Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования «Национальный исследовательский университет

«Московский институт электронной техники»

Институт микроприборов и систем управления имени Л.Н. Преснухина

Лабораторная работа №1

по дисциплине «Методы оптимизации»

Название: «Методы минимизации функций одной переменной»

Вариант №2

Дата выполнения: 04.10.2025

|  |  |  |
| --- | --- | --- |
|  |  | |
|  | | Выполнил: студент группы ИВТ-11М | |
|  | | Марков Владимир Иванович | |
|  | |  | |
|  |  | |

Москва 2025

Лабораторное задание:

1. Написать в среде MATLAB функции, реализующие метод перебора, метод поразрядного поиска, метод дихотомии, метод золотого сечения, метод парабол, метод средней точки, метод хорд и метод Ньютона.
2. Выбрать тестовую функцию в соответствии с вариантом
3. Построить график функции. Построить график числа вычислений функции (или её производной) от заданного значения точности. Сравнить методы, объяснить полученные результаты.
4. Сравнить результаты п.3 в методе хорд и методе Ньютона при использовании конечно-разностных производных.
5. С помощью метода Ньютона, метода Ньютона-Рафсона и метода Марквардта, используя аналитические производные и их численные аппроксимации решить задачу минимизации функции:

Сравнить результаты работы методов.

1. Составить программу нахождения глобального минимума многомодальных функций методом перебора и методом ломаных. Проверить её работоспособность на примере следующих функций и построить их графики:
3. Сдать лабораторную работу преподавателю.
4. Составление функций минимизации

Листинги функций минимизации на языке MATLAB приведены в Приложении А.

1. Выбор тестовой функции

Вариант 2

Функция:

1. Минимизация тестовой функции

На рисунке 1 представлен график функции согласно варианту. На рисунках 2 и 3 представлены скорости работы алгоритмов при разной требуемой точности вычисления (количество вычислений функции или её производных от точности).

