Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе №1 «Прямые методы одномерной минимизации функций»

по дисциплине «**Методы оптимизации**»

Авторы:

Антонов Кирилл Владимирович М3237

Чмыхалов Артемий Витальевич М3237

Якупова Айша Рустемовна М3234

Факультет: ИТИП

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург 2021

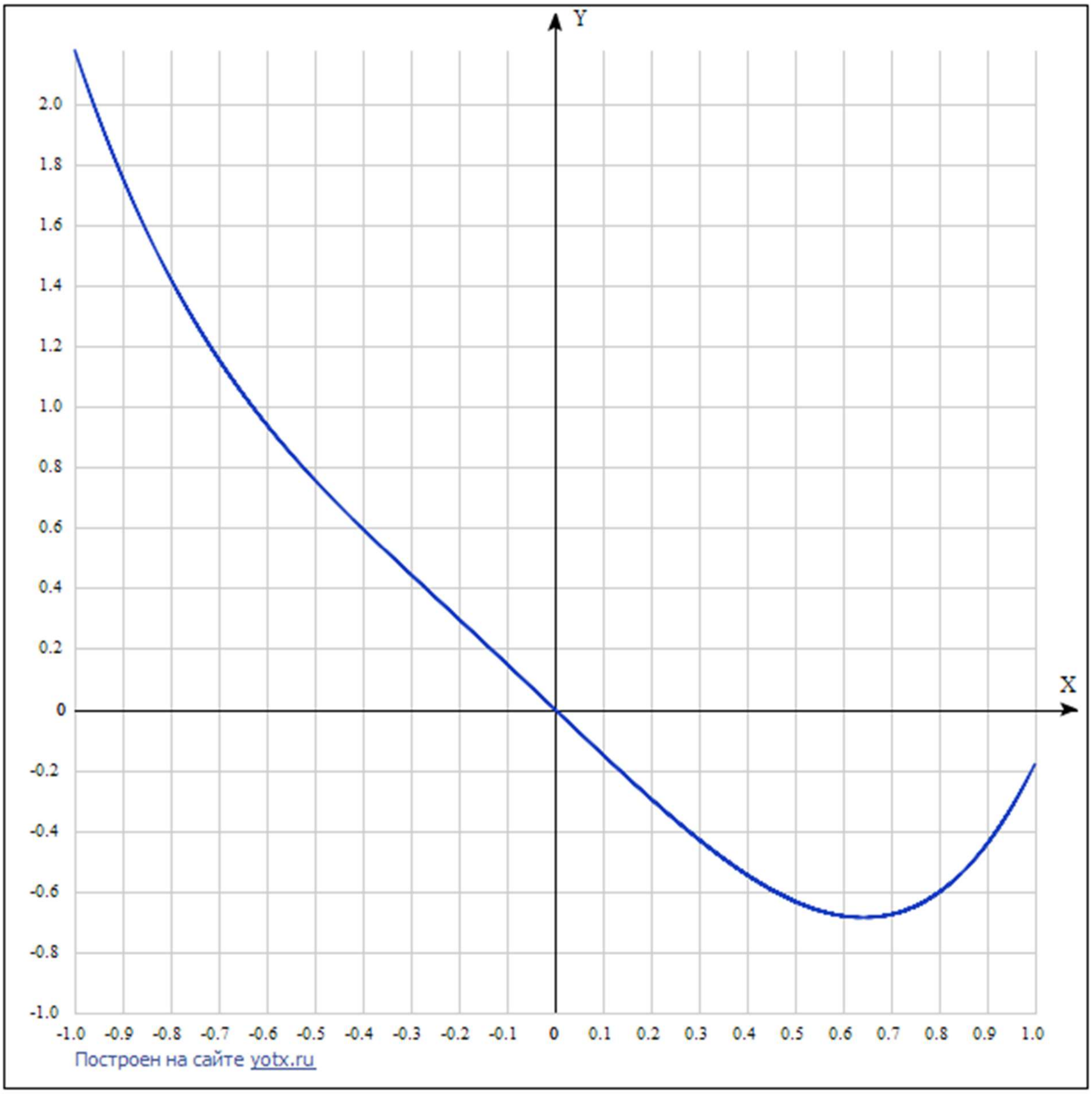
Цель работы:

Оценить работу прямых методов одномерной минимизации функций. Реализовать методы

1. Метод Дихотомии
2. Метод Золотого сечения
3. Метод Фибоначчи
4. Метод парабол
5. Комбинированный метод Брента

Аналитическое нахождение минимума:

Нам дана функция f(x) = $x^4 + 1.5arctg(x)$:



Найдём её экстремум на отрезке [-1;1]. Для этого вычислим производную:

f'(x) = $4x^3 - \frac{1.5}{1+x^2} = \frac{4x^3(1+x^2)-1.5}{1+x^2} = 0 \Rightarrow 4x^3(1+x^2) - 1.5 = 4x^5 + 4x^3 - 1.5 = 0$

