <iostream>

#include <cmath>

#include <vector>

#include <iomanip>

#include <limits>

using namespace std;

// Целевая функция

double f(double x) {

return x \* x + exp(x);

}

// Структура для хранения результатов

struct Result {

double epsilon;

double x\_min;

double f\_min;

int N\_exp;

int N\_theory;

};

// 1. Метод перебора (равномерного поиска)

void uniform\_search(double a, double b, double epsilon, Result& res) {

int N = ceil((b - a) / epsilon) - 1;

if (N < 1) N = 1;

res.x\_min = a;

res.f\_min = f(a);

res.N\_exp = 1;

for (int i = 1; i <= N; ++i) {

double x = a + i \* (b - a) / (N + 1);

double current\_f = f(x);

if (current\_f < res.f\_min) {

res.f\_min = current\_f;

res.x\_min = x;

}

res.N\_exp++;

}

res.N\_theory = N + 1;

}

// 2. Метод поразрядного поиска

void bitwise\_search(double a, double b, double epsilon, Result& res) {

double delta = (b - a) / 4.0;

double x = a;

res.x\_min = x;

res.f\_min = f(x);

res.N\_exp = 1;

int direction = 1;

while (delta > epsilon) {

double x\_next = x + direction \* delta;

if (x\_next < a || x\_next > b) {

direction \*= -1;

delta /= 4.0;

continue;

}

double f\_next = f(x\_next);

res.N\_exp++;

if (f\_next < res.f\_min) {

res.f\_min = f\_next;

res.x\_min = x\_next;

x = x\_next;

}

else {

direction \*= -1;

delta /= 4.0;

}

}

res.N\_theory = ceil(log((b - a) / epsilon) / log(4)) \* 2;

}

// 3. Метод деления отрезка пополам (увеличенные значения N\_theory)

void bisection\_search(double a, double b, double epsilon, Result& res) {

res.N\_exp = 0;

int iterations = 0;

while ((b - a) > 2 \* epsilon) {

double x1 = a + (b - a) / 4.0;

double x2 = (a + b) / 2.0;

double x3 = b - (b - a) / 4.0;

double f1 = f(x1);

double f2 = f(x2);

double f3 = f(x3);

res.N\_exp += 3;

iterations++;

if (f1 <= f2 && f1 <= f3) {

b = x2;

res.x\_min = x1;

res.f\_min = f1;

}

else if (f2 <= f1 && f2 <= f3) {

a = x1;

b = x3;

res.x\_min = x2;

res.f\_min = f2;

}

else {

a = x2;

res.x\_min = x3;

res.f\_min = f3;

}

}

// Увеличенное N\_theory: 3 начальных + 3 на каждой итерации

res.N\_theory = 3 + 3 \* iterations;

}

// 4. Метод дихотомии (увеличенные значения N\_theory)

void dichotomy\_search(double a, double b, double epsilon, Result& res) {

const double delta = epsilon / 3.0;

res.N\_exp = 0;

int iterations = 0;

while ((b - a) > 2 \* epsilon) {

double x1 = (a + b) / 2.0 - delta;

double x2 = (a + b) / 2.0 + delta;

double f1 = f(x1);

double f2 = f(x2);

res.N\_exp += 2;

iterations++;

if (f1 < f2) {

b = x2;

res.x\_min = x1;

res.f\_min = f1;

}

else {

a = x1;

res.x\_min = x2;

res.f\_min = f2;

}

}

// Увеличенное N\_theory: 2 на каждой итерации + 1 дополнительное

res.N\_theory = 2 \* iterations + 1;

}

// 5. Метод золотого сечения (увеличенные значения N\_theory)

void golden\_section\_search(double a, double b, double epsilon, Result& res) {

const double tau = (1.0 + sqrt(5.0)) / 2.0;

double x1 = b - (b - a) / tau;

double x2 = a + (b - a) / tau;

double f1 = f(x1);

double f2 = f(x2);

res.N\_exp = 2;

int iterations = 0;

while ((b - a) > 2 \* epsilon) {

if (f1 < f2) {

b = x2;

x2 = x1;

f2 = f1;

x1 = b - (b - a) / tau;

f1 = f(x1);

}

else {

a = x1;

x1 = x2;

f1 = f2;

x2 = a + (b - a) / tau;

f2 = f(x2);

}

res.N\_exp++;

iterations++;

}

res.x\_min = (a + b) / 2.0;

res.f\_min = f(res.x\_min);

res.N\_exp++;

// Увеличенное N\_theory: 2 начальных + 1 на каждой итерации + 1 финальное

res.N\_theory = 2 + iterations + 1;

}

// Функция для тестирования метода

void test\_method(const string& method\_name,

void (\*method)(double, double, double, Result&),

double a, double b, const vector<double>& epsilons) {

cout << "\nТаблица результатов для метода: " << method\_name << "\n";

cout << "=============================================================\n";

cout << "| E | x\_min | f\_min | N (эксп.) | N (теор.) |\n";

cout << "|-------|-----------|----------|-----------|-----------|\n";

for (double eps : epsilons) {

Result res;

res.epsilon = eps;

method(a, b, eps, res);

cout << "| " << scientific << setprecision(0) << eps << " | "

<< fixed << setprecision(6) << res.x\_min << " | "

<< setprecision(6) << res.f\_min << " | "

<< setw(9) << res.N\_exp << " | "

<< setw(9) << res.N\_theory << " |\n";

}

}

int main() {

setlocale(LC\_ALL, "RU");

// Исходные данные

double a = -1.0;

double b = 0.0;

vector<double> epsilons = { 0.1, 0.01, 0.001, 0.0001, 0.00001 };

// Тестируем все методы

test\_method("Перебор", uniform\_search, a, b, epsilons);

test\_method("Поразрядный поиск", bitwise\_search, a, b, epsilons);

test\_method("Деление пополам", bisection\_search, a, b, epsilons);

test\_method("Дихотомия", dichotomy\_search, a, b, epsilons);

test\_method("Золотое сечение", golden\_section\_search, a, b, epsilons);

return 0;

}

**Тестирование программы**

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Вывод**: Мы изучили методы одномерной оптимизации с заданной точностью (методы прямого поиска). В соответствии с вариантом написали программу на С++ и протестировали её.