Национальный исследовательский университет
информационных технологий, механики и оптики

Кафедра Компьютерных технологий и программирования

Лабораторная работа №1

Выполнили:
Дроботов И.В. М 3232
Бандурин В.А. М 3232
Ларский Н.А. М 3232

Преподаватель:
Казанков В.К.

Санкт-Петербург
2023

Содержание

1 Цель

Изучение методов оптимизации, включая градиентный спуск с постоянным шагом и метод одномерного поиска (например, метод дихотомии), а также анализ и сравнение их эффективности и сходимости на примере квадратичных функциях, необходимо выполнить следующие задачи.

Постановка задачи

1. Реализуйте градиентный спуск с постоянным шагом (learning rate).
2. Реализуйте метод одномерного поиска (метод дихотомии, метод Фибоначчи, метод золотого сечения) и градиентный спуск на его основе.
3. Проанализируйте траекторию градиентного спуска на примере квадратичных функций. Для этого придумайте две-три квадратичные функции от двух переменных, на которых работа методов будет отличаться.
4. Для каждой функции:
    4.1. исследуйте сходимость градиентного спуска с постоянным шагом, сравните полученные результаты для выбранных функций;
    4.2. сравните эффективность градиентного спуска с использованием одно- мерного поиска с точки зрения количества вычислений минимизируемой функции и ее градиентов;
    4.3. исследуйте работу методов в зависимости от выбора начальной точки;
    4.4. исследуйте влияние нормализации (scaling) на сходимость на примере масштабирования осей плохо обусловленной функции;
    4.5. в каждом случае нарисуйте графики с линиями уровня и траекториями методов;
5. Реализуйте генератор случайных квадратичных функций n переменных с числом обусловленности k.
6. Исследуйте зависимость числа итераций $T(n, k)$, необходимых градиентному спуску для сходимости в зависимости от размерности пространства $2 \leq n \leq 10^3$ и числа обусловленности оптимизируемой функции $1 \leq k \leq 10^3$.
7. Для получения более корректных результатов проведите множественный эксперимент и усредните полученные значения числа итераций.

2 Решение

1. Код градиентного спуска с постоянным шагом

```cpp
qreal MainWindow::grad(const func_t &f, const qreal x) noexcept
{
    return (f(x + EPSILON) - f(x)) / EPSILON;
}

qreal MainWindow::next_step(const func_t &f, qreal const x, qreal const lr) noexcept
{
    return x - lr * grad(f, x);
}

qreal MainWindow::func_sum(const v<func_t> & funcs, v<qreal> const& pos) noexcept
{
    qreal result{0};
    for (int i = 0; i < pos.size(); ++i) {
        result += funcs[i](pos[i]);
    }
    return result;
}

v<v<qreal>> MainWindow::gradient_decent(const v<func_t> & funcs, v<qreal>&& start, const qreal lr, int max_step) noexcept
{
    v<v<qreal>> way;
    int i;
    way.clear();
    way.push_back(start);

    int const metric = start.size();
    v<qreal> cur_pos = std::move(start);
    v<qreal> old_pos = cur_pos;

    for (i = 0; i < old_pos.size(); ++i) old_pos[i] -= EPSILON * 100;
    qDebug() << "============================";
    qDebug() << cur_pos;
    while (max_step-- > 0) {
        old_pos = cur_pos;

        for (i = 0; i < metric; ++i) {
            cur_pos[i] = next_step(funcs[i], cur_pos[i], lr);
        }
        if ((std::abs(func_sum(funcs, old_pos) - func_sum(funcs, cur_pos)) <= EPSILON)) {
            break;
        }
        qDebug() << cur_pos;
        way.push_back(cur_pos);
    }
    qDebug() << "============================";
    qDebug() << "Iretations: " << way.size() - 1;
    return way;
}
```

Описание кода

Итак, первая функция просто принимает одномерную квадратичную функцию (f) и считает для неё градиент в точке (x).

Вторая функция принимает одномерную квадратичную функцию (f), точку (x) и множитель градиета (lr). По данной информации даёт координаты новой точки, значение функции в которой меньше предыдущего. Другими словами это функция постоянного шага градиентного спуска.

Третья функция даёт из нескольких одномерных независимых квадратичных функций ($funcs$) n-мерную квадратичную функцию и высчитывает её значение в n-мерной точке (pos).

Четвёртая функция реализует градиентный спуск с постоянным шагом. Первый параметр ($funcs$) даёт набор одномерных независимых квадратичных функций, ($start$) - точка старта, (lr) - даёт в зависимости от ($next\_step$) либо множитель шага, либо максимальный шаг при оптимальном спуске.

2. Код градиентного спуска с оптимизацией через метод дихотомии сечения пополам.

```cpp
v<v<qreal> > MainWindow::gradient_decent_with_dihotomy(const v<func_t> & funcs, v<qreal>&& start, qreal const lr, int const max_step) noexcept
{
    v<qreal> a(start.size(), 0);
    v<qreal> b(start.size(), 0);
    qreal m;
    qreal grd;

    for (int i = 0; i < start.size(); ++i) {
        m = lr;
        grd = grad(funcs[i], start[i]);
        grd = grd / std::abs(grd);

        a[i] = start[i];
        while (grad(funcs[i], start[i] - m * grd) * grd >= 0) {
            m *= 2;
        }
        b[i] = start[i] - 2 * m * grd;

        if (a[i] > b[i]) std::swap(a[i], b[i]);
    }

    return dihotomy(funcs, std::move(a), std::move(b), std::move(start), max_step);
}

v<v<qreal>> MainWindow::dihotomy(const v<func_t> & funcs, v<qreal>&& a, v<qreal>&& b, v<qreal>&& start, int max_step) noexcept
{
    int i;
    v<v<qreal>> way = {std::move(start)};
    int const metric = a.size();
    v<qreal> x(metric, 0);
    v<bool> final(metric, false);
    qreal grd{0};
    bool contin = true;

    qDebug() << "============================";
    qDebug() << way[0];
    while (max_step-- > 0 && contin) {
        contin = false;

        for (i = 0; i < metric; ++i) {
            if (final[i]) {
                continue;
            }
            x[i] = (a[i] + b[i]) / 2;
            grd = grad(funcs[i], x[i]);
            if (std::abs(grd) < EPSILON) {
                final[i] = true;
            } else {
                if (grad(funcs[i], x[i]) > 0) {
                    b[i] = x[i];
                } else {
                    a[i] = x[i];
                }
                //final[i] = std::abs(a[i] - b[i]) < EPSILON;
            }
            contin |= !final[i];
        }
        qDebug() << x;
        way.push_back(x);
    }
    qDebug() << "============================";
    qDebug() << "Iretations: " << way.size() - 1;
    return way;
}
```

Описание кода

Выглядит жутковато, но после разбора станет гораздо легче.