Получаем, что $x_{min} = 0.6426$, f(x_{min}) = -0.6862

Описание методов:

I. Метод дихотомии (Допустимая погрешность: $4 * 10^{-16}$)

a	b	lencur/lenprev	x1	x2	f1	f2
-1	1	1	0	0	0	0
0	1	0.5	0.5	0.5	-0.633	-0.633
0.5	1	0.5	0.75	0.75	-0.6488	-0.6488
0.5	0.75	0.5	0.625	0.625	-0.6853	-0.6853
0.625	0.75	0.5	0.6875	0.6875	-0.68	-0.68
0.625	0.6875	0.5	0.6562	0.6563	-0.6857	-0.6857
0.625	0.6563	0.5	0.6406	0.6406	-0.6862	-0.6862
0.6406	0.6563	0.5	0.6484	0.6484	-0.6861	-0.6861
0.6406	0.6484	0.5	0.6445	0.6445	-0.6862	-0.6862
0.6406	0.6445	0.5	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6445	0.5	0.6436	0.6436	-0.6862	-0.6862
0.6426	0.6436	0.5	0.6431	0.6431	-0.6862	-0.6862
0.6426	0.6431	0.5	0.6428	0.6428	-0.6862	-0.6862
0.6426	0.6428	0.5	0.6427	0.6427	-0.6862	-0.6862
0.6426	0.6427	0.5	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5001	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5002	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5003	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5007	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5013	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5026	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5052	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5103	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5201	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5387	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.5718	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.6256	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.7008	0.6426	0.6426	-0.6862	-0.6862

II. Метод золотого сечения (Допустимая погрешность: $2 * 10^{-8}$)

a	b	lencur/lenprev	x1	x2	f1	f2
-1	1	1	-0.2361	0.2361	0.3508	-0.3446
-0.2361	1	0.618	0.2361	0.5279	-0.3446	-0.6509
0.2361	1	0.618	0.5279	0.7082	-0.6509	-0.6728
0.5279	1	0.618	0.7082	0.8197	-0.6728	-0.5785
0.5279	0.8197	0.618	0.6393	0.7082	-0.6862	-0.6728
0.5279	0.7082	0.618	0.5967	0.6393	-0.6802	-0.6862
0.5967	0.7082	0.618	0.6393	0.6656	-0.6862	-0.6846
0.5967	0.6656	0.618	0.6231	0.6393	-0.6851	-0.6862
0.6231	0.6656	0.618	0.6393	0.6494	-0.6862	-0.6861
0.6231	0.6494	0.618	0.6331	0.6393	-0.686	-0.6862
0.6331	0.6494	0.618	0.6393	0.6432	-0.6862	-0.6862
0.6393	0.6494	0.618	0.6432	0.6455	-0.6862	-0.6862
0.6393	0.6455	0.618	0.6417	0.6432	-0.6862	-0.6862
0.6417	0.6455	0.618	0.6432	0.6441	-0.6862	-0.6862
0.6417	0.6441	0.618	0.6426	0.6432	-0.6862	-0.6862
0.6417	0.6432	0.618	0.6423	0.6426	-0.6862	-0.6862
0.6423	0.6432	0.618	0.6426	0.6428	-0.6862	-0.6862
0.6423	0.6428	0.618	0.6425	0.6426	-0.6862	-0.6862
0.6425	0.6428	0.618	0.6426	0.6427	-0.6862	-0.6862
0.6425	0.6427	0.618	0.6425	0.6426	-0.6862	-0.6862
0.6425	0.6427	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6425	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6425	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6425	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6425	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6425	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6425	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6425	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6425	0.6426	0.618	0.6425	0.6426	-0.6862	-0.6862
0.6425	0.6426	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	0.618	0.6425	0.6425	-0.6862	-0.6862
0.6425	0.6425	1	0.6425	0.6425	-0.6862	-0.6862