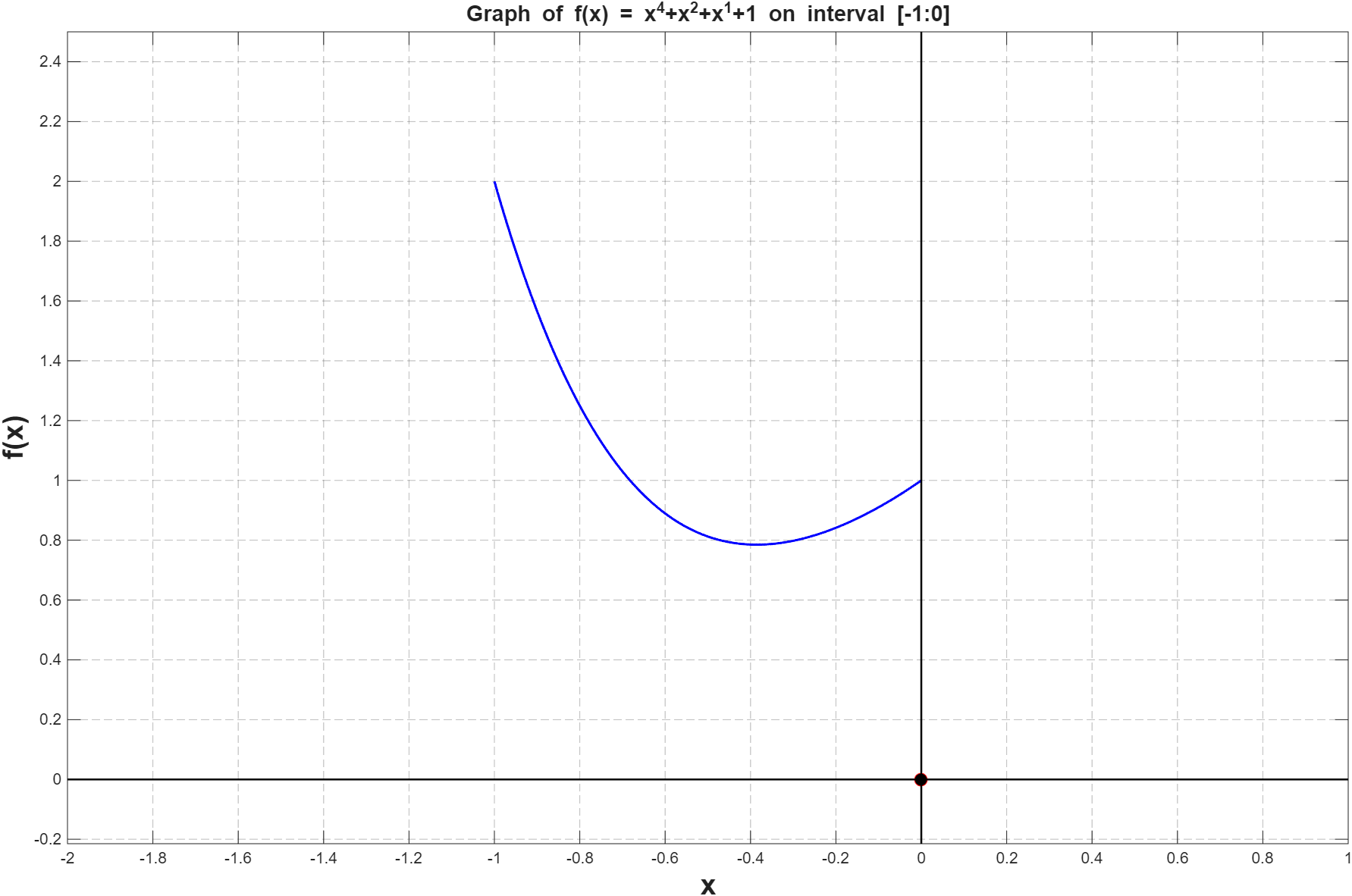


Рисунок 1 – График функции

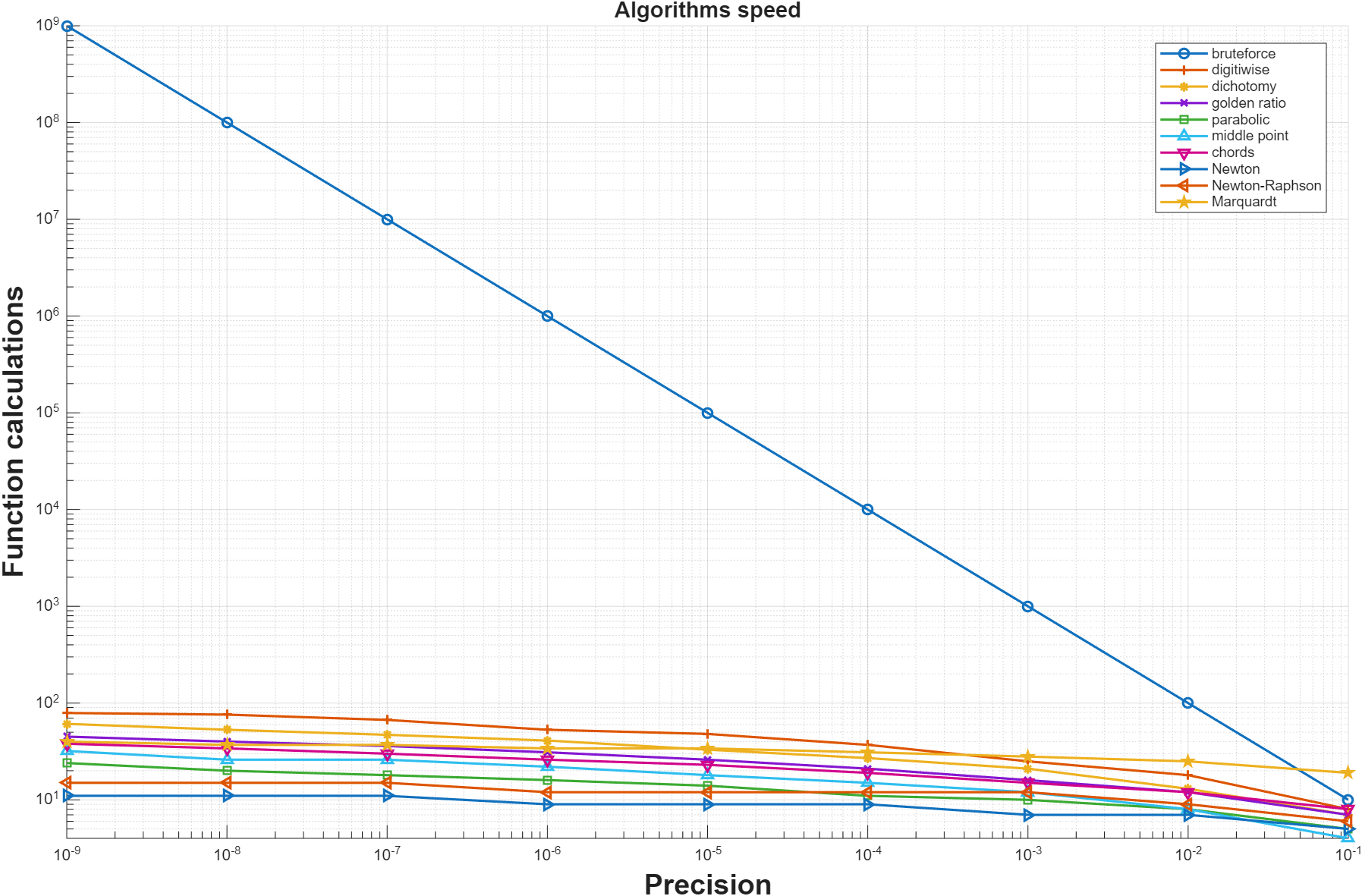


Рисунок 2 – График зависимости количества вычислений функции или производной от точности результата

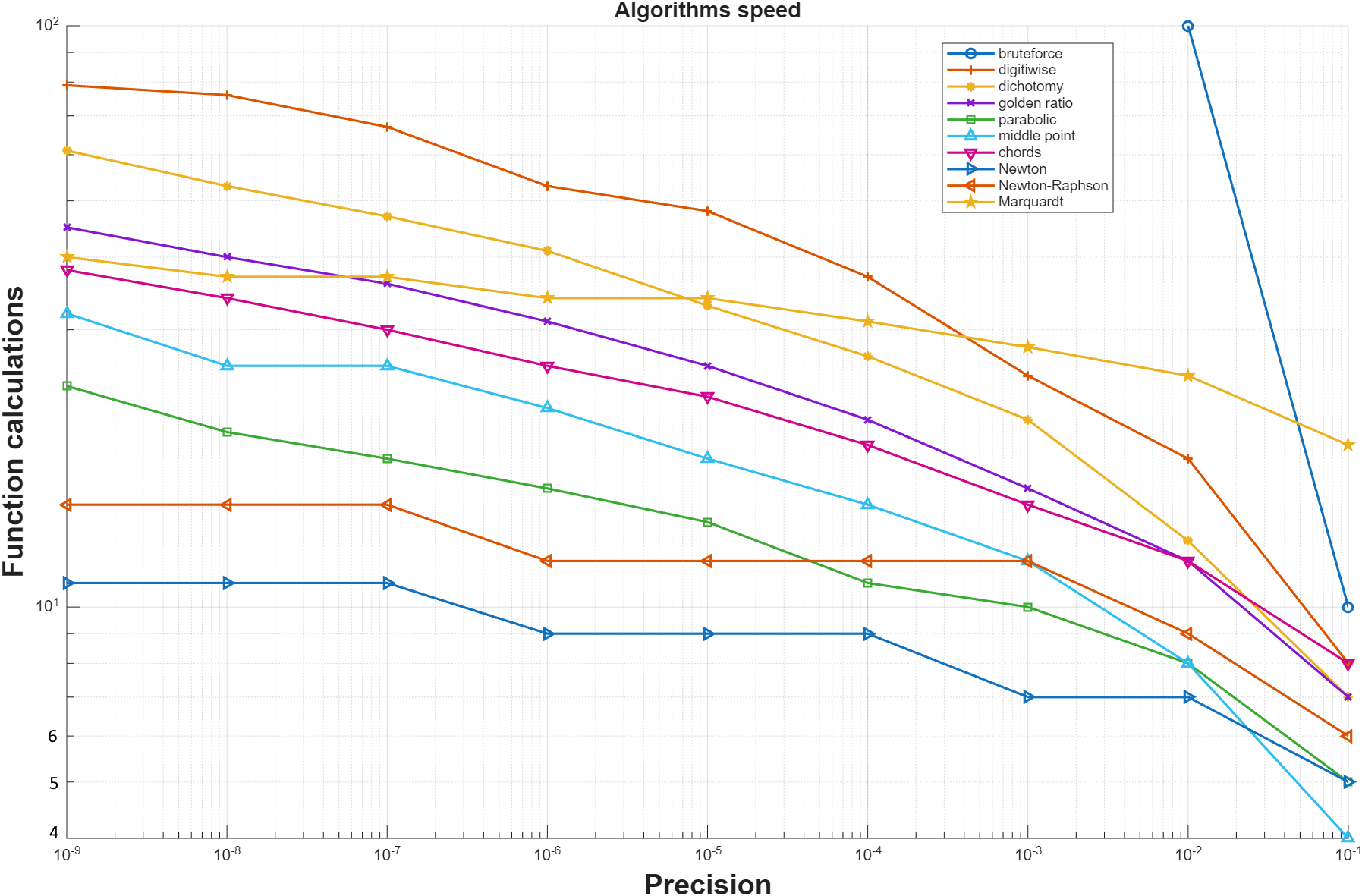


Рисунок 3 – График зависимости количества вычислений функции или производной от точности результата (приближено)

Исходя из результатов можно сделать вывод о том, что метод перебора является самым затратным из представленных. Метод Ньютона же, напротив, показывает очень высокую точность за сравнительно малое количество вычислений функции и её производных. Наилучшим прямым методом в данном случае можно назвать метод парабол. Также для сравнения на график добавлены результаты для метода Ньютона-Рафсона и метода Марквардта. Примечательно, что оба метода требуют больше вычислений, чем обычный метод Ньютона.