Итак, первая функция представляет собой метод градиентного спуска с помощью оптимизации, код которой находится ниже. Первая часть функции нацелена на то, чтобы найти правильные параметры (a) и (b), которые уже используются самой оптимизацией. И как раз последняя строчка кода первой функции вызывает её с данными параметрами.

Вторая же функция просто вычисляет путь путём выслисления его для каждой из осей. $(contin)$ отвечает за продолжение вычисления пути, пока есть хотя бы одна ось, на которой потенциально может быть новый ход. Однако, те оси на которых нашли уже минимум, блокируют повторные вызовы, через $(final)$.

3. Для анализа возьмём эти три функции:

   3.1. $f(x,y) = x^2 + y^2$ стартовая точка (100, 200)

   3.2. $f(x,y) = 0.01x^2 - 3x + 4 + 0.05y^2 + 4y - 15$ стартовая точка (0, 10)

   3.3. $f(x,y) = 0.1x^2 - 3x + \frac{1}{3}y^2 + 5y + 6$ стартовая точка (50, -30)

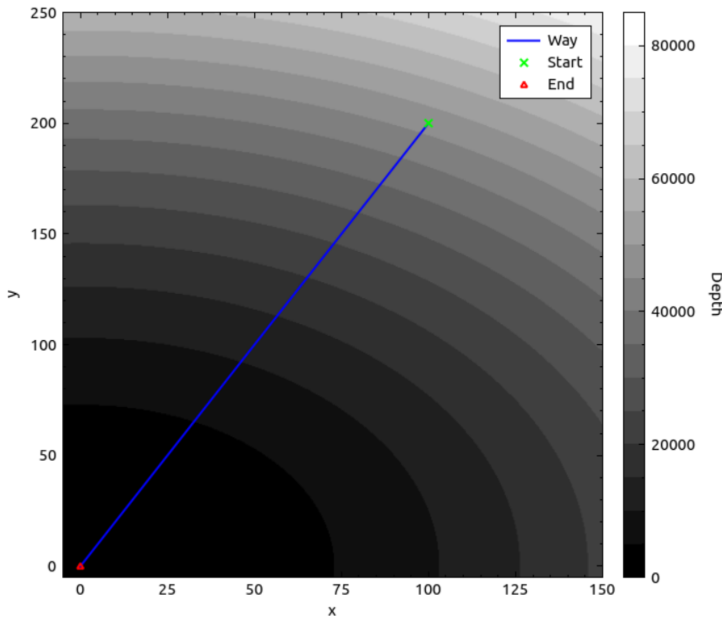
Графики

3.1. Постояный шаг

`Шаг:` 0.1

`Количество ходов:` 69.

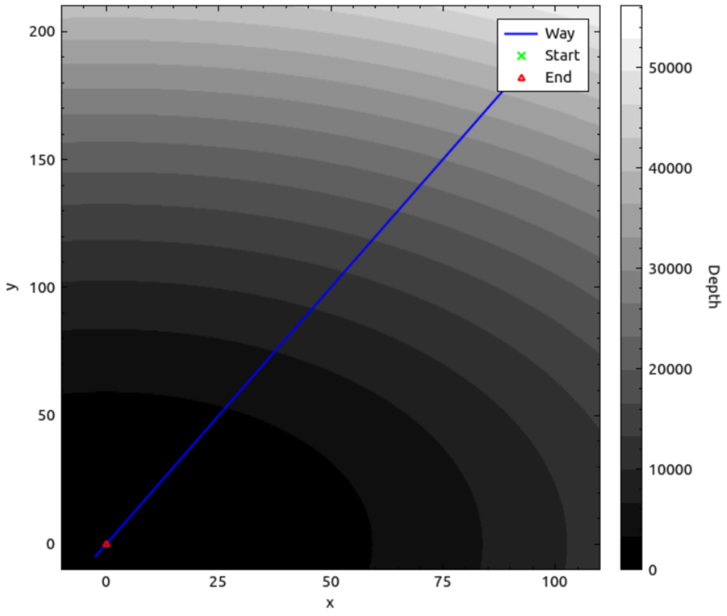
`Результат:` (2.05683e-05, 4.11374e-05)



Метод дихотомии

`Количество ходов:` 8.

`Результат:` (-5.55112e-15, -1.11022e-14)

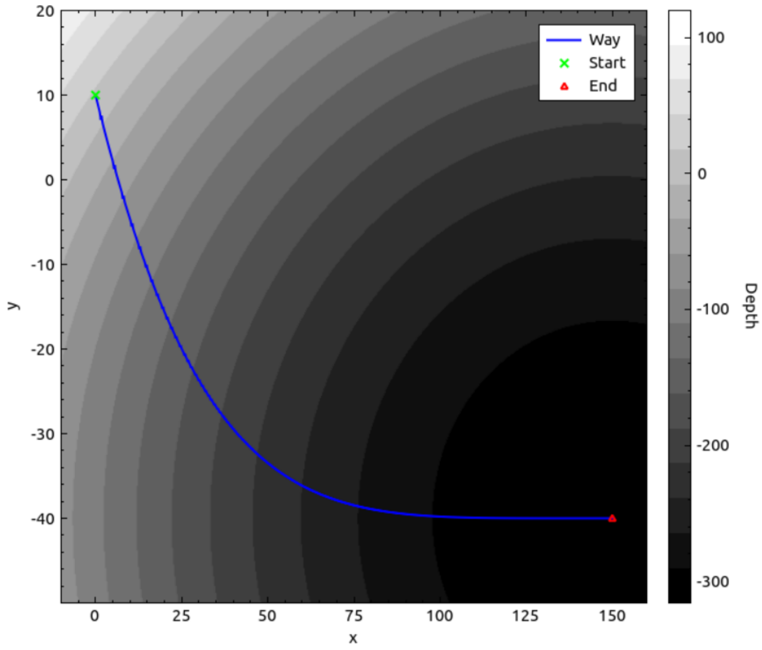


3.2. Постояный шаг

Шаг: 0.1

Количество ходов: 5144.