III. Метод Фибоначчи (Допустимая погрешность: $3 * 10^{-16}$)

a	b	lencur/lenprev	x1	x2	f1	f2
-1	1	1	-0.2361	0.2361	0.3508	-0.3446
-0.2361	1	0.618	0.2361	0.5279	-0.3446	-0.6509
0.2361	1	0.618	0.5279	0.7082	-0.6509	-0.6728
0.5279	1	0.618	0.7082	0.8197	-0.6728	-0.5785
0.5279	0.8197	0.618	0.6393	0.7082	-0.6862	-0.6728
0.5279	0.7082	0.618	0.5967	0.6393	-0.6802	-0.6862
0.5967	0.7082	0.618	0.6393	0.6656	-0.6862	-0.6846
0.5967	0.6656	0.618	0.6231	0.6393	-0.6851	-0.6862
0.6231	0.6656	0.618	0.6393	0.6494	-0.6862	-0.6861
0.6231	0.6494	0.618	0.6331	0.6393	-0.686	-0.6862
0.6331	0.6494	0.618	0.6393	0.6432	-0.6862	-0.6862
0.6393	0.6494	0.618	0.6432	0.6455	-0.6862	-0.6862
0.6393	0.6455	0.618	0.6417	0.6432	-0.6862	-0.6862
0.6417	0.6455	0.618	0.6432	0.6441	-0.6862	-0.6862
0.6417	0.6441	0.618	0.6426	0.6432	-0.6862	-0.6862
0.6417	0.6432	0.618	0.6423	0.6426	-0.6862	-0.6862
0.6423	0.6432	0.618	0.6426	0.6428	-0.6862	-0.6862
0.6423	0.6428	0.618	0.6425	0.6426	-0.6862	-0.6862
0.6425	0.6428	0.618	0.6426	0.6427	-0.6862	-0.6862
0.6425	0.6427	0.618	0.6425	0.6426	-0.6862	-0.6862
0.6425	0.6427	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6427	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.6181	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.618	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.6182	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.6176	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.619	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.6154	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.625	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	0.6	0.6426	0.6426	-0.6862	-0.6862
0.6426	0.6426	1	0.6426	0.6426	-0.6862	-0.6862

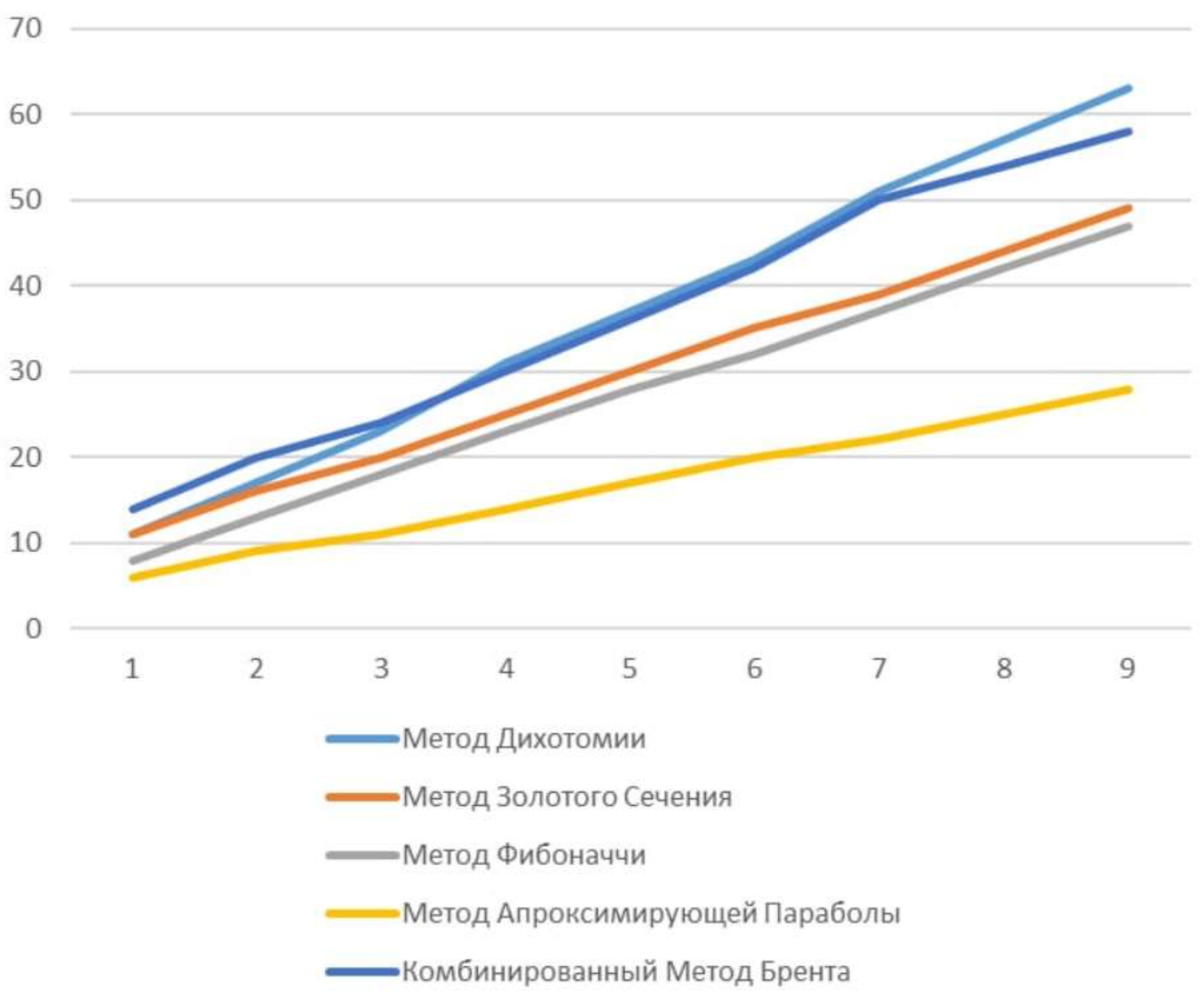
IV. Метод парабол (Допустимая погрешность: $4 * 10^{-16}$)