1. Разностные аналоги производных

На рисунке 4 представлены результаты анализа скорости работы функций, использующих производные функции, вычисляемые аналитически.

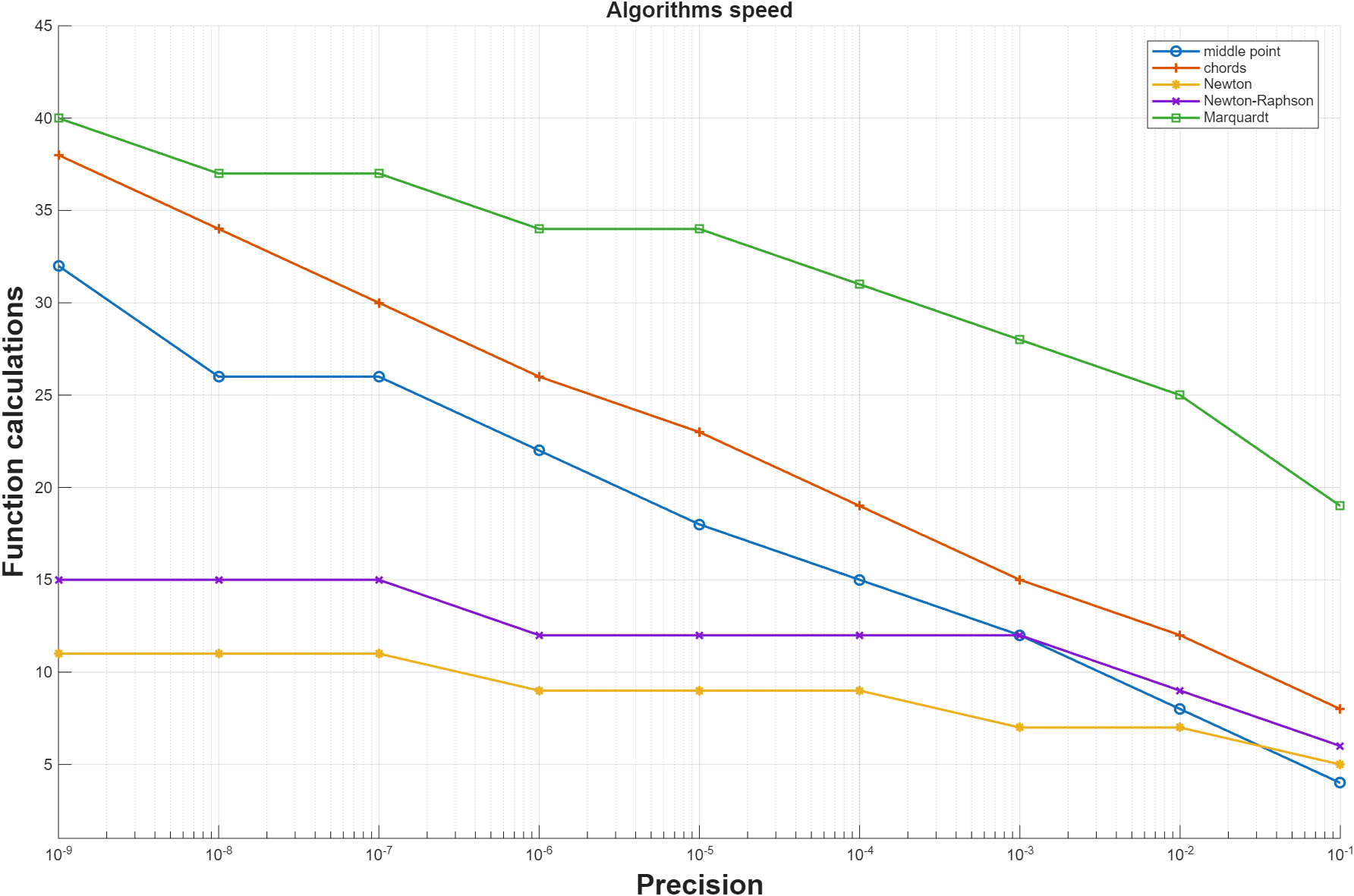


Рисунок 4 – Производительность алгоритмов при использовании аналитических производных

На рисунке 5 представлены результаты анализа скорости работы функций, использующих производные функции, вычисляемые левым разностным аналогом.