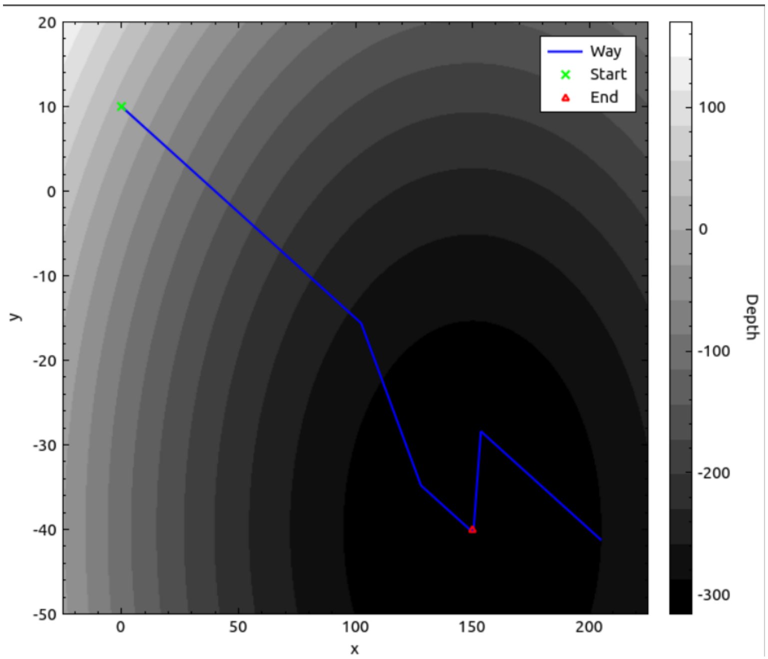
Результат: (149.994, -39.9999)



Метод дихотомии

Количество ходов: 10.

Результат: (150, -40)

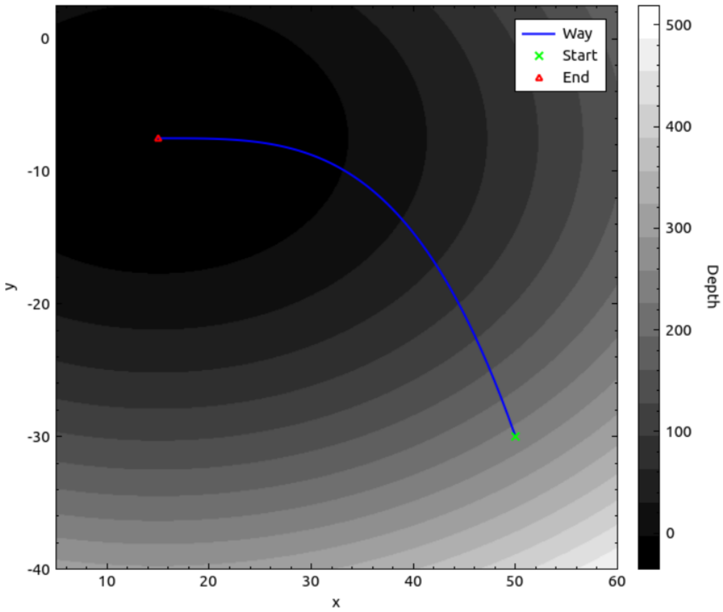


3.3. Постояный шаг

`Шаг:` 0.1

`Количество ходов:` 552.

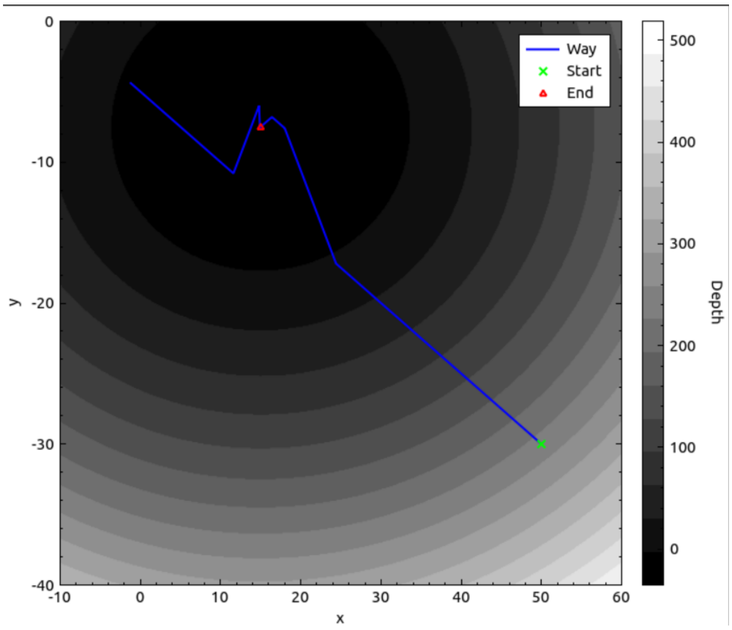
`Результат:` (15.0005, -7.50751)



Метод дихотомии

`Количество ходов:` 22.

`Результат:` (15, -7.50751)



4. 4.1. Проанализируем работы методов с постоянным шагом. В первой нункции

у нас дуги возрастают с большой скоростью относительно стартовой точки. Следовательно скорость спуска будет тоже быстрой, что мы и видим. Количество шагов в этом поиске 69, что по сравнению с другими функциями и запусками поиска довольно мало.

Если же мы посмотрим на запуск второй функции, то увидим, что дуги парабол возрастают медленнее тем самым спуск тоже оказался не маленьким. Тут нужно ещё учесть тот факт, что расстояние до точки экстремума, то есть ответа, тоже оказалось довольно больши. Поэтому количество шагов составило 5144.

У третьей же функции дуги восрастают не так быстро как в перво и не так медленно как во второй, а расстояние до ответа примерно такое же. В результате, получаем 552 шага, которое больше чем у первой функции и гораздо меньше, чем у второй. Но стоит учесть, что это довольно приблизительный анализ, так как запуски поиска и сами функции разные.

4.2. Теперь проанализируем результаты градиентного спуска с постоянным шагом. Тут реультаты немного не в таком же отношении как при использовании первого метода. Объясняется это спецификой выбора точек (a) и (b). Когда нам дают стартовую точку в одномерной квадратичной функции, то нем нужно найти противоположнуюю точку такую, чтобы произведение производных в этих точках было разным, для этого я просто беру начальный шаг и увеличиваю его в два раза пока не попадаю на противоположную сторону. В результате у нас могут получиться не всегда оптимальные значения границ.