a	b	lencur/lenprev	x*	f*
-1	1	1	0	0
-1	0.589	0.7945	0.589	-0.6781
0.5376	1	0.291	0.5376	-0.6564
0.5376	0.6229	0.1844	0.6229	-0.6851
0.589	0.6333	0.5191	0.6333	-0.686
0.6229	0.6389	0.3627	0.6389	-0.6862
0.6333	0.641	0.4769	0.641	-0.6862
0.6389	0.6419	0.3942	0.6419	-0.6862
0.641	0.6423	0.4517	0.6423	-0.6862
0.6419	0.6425	0.4108	0.6425	-0.6862
0.6423	0.6426	0.4392	0.6426	-0.6862
0.6425	0.6426	0.4192	0.6426	-0.6862
0.6426	0.6426	0.4331	0.6426	-0.6862
0.6426	0.6426	0.4233	0.6426	-0.6862
0.6426	0.6426	0.4302	0.6426	-0.6862
0.6426	0.6426	0.4253	0.6426	-0.6862
0.6426	0.6426	0.4287	0.6426	-0.6862
0.6426	0.6426	0.4264	0.6426	-0.6862
0.6426	0.6426	0.4279	0.6426	-0.6862
0.6426	0.6426	0.4263	0.6426	-0.6862
0.6426	0.6426	0.4298	0.6426	-0.6862
0.6426	0.6426	0.4327	0.6426	-0.6862
0.6426	0.6426	1	0.6426	-0.6862

V. Комбинированный метод Брента (Допустимая погрешность: $5 * 10^{-16}$)

a	b	lencur/lenprev	x*	f*
-1	1	1	-0.2361	0.3508
-1	1	1	0.2361	-0.3446
-0.2361	1	0.618	0.2361	-0.3446
-0.2361	1	0.618	0.5279	-0.6509
0.2361	1	0.618	0.5279	-0.6509
0.2361	1	0.618	0.7082	-0.6728
0.5279	1	0.618	0.7082	-0.6728
0.5279	1	0.618	0.8197	-0.5785
0.5279	0.8197	0.618	0.2361	-0.3446
0.5279	0.8197	0.618	0.7082	-0.6728
0.5279	0.8197	0.618	0.6363	-0.6861
0.5279	0.7082	0.618	0.6363	-0.6861
0.5279	0.7082	0.618	0.6395	-0.6862
0.6363	0.7082	0.3985	0.6395	-0.6862
0.6363	0.7082	0.3985	0.6424	-0.6862
0.6395	0.7082	0.9565	0.6424	-0.6862
0.6395	0.7082	0.9565	0.6675	-0.6843
0.6395	0.6675	0.4082	0.6363	-0.6861
0.6395	0.6675	0.4082	0.6424	-0.6862
0.6395	0.6675	0.4082	0.652	-0.686
0.6395	0.652	0.4463	0.6363	-0.6861
0.6395	0.652	0.4463	0.6424	-0.6862
0.6395	0.652	0.4463	0.6461	-0.6862
0.6395	0.6461	0.5262	0.6363	-0.6861
0.6395	0.6461	0.5262	0.6424	-0.6862
0.6395	0.6461	0.5262	0.6426	-0.6862
0.6424	0.6461	0.5565	0.6426	-0.6862
0.6424	0.6461	0.5565	0.6439	-0.6862
0.6424	0.6439	0.4232	0.6395	-0.6862
0.6424	0.6439	0.4232	0.6426	-0.6862
0.6424	0.6439	0.4232	0.6426	-0.6862
0.6426	0.6439	0.8423	0.6426	-0.6862
0.6426	0.6439	0.8423	0.6431	-0.6862
0.6426	0.6431	0.3841	0.6424	-0.6862
0.6426	0.6431	0.3841	0.6426	-0.6862
0.6426	0.6431	0.3841	0.6428	-0.6862
0.6426	0.6428	0.3875	0.6424	-0.6862
0.6426	0.6428	0.3875	0.6426	-0.6862
0.6426	0.6428	0.3875	0.6426	-0.6862
0.6426	0.6428	0.9768	0.6426	-0.6862
0.6426	0.6428	0.9768	0.6427	-0.6862
0.6426	0.6427	0.3824	0.6426	-0.6862
0.6426	0.6427	0.3824	0.6426	-0.6862
0.6426	0.6427	0.3824	0.6427	-0.6862
0.6426	0.6427	0.3832	0.6426	-0.6862
0.6426	0.6427	0.3832	0.6426	-0.6862
0.6426	0.6427	0.3832	0.6426	-0.6862
0.6426	0.6426	0.3851	0.6426	-0.6862
0.6426	0.6426	0.3851	0.6426	-0.6862
0.6426	0.6426	0.3851	0.6426	-0.6862
0.6426	0.6426	0.3901	0.6426	-0.6862
0.6426	0.6426	0.3901	0.6426	-0.6862
0.6426	0.6426	0.3901	0.6426	-0.6862
0.6426	0.6426	0.0337	0.6426	-0.6862
0.6426	0.6426	0.0337	0.6426	-0.6862
0.6426	0.6426	0.142	0.6426	-0.6862
0.6426	0.6426	0.142	0.6426	-0.6862
0.6426	0.6426	0.142	0.6426	-0.6862
0.6426	0.6426	0.403	0.6426	-0.6862
0.6426	0.6426	0.403	0.6426	-0.6862