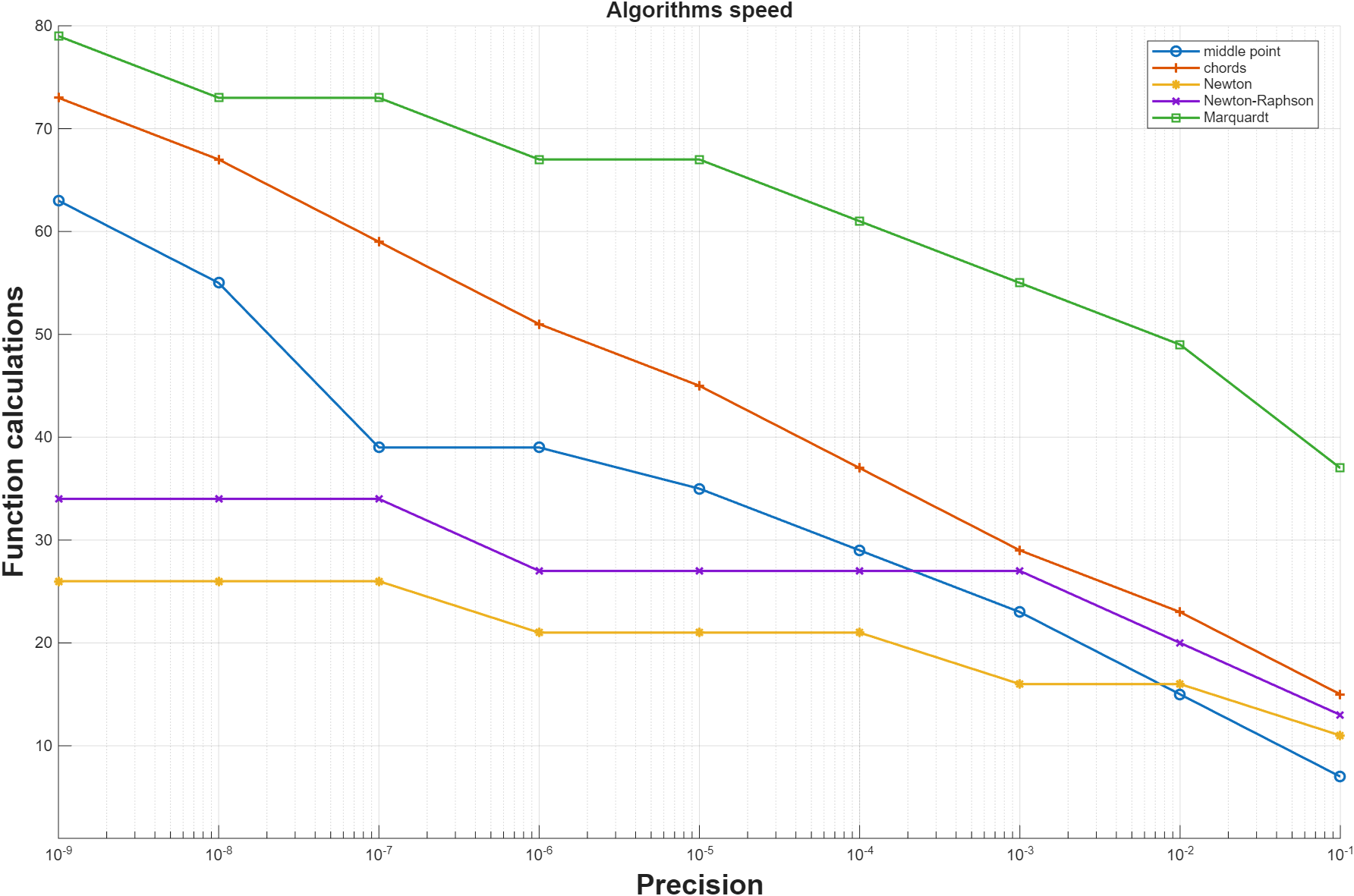


Рисунок 5 – Производительность алгоритмов при использовании левой разности для производных

На рисунке 6 представлены результаты анализа скорости работы функций, использующих производные функции, вычисляемые правым разностным аналогом.

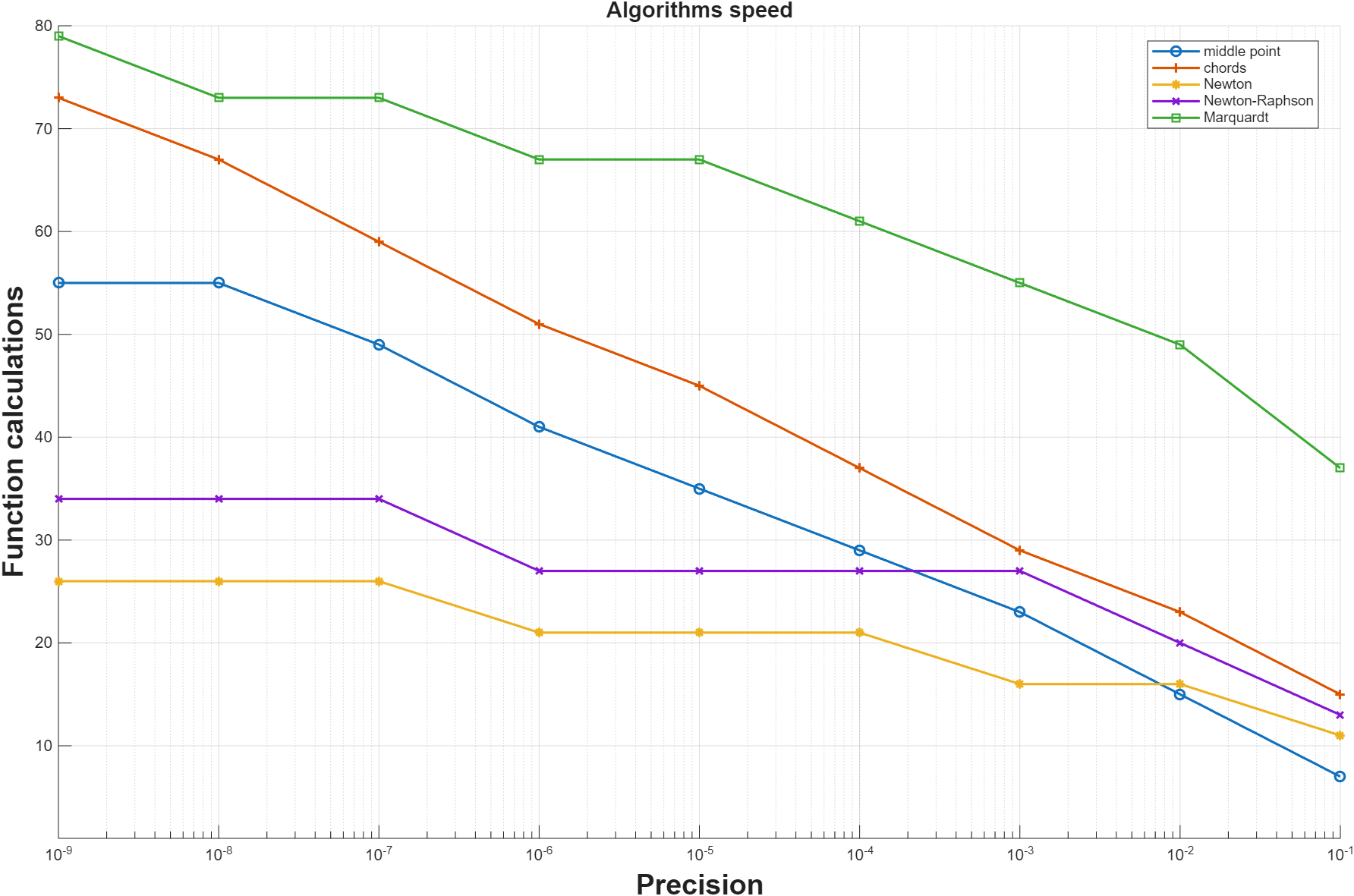


Рисунок 6 – Производительность алгоритмов при использовании правой разности для производных

На рисунке 7 представлены результаты анализа скорости работы функций, использующих производные функции, вычисляемые центральным разностным аналогом.