4.3. Для этого пункта немного изменим наши анализируемые функции, чтобы число обучловленности было большим и нам было бы удобнее показывать различия итераций при нормализации в будущем. Теперь наши функции такие:

   i. $f(x, y) = x^2 + y^2$ стартовые точки (1, 1), (100, 200)
   ii. $f(x, y) = 0.01x^2 - 3x + 4 + 0.05y^2 + 4y - 15$ стартовая точка (250, 50), (-1000, -1000)
   iii. $f(x, y) = 0.5x^2 - 3x + 5y^2 + 5y + 6$ стартовая точка (-1, 1), (50, -40)

Графики

i. Первая функция

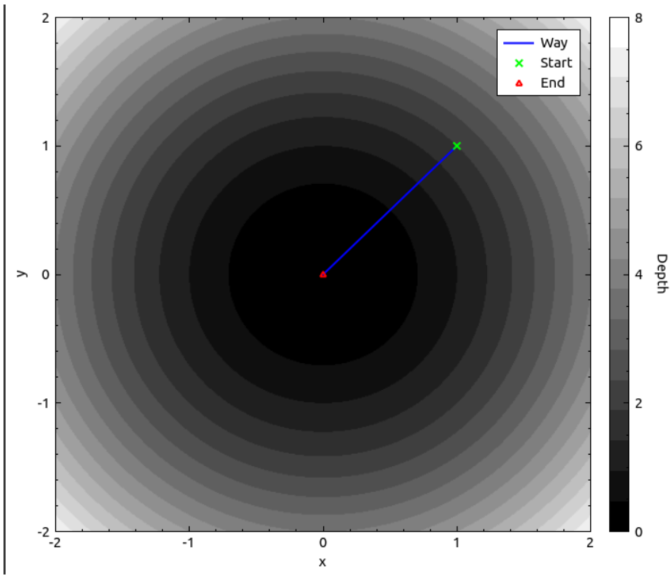
A. Первая стартовая точка

Постояный шаг

Шаг: 0.1

Количество ходов: 46.

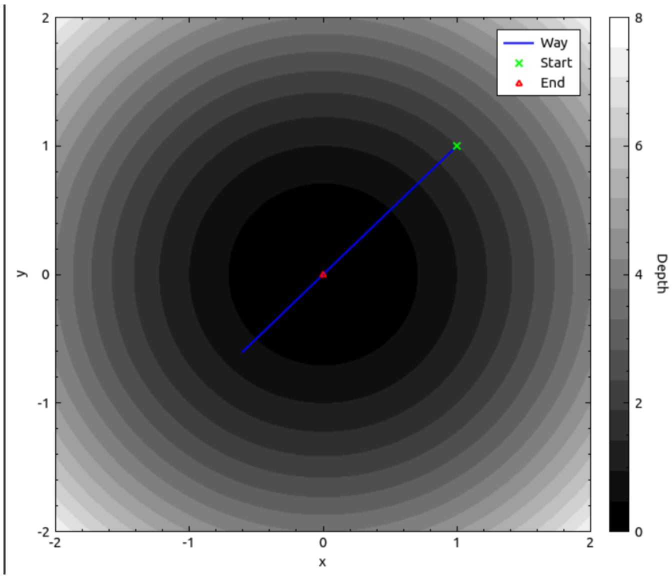
Результат: (3.48444e-05, 3.48444e-05)



Дихотомия

Количество ходов: 4.

Результат: (-5.55112e-17, -5.55112e-17)



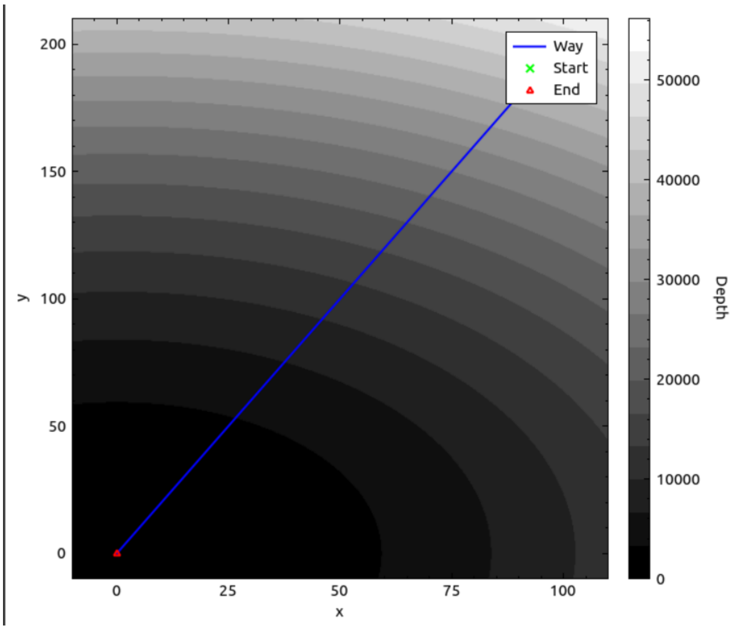
B. Вторая стартовая точка

Постояный шаг

`Шаг:` 0.1

`Количество ходов:` 69.

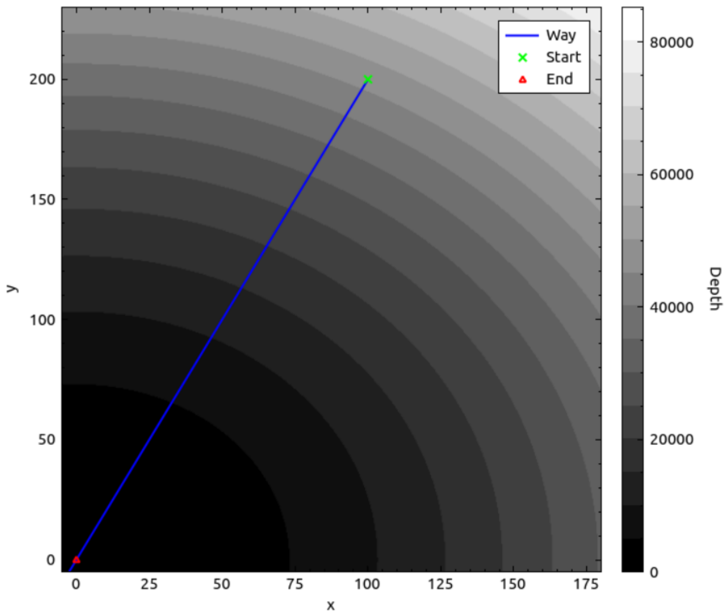
`Результат:` (2.05683e-05, 4.11374e-05)



Дихотомия

`Количество ходов:` 8.