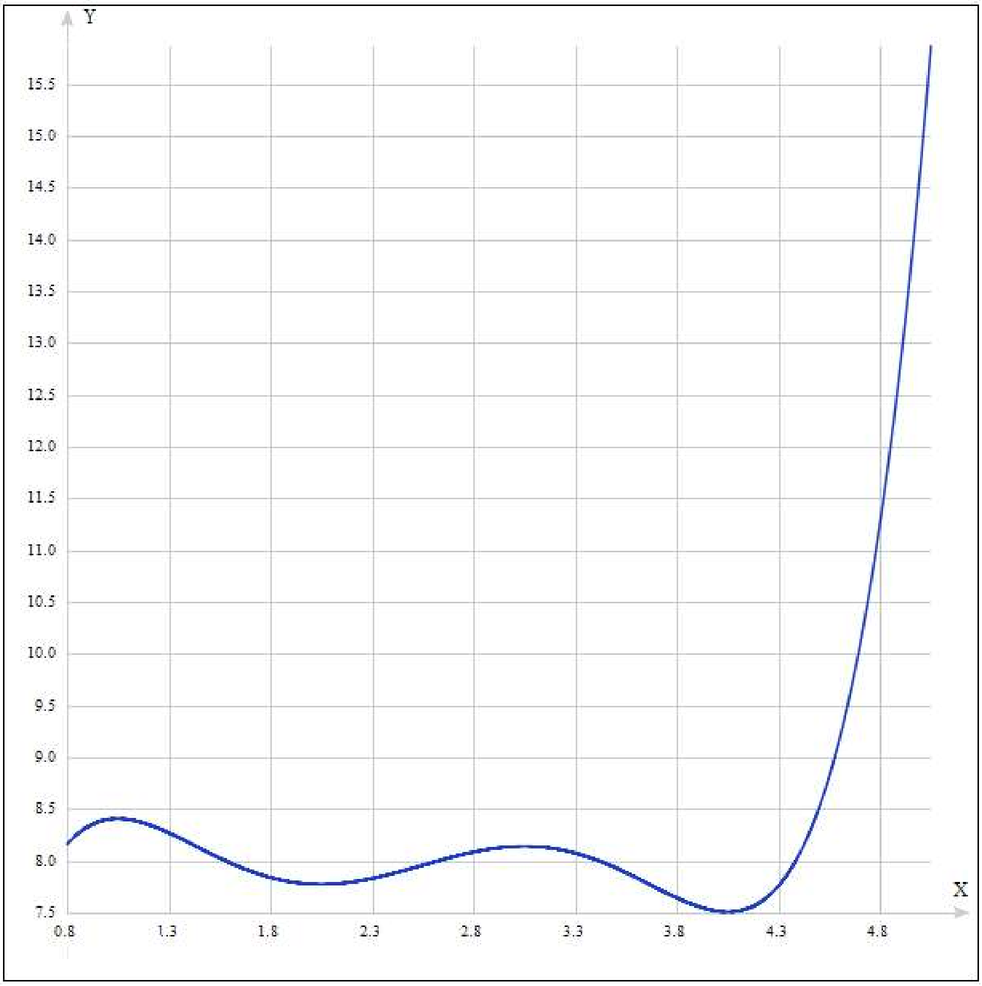
График зависимости количества вычислений от логарифма задаваемой точности:



Вывод: Исходя из графиков заметим, что наилучший вариант для нашей функции – метод золотого сечения, он позволяет нам достигать высокой точности при меньшем количестве вычислений. Зависимость количества вычислений от логарифма заданной точности практически линейная, в отличие от методов дихотомии, Фибоначчи, парабол, Брента.