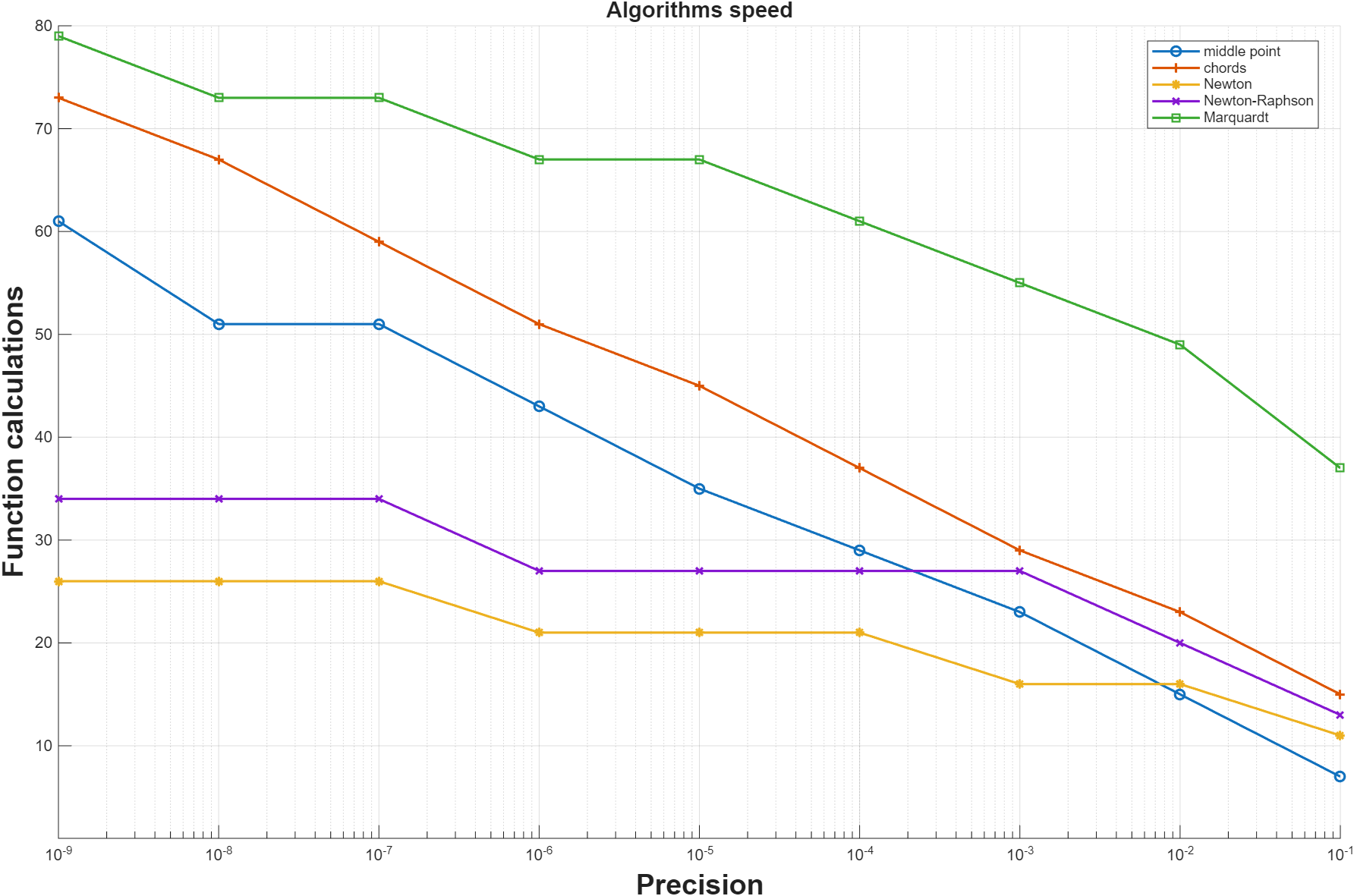


Рисунок 7 – Производительность алгоритмов при использовании центральной разности для производных

Применение разностных аналогов производных увеличивает количество вычислений функции, необходимых для достижения заданной точности. При этом тип конечной разности (левая, правая, центральная) для данного примера не влияет на скорость работы методов хорд, Ньютона, Ньютона-Рафсона и Марквардта. Метод средней точки же показал наилучший результат при использовании правой разности.

1. Изучение сходимости метода Ньютона и модификаций

На рисунке 8 представлен график функции на интервале [-4;4]

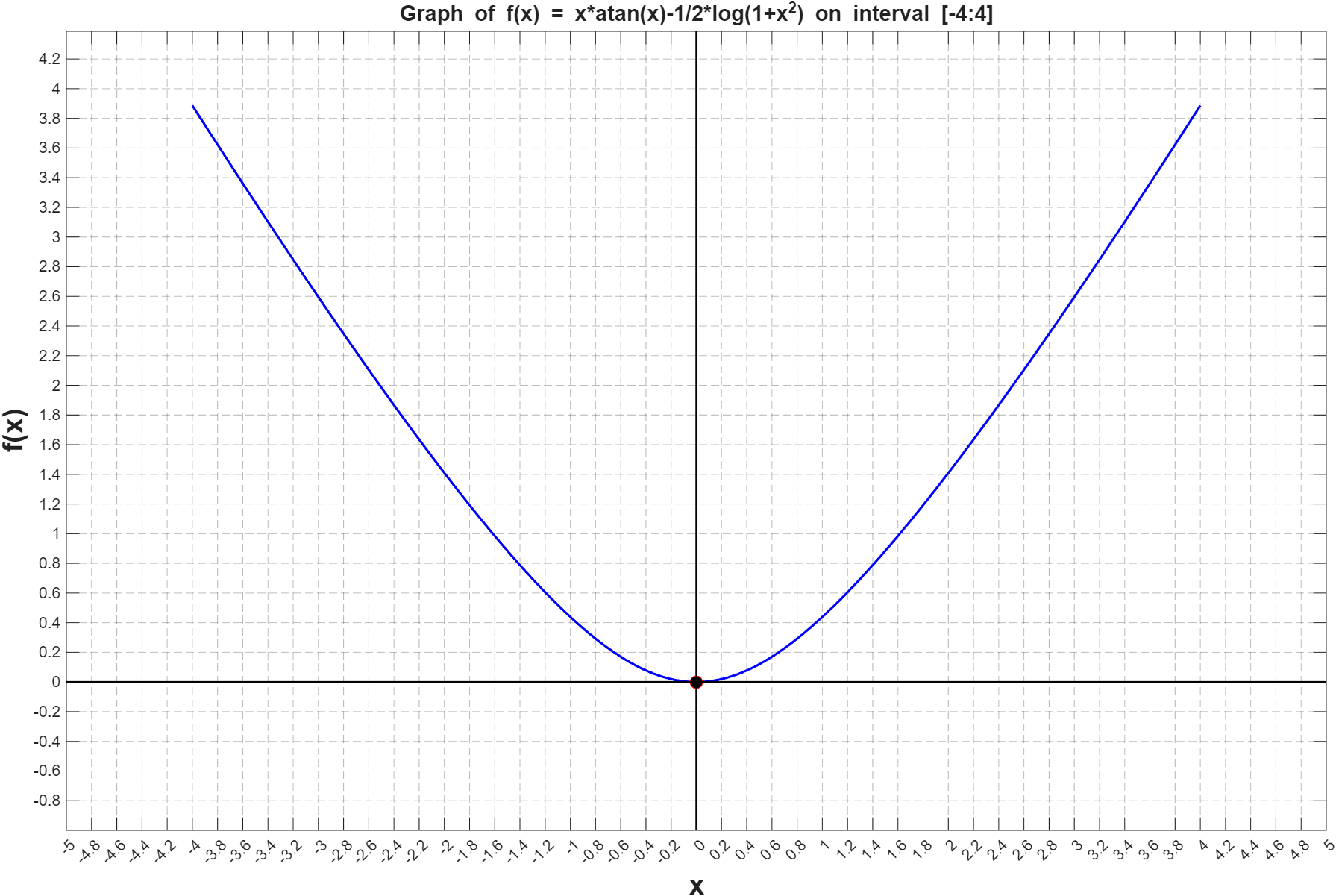


Рисунок 8 – График функции для изучения сходимости метода Ньютона и модификаций

В Таблице 1 приведены результаты изучения сходимости методов Ньютона, Ньютона-Рафсона и Марквардта. Границы вычислялись с точностью 0.001. Метод Ньютона показывает самый узкий интервал сходимости, метод Марквардта – самый широкий. Примечательно, что способ вычисления производной не влияет на метод Ньютона, а применение разностных аналогов ширины интервалов сходимости у методов Ньютона-Рафсона и Марквардта возрастает.