`Результат:` (-5.55112e-15, -1.11022e-14)



ii. Вторая функция

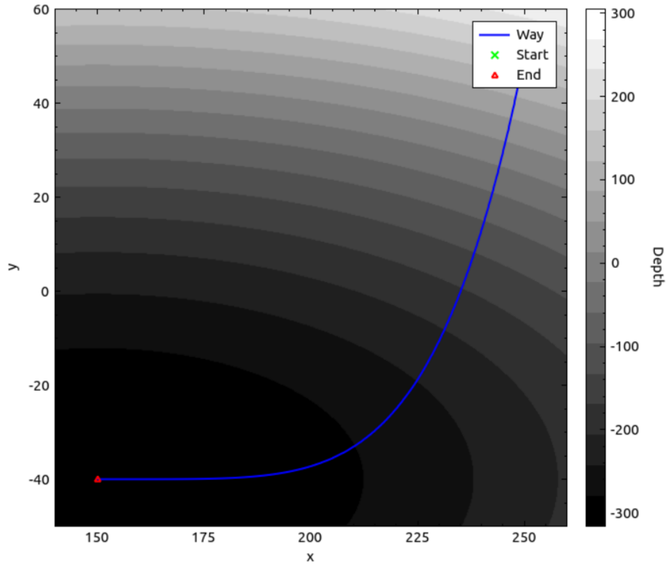
A. Первая стартовая точка

Постояный шаг

`Шаг`: 0.1

`Количество ходов`: 4896.

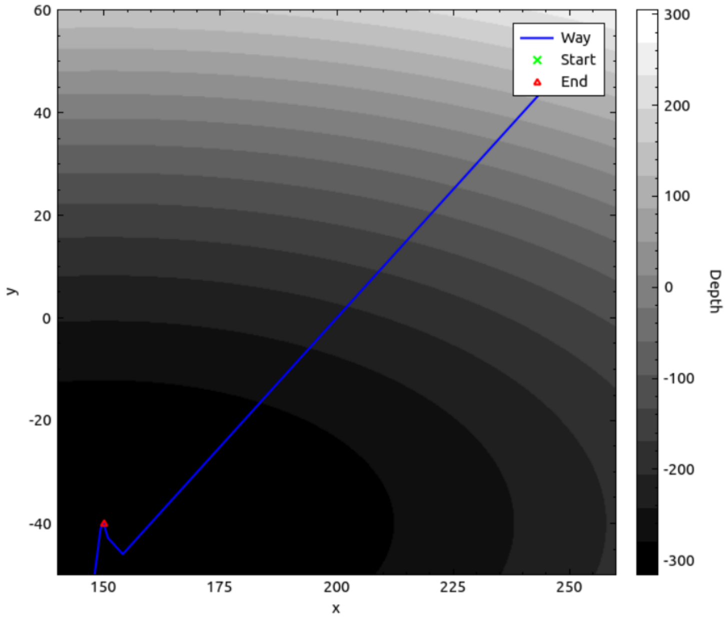
`Результат`: (150.005, -39.9999)



Дихотомия

`Количество ходов`: 17.

`Результат`: (149.998, -40)



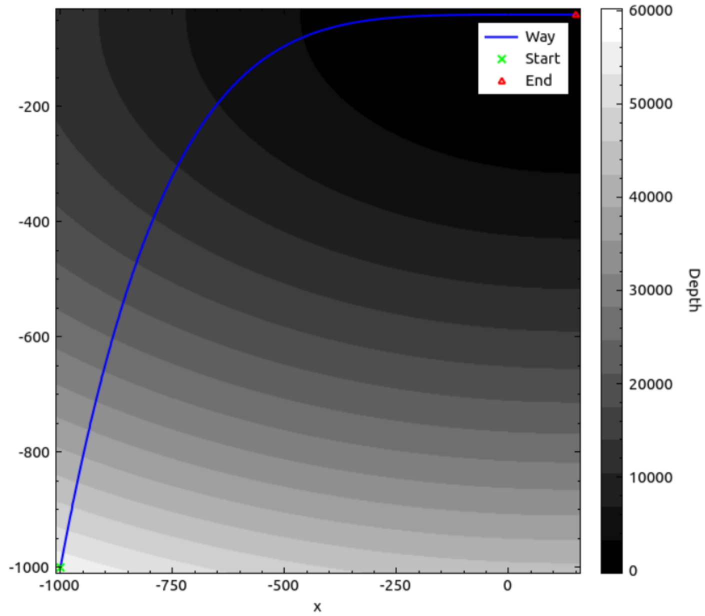
B. Вторая стартовая точка

Постояный шаг

Шаг: 0.1

Количество ходов: 6164.

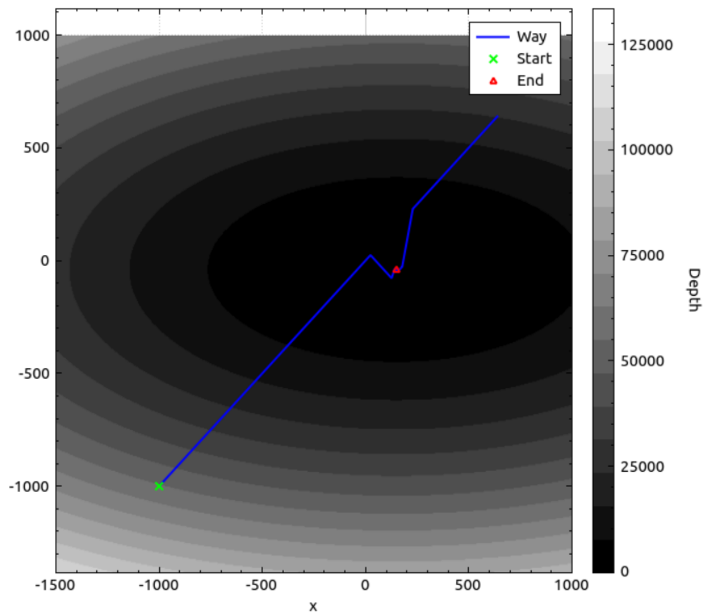
Результат: (149.994, -40.0002)



Дихотомия

Количество ходов: 26.