Использование методов минимизации функций на многомодальных функциях:

Рассмотрим функцию y = 24 * x – 25 * x + $\frac{35 * x^3}{3}$ - $\frac{5 * x^4}{2}$ + $\frac{x^5}{5}$ на отрезке [0.75, 5]



Метод дихотомии:
Точка минимума = 1.99999965444267   Минимум функции = 7.73333333333349

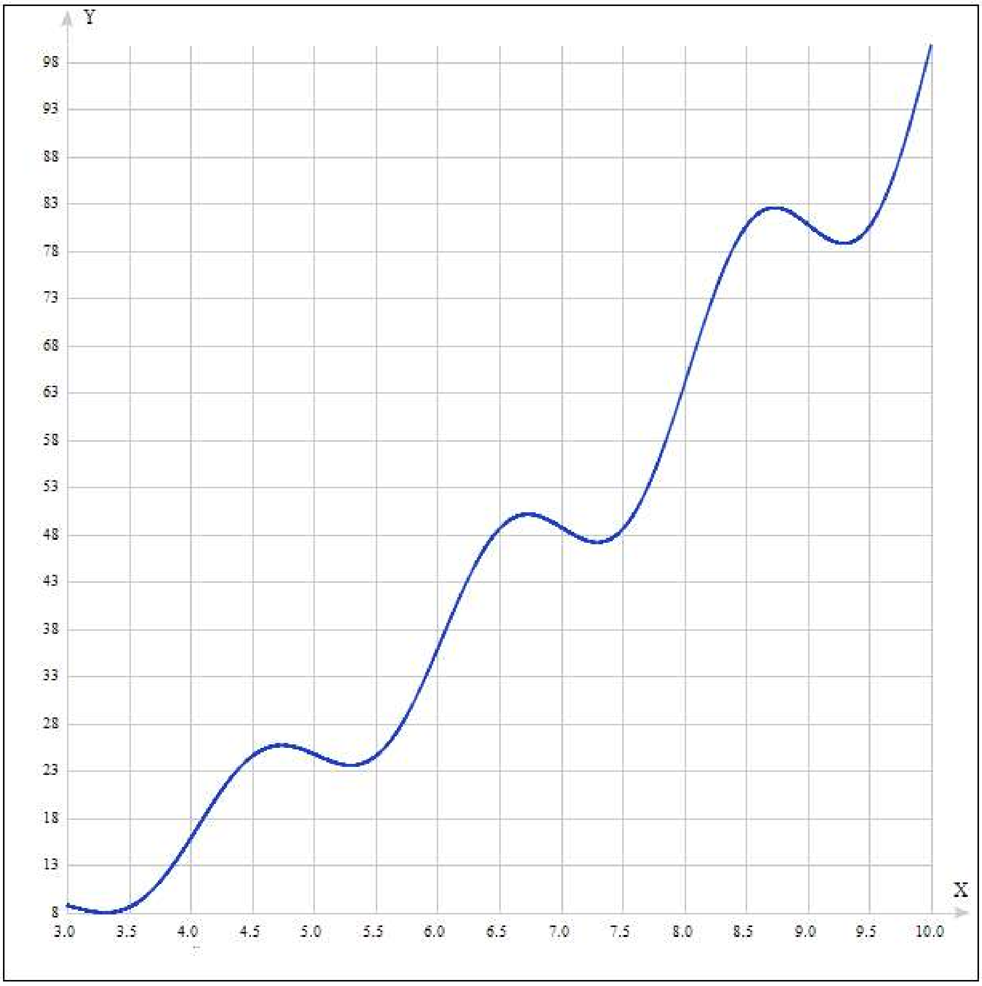
Метод золотого сечения:
Точка минимума = 1.99986906815959 Минимум функции = 7.73333335047721

Метод Фибоначчи:
Точка минимума = 2.00000000663215 Минимум функции = 7.73333333333331

Метод парабол:
Точка минимума = 2.00000005556338 Минимум функции = 7.73333333333332

Комбинированный метод Брента:
Точка минимума = 1.99999999429516 Минимум функции = 7.7333333333333

Рассмотрим другую функцию y = x^2 + x * $\sin \pi x$ на отрезке [3; 10]



Метод дихотомии:
Точка минимума = 5.29989908015435 Минимум функции = 23.8022097229733

Метод золотого сечения:
Точка минимума = 3.31199901902458 Минимум функции = 8.21841038272098

Метод Фибоначчи:
Точка минимума = 3.31203358498817 Минимум функции = 8.21841036739555

Метод парабол:
Точка минимума = 3.3120335859281 Минимум функции = 8.21841036739555

Комбинированный метод Брента:
Точка минимума = 3.31203358546464 Минимум функции = 8.21841036739555

Вывод: мы реализовали несколько прямых методов одномерной минимизации функции и применили их к заданной функции. Мы убедились в том, что для унимодальных функций эти методы работают хорошо и находят минимум с заданной точностью. Мы применили эти методы на многомодальных функциях и заметили, что эти методы находят локальный минимум функции, который не обязательно является глобальным, поэтому эти методы неприменимы к многомодальным функциям.