Таблица 1 – Результаты изучения сходимости

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Ньютона | | Ньютона-Рафсона | | Марквардта | |
| начало | конец | начало | конец | начало | конец |
| Аналитич | -1,3916 | 1,3916 | -3,3692 | 3,3692 | -8,8653 | 8,8653 |
| Левая | -1,3916 | 1,3916 | -3,3986 | 3,3889 | -10,9863 | 10,9863 |
| Правая | -1,3916 | 1,3916 | -3,3889 | 3,3986 | -10,9863 | 10,9863 |
| Центральная | -1,3916 | 1,3916 | -3,3889 | 3,3889 | -10,9863 | 10,9863 |

1. Минимизация многомодальных функций

На рисунках 9 и 10 представлены графики функций

и

соответственно

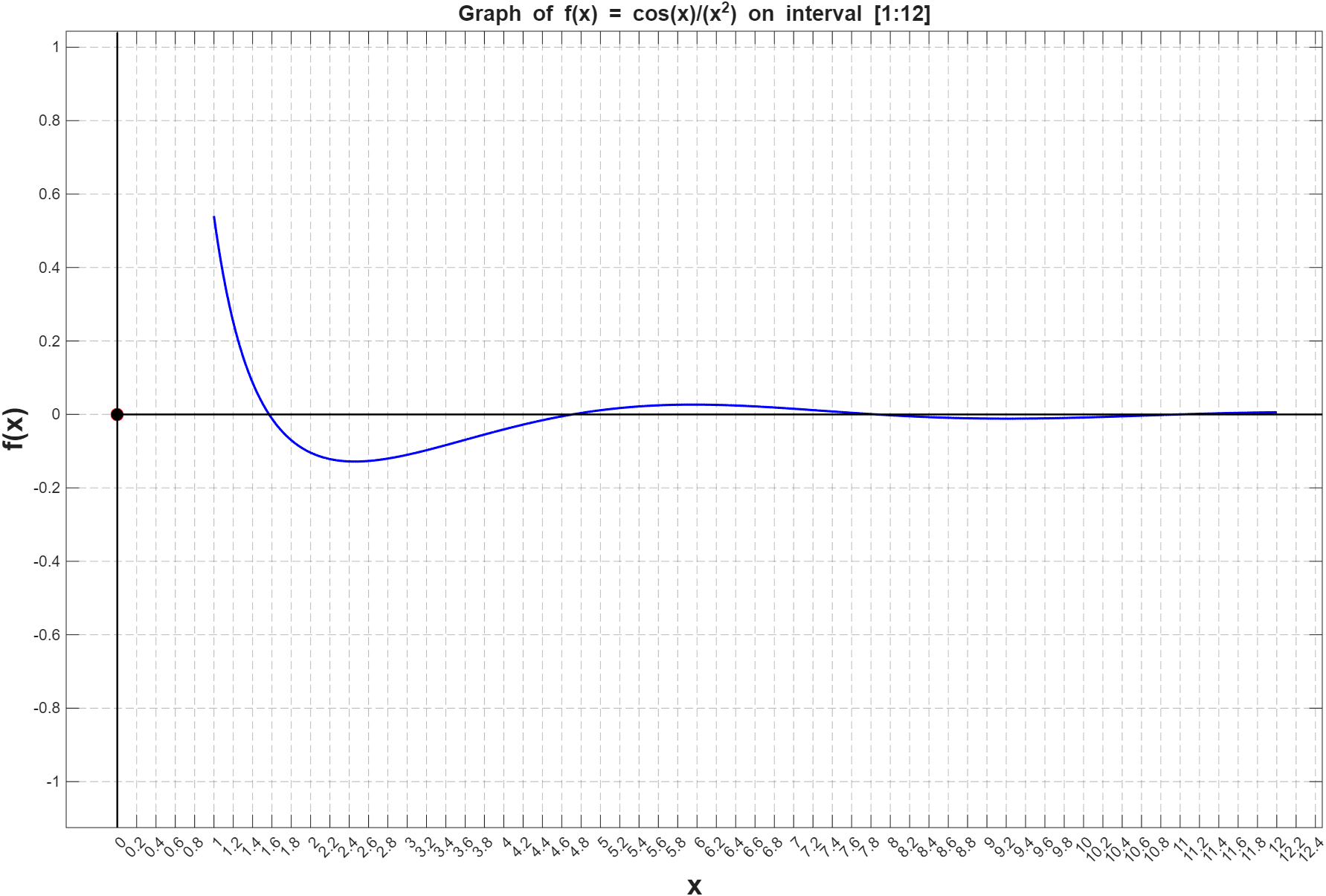


Рисунок 9 – График первой мультимодальной функции

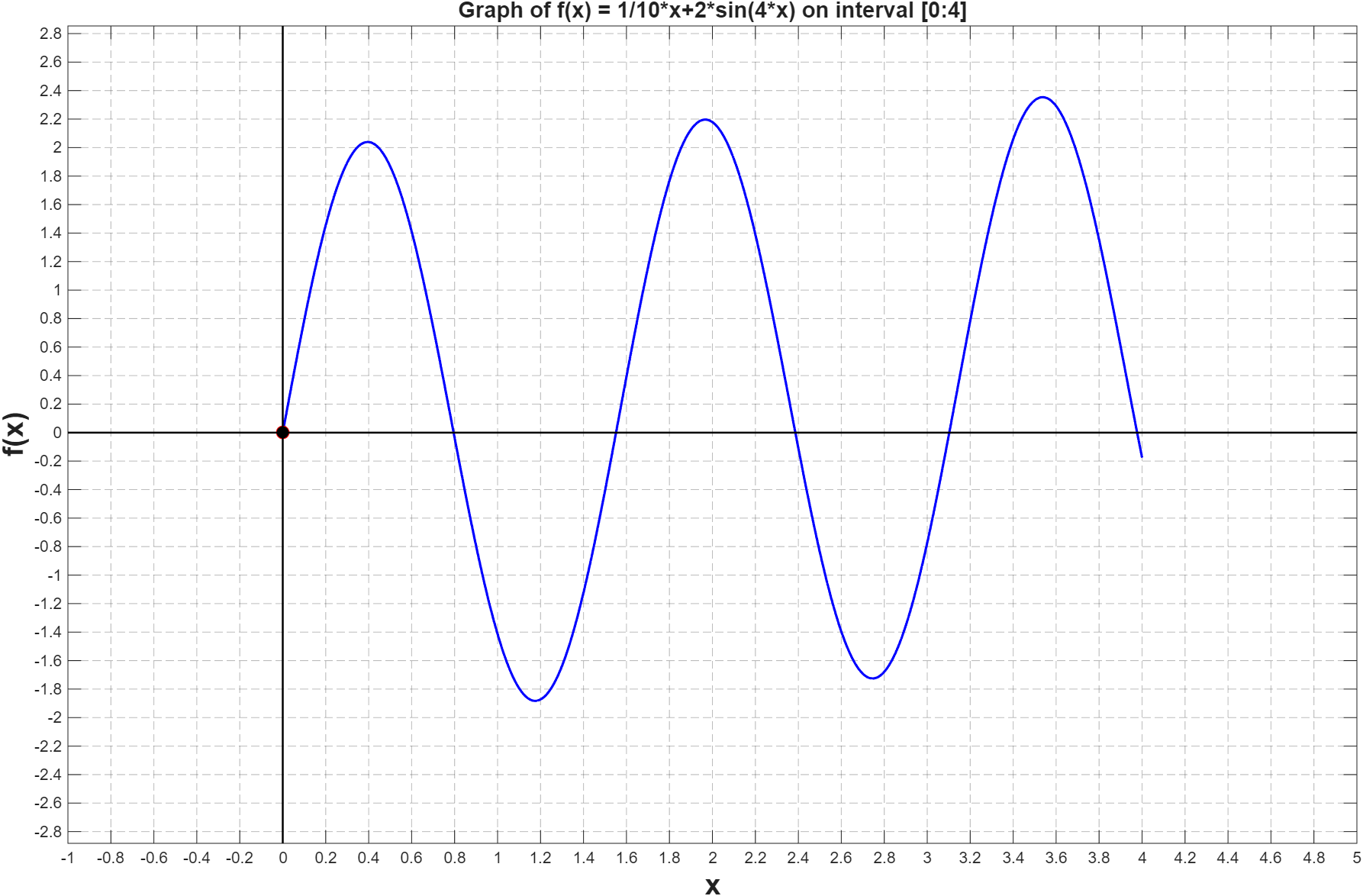


Рисунок 10 – График второй мультимодальной функции

В Таблице 2 приведено количество вычислений функции для получения заданной точности нахождения глобального минимума. Для минимизации были получены константы Липшица:

Таблица 2 – Сравнение скоростей работы разных методов минимизации на двух многомодальных функциях

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № ф-и | Точность | 10-1 | 10-2 | 10-3 | 10-4 | 10-5 | 10-6 | 10-7 |
| 1 | Перебор | 106 | 1058 | 10572 | 105715 | 1057142 | 10571416 | 105714158 |
| Ломаные | 80 | 224 | 658 | 1698 | 4662 | 18402 | 51926 |
| 2 | Перебор | 162 | 1620 | 16200 | 162000 | 1620000 | 16200000 | 162000000 |
| Ломаные | 29 | 68 | 164 | 606 | 1692 | 4758 | 18988 |

Выводы

Мы изучили некоторые основные методы минимизации унимодальных функций, а именно – метод перебора, метод поразрядного поиска, метод дихотомии, метод золотого сечения, метод парабол, метод средней точки, метод хорд, метод Ньютона и его модификации – метод Ньютона-Рафсона и метод Марквардта. Был проведён сравнительный анализ этих методов на тестовой функции, где метод перебора показал наименьшую скорость работы. Также были изучены интервалы сходимости методов Ньютона, Ньютона-Рафсона и Марквардта. Последний обладает наиболее широким интервалом сходимости для данной функции. Проведено сравнение результатов работы методов при вычислении производных аналитически и разностными методами. Были изучены методы минимизации многомодальных функций, а именно – метод перебора и метод ломаных. Метод ломаных при высокой требуемой точности на порядки выигрывает по скорости у метода перебора.