Результат: (150, -40)



iii. Третья функция

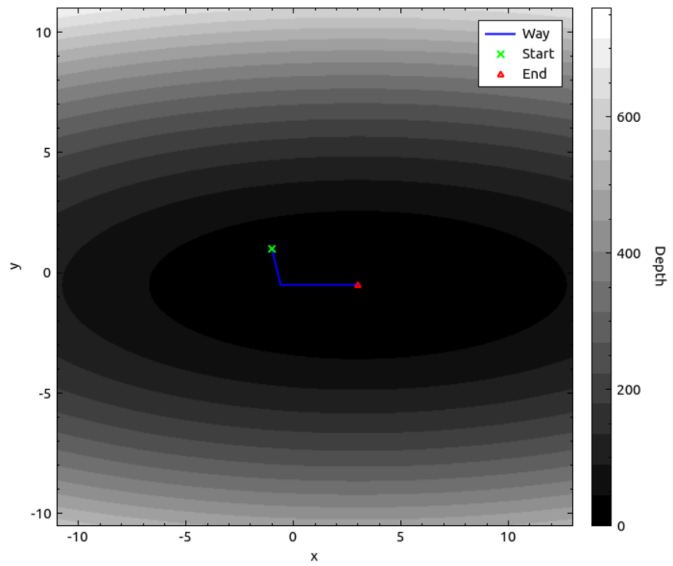
A. Первая стартовая точка

Постояный шаг

`Шаг`: 0.1

`Количество ходов`: 101.

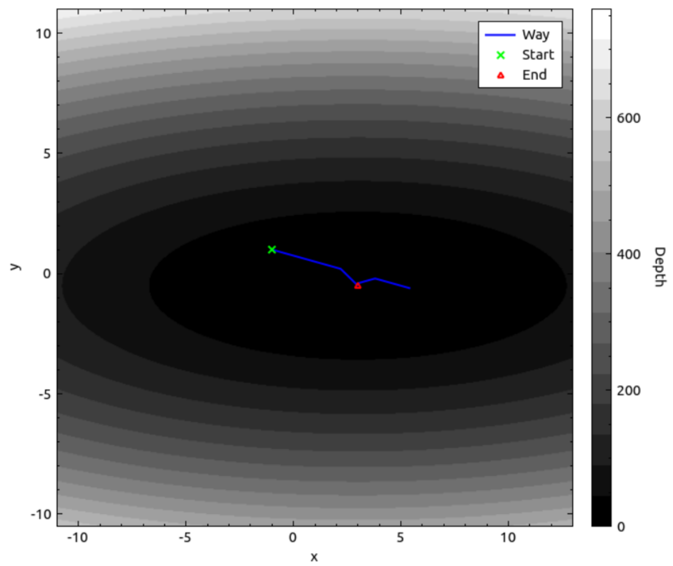
`Результат`: (2.9999, -0.5)



Дихотомия

`Количество ходов`: 5.

`Результат`: (3, -0.5)



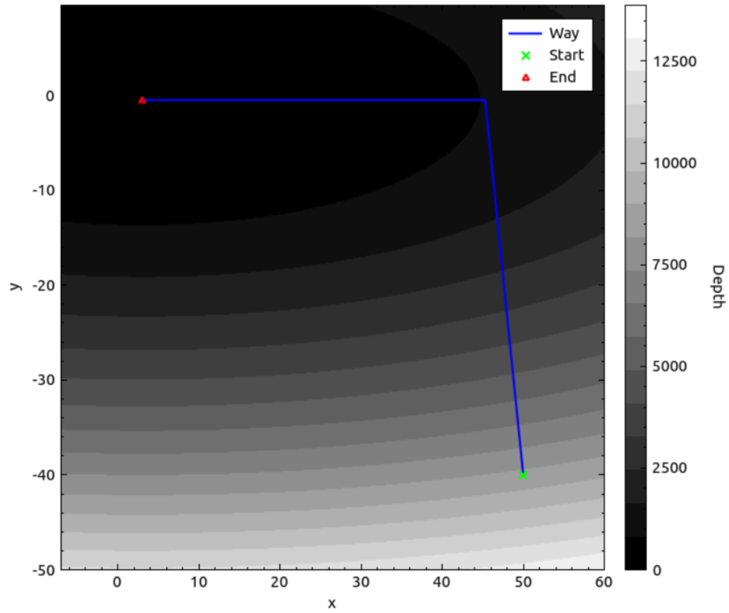
B. Вторая стартовая точка

Постояный шаг

`Шаг`: 0.1

`Количество ходов`: 124.

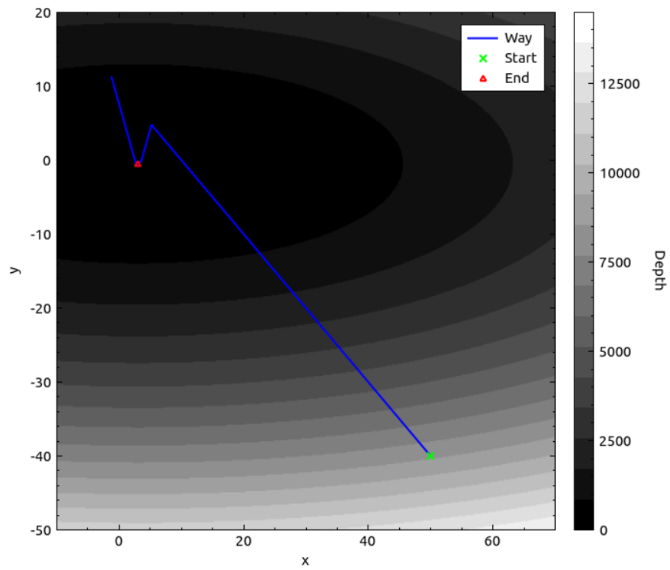
`Результат`: (3.0001, -0.5)



Дихотомия

`Количество ходов`: 10.

`Результат`: (3, -0.5)



Проанализировав полученные результаты, можно сказать следующее. Градиентный спуск с постоянным шагом очень чувствителен к градиенту, причем если градиент меняется медленно в конце пути, то получается очень много шагов. Так что Этот метод лучше использовать с квадратичными функциями у которых ветви поднимаются быстро.

Второй минус обычного градиента заключается в том, что если число обусловленности многомерной вадратичной функции большое, то путь обычного градиента делает большую дугу, что сказывается на большем количестве ходов по правилу треугольника.