Программный код:
“functions.h”:

```cpp
#pragma once

#include <cmath>

double func1(double x) {
	return x * x + exp(-0.35 * x);
}

double func2(double x) {
	return x * x * x * x - 1.5 * atan(x);
}

double func3(double x) {
	return x * sin(x) + 2 * cos(x);
}

double func4(double x) {
	return x - log(x);
}

double func5(double x) {
	return 10 * x * log(x) - x * x / 2;
}

double func6(double x) {
	return -5 * x * x * x * x * x + 4 * x * x * x * x - 12 * x * x * x + 11 * x * x -
2 * x + 1;
}

double func7(double x) {
	return log10(x - 2) * log10(x - 2) + log10(10 - x) * log10(10 - x) - pow(x, 0.2);
}

double func8(double x) {
	return 3 * x * sin(0.75 * x) + exp(-2 * x);
}

double func9(double x) {
	return exp(3 * x) + 5 * exp(-2 * x);
}

double func10(double x) {
	return 0.2 * x * log10(x) + (x - 2.3) * (x - 2.3);
}

double func11(double x) {
	return x * (x - 2) * (x - 3);
}

double func12(double x) {
	const double PI = 3.14159265358979323846З;
	return x * x + x * sin(PI * x);
}

double func13(double x) {
	const double PI = 3.14159265358979323846З;
	return x * x + x * x * sin(PI * x);
}
```

“All_inclusive.h”:

```cpp
#pragma once

#include <iostream>
#include <algorithm>
#include <vector>
#include <fstream>
#include "functions.h"
```

```
#define function func2
double a = -1;
double b = 1;

double eps = 1e-8;

double extr(double a, double b) {
	double res = a;
	double min = 1000000;
	for (double i = a; i < b; i += eps) {
		double f = function(res);
		if (f <= min) {
			res = i;
			min = f;
		}
	}
	return function(res);
}
```

"Dychotomy_method.h":

```
#pragma once

#include "All_inclusive.h"

struct Dychotomy_method {
	double operator()(double a, double b) {
		double d = eps;
		while ((b - a) / 2 > eps) {

			double x1 = (a + b - d) / 2;
			double x2 = (a + b + d) / 2;
			double f1 = function(x1);
			double f2 = function(x2);
			if (f1 <= f2)
				b = x2;
			else
				a = x1;
		}

		double x = (a + b) / 2;
		double f = function(x);
		return f;
	}
};
```

"Golden_ratio_method.h":

```
#pragma once

#include "All_inclusive.h"

struct Golden_ratio_method {
	double operator()(double a, double b, double eps) {
		double t = (sqrt(5) - 1) / 2;

		double x1 = a + (1 - t) * (b - a);
		double x2 = a + t * (b - a);

		double f1 = function(x1);
		double f2 = function(x2);
		double eps_n = (b - a) / 2;

		while (eps_n > eps) {
			if (f1 - f2 <= eps) {
				b = x2;
				x2 = x1;
				f2 = f1;
				x1 = a + (1 - t) * (b - a);
				f1 = function(x1);
			}
			else {
```

```
					a = x1;
					x1 = x2;
					f1 = f2;
					x2 = a + t * (b - a);
					f2 = function(x2);
				}
				eps_n *= t;
			}

			double x = (a + b) / 2;
			double f = function(x);
			return f;
		}
};
```

“Fibonacci_method.h”:

```
#pragma once

#include "All_inclusive.h"

struct Fibonacci_method {

	double operator()(double a, double b) {

		computation_fib();

		int n = 0;
		int k = 0;

		double eps_n = (b - a) / eps;
		while (eps_n > fib[n]) {
			n++;
		}
		double x1 = a + ((double)fib[n - 2] / fib[n]) * (b - a);
		double x2 = a + ((double)fib[n - 1] / fib[n]) * (b - a);
		double f1 = function(x1);
		double f2 = function(x2);

		while (k != n - 3) {
			if (f1 > f2) {
				a = x1;
				x1 = x2;
				f1 = f2;
				x2 = a + ((double)fib[n - k - 2] / fib[n - k - 1]) * (b - a);
				f2 = function(x2);
			}
			else {
				b = x2;
				x2 = x1;
				f2 = f1;
				x1 = a + ((double)fib[n - k - 3] / fib[n - k - 1]) * (b - a);
				f1 = function(x1);
			}
			k++;
		}

		double x = (x1 + x2) / 2;
		double f = function(x);
		return f;
	}

private:
	int n = 45;
	std::vector<int> fib;

	void computation_fib() {
		fib.resize(46);
		fib[0] = 1;
		fib[1] = 1;
		for (int i = 2; i < 46; i++) {
			fib[i] = fib[i - 1] + fib[i - 2];
		}
	}
```

```
};
```