**Приложение А**

Метод перебора:

function [xmin, fmin, func\_calculations] = bruteforce(f, start, stop, precision)

n = ceil((stop - start)/precision);

arg = linspace(start, stop, n);

y = f(arg);

func\_calculations = n;

[fmin, ind] = min(y);

xmin = arg(ind);

end

Метод поразрядного поиска:

function [xmin, fmin, func\_calculations] = digitwise(f, start, stop, precision)

func\_calculations = 0;

delta = 1;

x = start;

direction = 1;

yprev = 0;

while delta > precision

delta = delta / 4;

yprev = f(start);

func\_calculations = func\_calculations + 1;

x = x + delta\*direction;

y = f(x);

func\_calculations = func\_calculations + 1;

while y < yprev

yprev = y;

xprev = x;

x = x + delta\*direction;

y = f(x);

func\_calculations = func\_calculations + 1;

end

direction = direction \* -1;

end

fmin = yprev;

xmin = xprev;

end

Метод дихотомии:

function [xmin, fmin, func\_calculations] = dichotomy(f, start, stop, target\_precision)

func\_calculations = 0;

d = 0.2\*target\_precision;

achieved\_precision = (stop - start) / 2;

while achieved\_precision > target\_precision

x1 = (start + stop - d) / 2;

x2 = (start + stop + d) / 2;

y1 = f(x1);

y2 = f(x2);

func\_calculations = func\_calculations + 2;

if y1 <= y2

stop = x2;

else

start = x1;

end

achieved\_precision = (stop - start) / 2;

end

xmin = (stop + start) / 2;

fmin = f(xmin);

func\_calculations = func\_calculations + 1;

end

Метод золотого сечения:

function [xmin, fmin, func\_calculations] = goldenratio(f, start, stop, target\_precision)

tau = (sqrt(5) - 1) / 2;

x1 = start + (1 - tau) \* (stop - start);

x2 = start + tau \* (stop - start);

y1 = f(x1);

y2 = f(x2);

func\_calculations = 2;

n = 0;

achieved\_precision = (stop - start) / 2 \* tau^n;

while achieved\_precision > target\_precision

if y1 <= y2

stop = x2;

x2 = x1;

x1 = start + stop - x2;

y2 = y1;

y1 = f(x1);

func\_calculations = func\_calculations + 1;

else

start = x1;

x1 = x2;

x2 = start + stop - x1;

y1 = y2;

y2 = f(x2);

func\_calculations = func\_calculations + 1;

end

n = n + 1;

achieved\_precision = achieved\_precision \* tau;

end

xmin = (x1 + x2) / 2;

fmin = f(xmin);

func\_calculations = func\_calculations + 1;