Третий не существенный минус заключается в не такой хорошей точности решения как у градиента с дихотомией. Так как обычному градиенту в конце нужно всё больше и больше ходов для удовлетворительной точности, тогда как градиент с оптимизацией не ограничен стороной стартовой точки от точки решения и поэтому получаются более точные результаты.

Ещё стоит отметить скорость увеличения ходов градиентного спуска с оптимизацией от дальности стартовой точки до ответа. Она логарифмическая. Оно и понятно, так как оптимизация похожа на бинарный поиск, который имеет ту же ассимптотику.

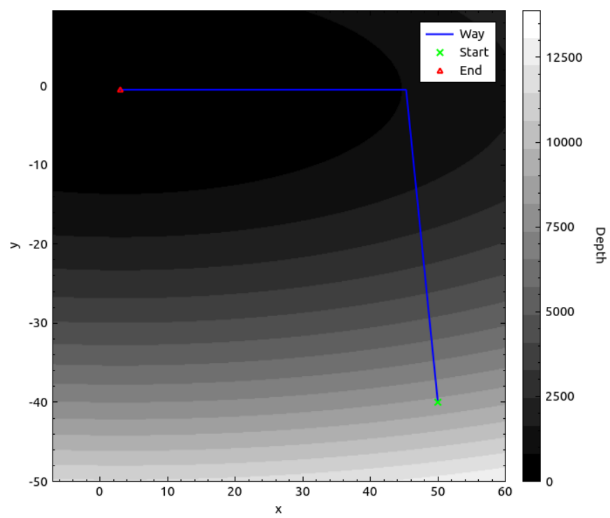
4.4. Для этого пункта возьмём как раз третью функцию из предыдущего пункта $f(x, y) = 0.5x^2 - 3x + 5y^2 + 5y + 6$. У неё k - коэффициент обусловленности равен 10. Посмотрим как выглядел обычный градиентный спуск от дальней точки

Постояный шаг

`Шаг`: 0.1

`Количество ходов`: 124.

`Результат`: (3.0001, -0.5)



И заменим коэффициент при x^2 на 2.

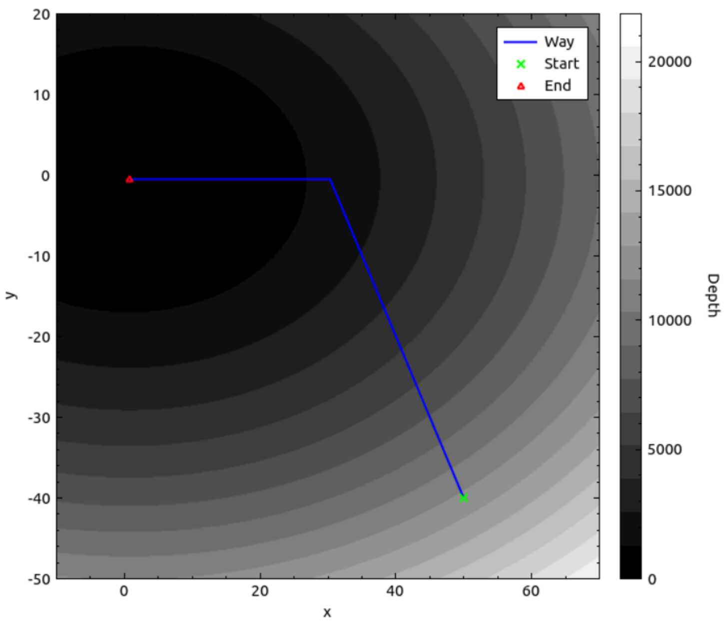
Получим вот такую функцию $f(x, y) = 2x^2 - 3x + 5y^2 + 5y + 6$.

Постояный шаг

`Шаг:` 0.1

`Количество ходов:` 29.

`Результат:` (3, -0.5)



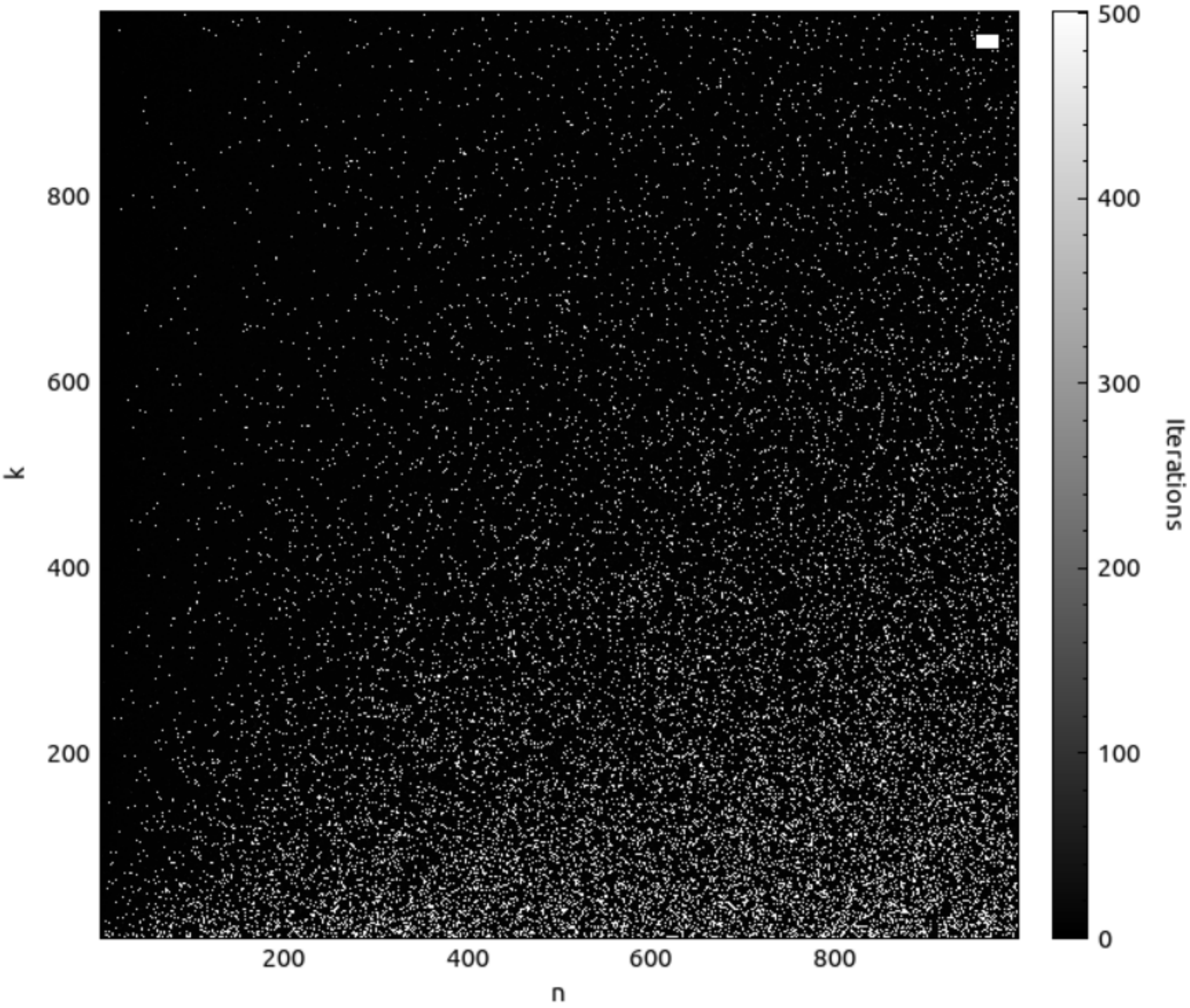
Как мы видим, количество ходов уменьшилось почти в 5 раз при изменении коэффициента обусловленности в 4 раза, что довольно любопытно. То есть тут можно заметить линейную зависимость между этими двумя показателями.