“Parabola_method.h”:

```
#pragma once

#include "All_inclusive.h"

struct Parabola_method {
	double operator()(double a, double b) {
		double x1 = a;
		double x;
		double x2;
		double x3 = b;
		double f1 = function(x1);
		double f2;
		double f3 = function(x3);
		double a_proto = a;
		double b_proto = b;
		double cur;
		double D = 1;
		int i = 1;
		while (true) {
			x2 = (a_proto + b_proto) / 2;
			f2 = function(x2);
			if (f2 <= f1 && f2 <= f2) {
				break;
			}
			else if (f2 <= f1 && f2 >= f3) {
				a_proto = x2;
			}
			else {
				b_proto = x2;
			}
		}

		while (D > eps) {
			double a_0 = f1;
			double a_1 = (f2 - f1) / (x2 - x1);
			double a_2 = ((f3 - f1) / (x3 - x1) - (f2 - f1) / (x2 - x1)) / (x3 - x2);
			x = (x1 + x2 - a_1 / a_2) / 2;
			if (i != 1) {
				D = abs(x - cur);
			}
			cur = x;

			double f = function(x);
			if (x1 < x && x < x2 && f >= f2) {
				a = x;
				b = x3;
				x1 = x;
				f1 = f;
			}
			else if (x1 < x && x < x2 && f < f2) {
				a = x1;
				b = x2;
				x3 = x2;
				f3 = f2;
				x2 = x;
				f2 = f;
			}
			else if (x2 < x && x < x3 && f >= f2) {
				a = x2;
				b = x3;
				x3 = x;
				f3 = f;
			}
			else {
				a = x1;
				b = x;
				x1 = x2;
				f1 = f2;
```

```
                x2 = x;
                f2 = f;
            }

            i++;
        }

        double f = function(x);
        return f;

    }
};
```

"Combined_Brent_method.h":

```
#pragma once

#include "All_inclusive.h"

struct Combined_Brent_method {
	double operator()(double a, double b, double eps) {

		double cnt = 0;

		double t = (3 - sqrt(5)) / 2;
		double x, w, v, u;
		x = w = v = a + t * (b - a);
		double d_cur, d_prev;
		d_cur = d_prev = b - a;
		double g;
		double f = function(x);
		cnt++;
		double fw = f, fv = f;

		while (true) {

			if (std::max(x - a, b - x) < eps) {
				f = function(x);
				cnt++;
				std::cout << cnt;
				return f;
			}

			g = d_prev / 2;
			d_prev = d_cur;

			double a_1 = (fw - f) / (w - x);
			double a_2 = ((fv - f) / (v - x) - a_1) / (v - x);
			u = (w + x - a_1 / a_2) / 2;

			if ((w == x) || (x == v) || (w == v) || !(u >= a && u <= b) ||
(std::abs(u - x) >= g / 2)) {
				if (x < (a + b) / 2) {
					u = x + (1 - t) * (b - x);
					d_prev = b - x;
				}
				else {
					u = a + (1 - t) * (x - a);
					d_prev = x - a;
				}
			}
			d_cur = std::abs(u - x);
			double fu = function(u);
			cnt += 2;
			if (fu > f) {
				if (u < x)
					a = u;
				else
					b = u;
				if (fu <= fw || w == x) {
					v = w;
					w = u;
					fv = fw;
```

```
                    fw = fu;
                }
                else {
                    if (fu <= fv || v == x || v == w) {
                        v = u;
                        fv = fu;
                    }
                }
            }
            else {
                if (u < x)
                    b = x;
                else
                    a = x;
                v = w;
                w = x;
                x = u;
                fv = fw;
                fw = f;
                f = fu;
            }
        }
    }

};
```

“MethOpt_lab1.cpp”:

```
#include "Dychotomy_method.h"
#include "Golden_ratio_method.h"
#include "Fibonacci_method.h"
#include "Parabola_method.h"
#include "Combined_Brent_method.h"

int main()
{
    setlocale(LC_ALL, "Rus");

    double e = extr(a, b);
    Dychotomy_method dm;
    Golden_ratio_method gdm;
    Fibonacci_method fm;
    Parabola_method pm;
    Combined_Brent_method cbm;

    std::cout.precision(4);
    std::cout << "y_min = " << e << std::endl << std::endl;

    std::cout << "Метод дихотомии:" << std::endl;
    double d = dm(a, b);
    std::cout << std::endl;

    std::cout << "Метод золотого сечения:" << std::endl;
    double g = gdm(a, b);
    std::cout << std::endl;

    std::cout << "Метод Фибоначчи:" << std::endl;
    double f = fm(a, b);
    std::cout << std::endl;

    std::cout << "Метод аппроксимирующей параболы:" << std::endl;
    double p = pm(a, b);
    std::cout << std::endl;

    std::cout << "Комбинированный метод Брента:" << std::endl;
    double c = cbm(a, b);
    std::cout << std::endl;

    std::cout << "Погрешности:" << std::endl;
    std::cout << std::abs(e - d) << std::endl;
    std::cout << std::abs(e - g) << std::endl;
    std::cout << std::abs(e - f) << std::endl;
    std::cout << std::abs(e - p) << std::endl;
    std::cout << std::abs(e - c) << std::endl;
```

```
}
```