end

Метод парабол:

function [xmin, fmin, func\_calculations] = parabolic(f, start, stop, target\_precision)

x1 = start;

x2 = (start + stop) / 2;

x3 = stop;

f1 = f(x1);

f2 = f(x2);

f3 = f(x3);

func\_calculations = 3;

while ~((f1 >= f2) && (f2 <= f3))

if f1 > f3

x2 = (x2 + x3) / 2;

else

x2 = (x1 + x2) / 2;

end

f2 = f(x2);

func\_calculations = func\_calculations + 1;

end

xmin = x2;

delta = target\_precision + 1;

while delta > target\_precision

a0 = f1;

a1 = (f2 - f1)/(x2 - x1);

a2 = 1/(x3 - x2) \* (((f3-f1)/(x3-x1))-((f2-f1)/(x2-x1)));

xminprev = xmin;

xmin = 1/2\*(x1 + x2 - a1/a2);

fmin = f(xmin);

func\_calculations = func\_calculations + 1;

if xmin < x2

if fmin >= f2

x1 = xmin;

f1 = fmin;

else

x1 = x2;

f1 = f2;

x2 = xmin;

f2 = fmin;

end

else

if fmin >= f2

x3 = xmin;

f3 = fmin;

else

x1 = x2;

f1 = f2;

x2 = xmin;

f2 = fmin;

end

end

delta = abs(xminprev - xmin);

end

end

Метод средней точки:

function [xmin, fmin, func\_calculations] = middlepoint(func, start, stop, target\_precision)

global FIRST\_DERIVATIVE\_COST;

func\_calculations = 0;

xmin = (stop + start) / 2;

dfmin = first\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST;

while abs(dfmin) > target\_precision

if dfmin > 0

stop = xmin;

else

start = xmin;

end

xmin = (start + stop) / 2;

dfmin = first\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST;

end

fmin = func(xmin);

func\_calculations = func\_calculations + 1;

end

Метод хорд:

function [xmin, fmin, func\_calculations] = chord(func, start, stop, target\_precision)

global FIRST\_DERIVATIVE\_COST;

func\_calculations = 0;

dfa = first\_derivative(func, start);

dfb = first\_derivative(func, stop);

xmin = start - (dfa / (dfa - dfb)) \* (start - stop);

dfmin = first\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST \* 3;

while abs(dfmin) > target\_precision

if dfmin > 0

stop = xmin;

dfb = dfmin;

else

start = xmin;

dfa = dfmin;

end

xmin = start - (dfa / (dfa - dfb)) \* (start - stop);

dfmin = first\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST;

end

fmin = func(xmin);

func\_calculations = func\_calculations + 1;

end

Метод Ньютона:

function [xmin, fmin, func\_calculations] = Newton(func, start, target\_precision)

global FIRST\_DERIVATIVE\_COST;

global SECOND\_DERIVATIVE\_COST;

func\_calculations = 0;

xmin = start;

dfmin = first\_derivative(func, xmin);

ddfmin = second\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST + SECOND\_DERIVATIVE\_COST;

step = abs(dfmin/ddfmin);

while abs(dfmin) > target\_precision

xmin = xmin - dfmin/ddfmin;

prevstep = step;

step = abs(dfmin/ddfmin);

if (step - prevstep) > 0

xmin = Inf;

break

end

dfmin = first\_derivative(func, xmin);

ddfmin = second\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST + SECOND\_DERIVATIVE\_COST;

end

fmin = func(xmin);

func\_calculations = func\_calculations + 1;

end

Метод Ньютона-Рафсона:

function [xmin, fmin, func\_calculations] = Newton\_Raphson(func, start, target\_precision)

global FIRST\_DERIVATIVE\_COST;

global SECOND\_DERIVATIVE\_COST;

func\_calculations = 0;

xmin = start;

dfmin = first\_derivative(func, xmin);

ddfmin = second\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST + SECOND\_DERIVATIVE\_COST;

step = abs(dfmin/ddfmin);

while abs(dfmin) > target\_precision

xrude = xmin - dfmin/ddfmin;

dfxrude = first\_derivative(func, xrude);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST;

tau = dfmin^2 / (dfmin^2 + dfxrude^2);

xmin = xmin - dfmin/ddfmin \* tau;

prevstep = step;

step = abs(dfmin/ddfmin);

if (step - prevstep) > 0

xmin = Inf;

break

end

dfmin = first\_derivative(func, xmin);

ddfmin = second\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST + SECOND\_DERIVATIVE\_COST;

end

fmin = func(xmin);

func\_calculations = func\_calculations + 1;

end

Метод Марквардта:

function [xmin, fmin, func\_calculations] = Marquardt(func, start, target\_precision)

global FIRST\_DERIVATIVE\_COST;

global SECOND\_DERIVATIVE\_COST;

func\_calculations = 0;

xmin = start;

dfmin = first\_derivative(func, xmin);

ddfmin = second\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST + SECOND\_DERIVATIVE\_COST;

step = abs(dfmin/ddfmin);

mu = ddfmin \* 10;

fx = func(xmin);

func\_calculations = func\_calculations + 1;

while abs(dfmin) > target\_precision

xmin = xmin - dfmin/(ddfmin + mu);

fxnew = func(xmin);

func\_calculations = func\_calculations + 1;

if fx > fxnew

mu = mu / 2;

else

mu = mu \* 2;

end

fx = fxnew;

prevstep = step;

step = abs(dfmin/ddfmin);

if (step - prevstep) > 0

xmin = Inf;

break

end

dfmin = first\_derivative(func, xmin);

ddfmin = second\_derivative(func, xmin);

func\_calculations = func\_calculations + FIRST\_DERIVATIVE\_COST + SECOND\_DERIVATIVE\_COST;

end

fmin = func(xmin);

func\_calculations = func\_calculations + 1;

end

Метод ломаных:

function [xmin, fmin, func\_calculations] = Polyline(func, start, stop, target\_precision)

L = abs(double(get\_lipschitz\_const(func, start, stop, target\_precision)));

f\_a = func(start);

f\_b = func(stop);

func\_calculations = 2;

x = (f\_a - f\_b + L\*(start + stop)) / (2\*L);

y = (f\_a + f\_b + L\*(start - stop)) / 2;

x\_p\_arr = [[x, y]];

delta\_y = target\_precision + 1;

while delta\_y > target\_precision

[p, i] = min(x\_p\_arr(:,2));

x = x\_p\_arr(i,1);

y = func(x);

func\_calculations = func\_calculations + 1;

delta\_x = (y - p) / (2\*L);

delta\_y = delta\_x \* (2\*L);

x1 = x - delta\_x;

x2 = x + delta\_x;

p = (y + p) / 2;

x\_p\_arr = [x\_p\_arr(1:i - 1,:); [x1, p]; [x2, p]; x\_p\_arr(i + 1:end,:)];

end

xmin = x;

fmin = y;

end