5. Реализация n-мерной квадратичной функции с k-обусловленностью.

```
v<std::function<qreal (qreal)> > generate_func(int n, const qreal k)
{
    v<std::function<qreal(qreal)>> result;
    qDebug() << "======================================";
    qreal min_ = random(1., 10., 5);
    qreal max_ = min_ * k;
    if (max_ < min_) {
        qDebug() << "reversed";
        std::swap(max_, min_);
    }
    qDebug() << min_ << max_;

    qDebug() << "======================================";
    qfunc temp(0, 0, 0);
    while (n-- > 0) {
        temp = qfunc(random(min_, max_, 5), random(-1000000., 10000., 5), random(-1000000., 10000., 5));
        qDebug() << temp.a << temp.b << temp.c;
        result.push_back(temp);
    }
    qDebug() << "======================================";
    return result;
}
```

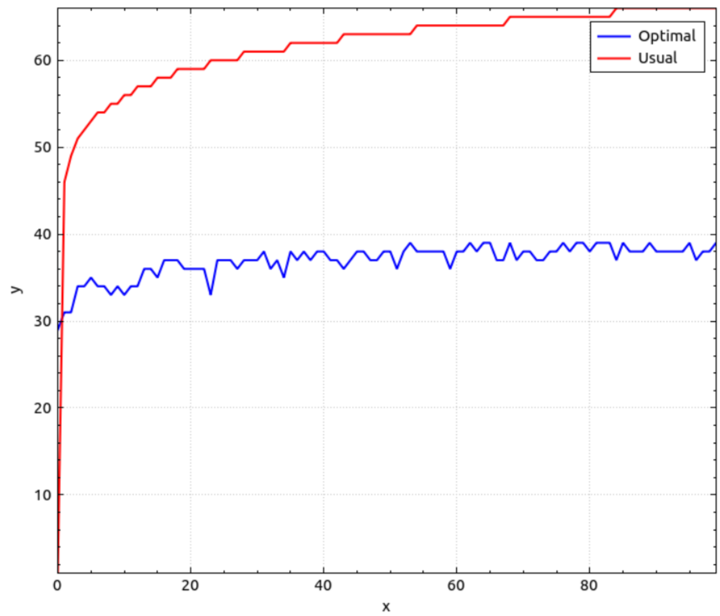
6. Цветовая карта зависимости числа итераций $T(n, k)$



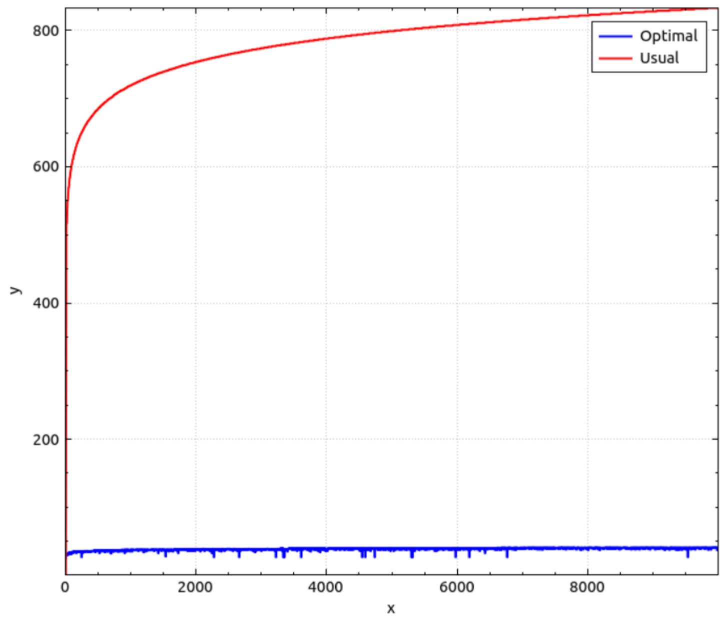
В результате проведенного эксперимента было выяснено, что размерность пространства не оказывает значительного влияния на результат работы алгоритма, важным фактором является лишь число обусловленности функции. При использовании малого шага и низком числе обусловленности алгоритму требуется большое количество итераций. В случае, если число обусловленности высокое, алгоритму потребуется меньшее количество шагов, так как он перескакивает через минимум функции, и процесс оптимизации завершается.

Под конец добавлю график отношения количества итераций у методов к расстоянию стартовой точки до ответа.

6.1. $f(x, y) = x^2 + y^2$



6.2. $f(x, y) = 0.1x^2 + 0.1y^2$



Можно заметить, что чем меньше параметр (a) у квадратичной функции, то различия в количестве ходов для каждого метода будут только усиливаться.

3 Вывод

В процессе работы были изучены методы оптимизации, включая градиентный спуск и градиентный спуск на основе дихотомии. Было проведено сравнение эффективности этих методов на различных функциях, выполнены поставленные задачи и произведен анализ перечисленных методов.