

Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирования
Прикладная математика и информатика

Методы оптимизации

Отчёт по лабораторной работе №1

Работу выполнили:

Щербина К. А.

Чеканова П. А.

Викулаев И.А.

Преподаватель:

Станислав Ким

Санкт-Петербург
2023г.

Задание 1.

Перед нами стоит задача реализации градиентного спуска с постоянным шагом (learning rate).

В методе градиентного спуска нам дается начальное приближение x_0 . Предполагается, что в некоторой окрестности минимума функция выпукла вниз и точка x_0 принадлежит этой окрестности. Метод градиентного спуска носит итерационный характер: мы вычисляем последовательность приближений $x_0, x_1, x_2, x_3, \dots$ в которой текущая точка движется к точке минимума. Количество итераций может быть постоянным, а может быть заранее неизвестно, и даже сходимость метода гарантирована далеко не всегда. Параметром в методе градиентного спуска является размер шага α , в методах машинного обучения его называют также скоростью обучения — *learning rate*.

Определение. Направление вектора p_k называется направлением убывания функции $f(x)$ в точке x_k , если при всех достаточно малых положительных α выполняется неравенство $f(x_k + \alpha p_k) < f(x_k)$.

Пусть функция $f(x)$ дифференцируема в точке x^k . Если вектор p^k удовлетворяет условию $\langle \nabla f(x^k), p^k \rangle < 0$, то направление вектора p^k является направлением убывания.

Предположим, что $p_k = -\nabla f(x_k)$. Если $\nabla f(x_k) \neq 0$, то $\langle \nabla f(x^k), p^k \rangle < 0$, следовательно p_k — направление убывания $f(x)$, причём в малой окрестности точки x_k направление p_k обеспечивает наискорейшее убывание функции. Таким образом, $\exists \alpha_k > 0$, такое что условие остановки итерационного процесса: $\|\nabla f(x_k)\| < \varepsilon$ выполнено.

Алгоритм метода градиентного спуска:

1. Выбрать $\varepsilon > 0$, $\alpha > 0$ (learning rate), $x \in E_n$, вычислить $f(x)$.
2. Вычислить $\nabla f(x)$. Проверить условие $\|\nabla f(x)\| < \varepsilon$. Если оно выполнено, то завершить процесс, иначе перейти к шагу 3.
3. Найти $y = x - \alpha \nabla f(x)$ и $f(y)$. Если $f(y) < f(x)$, то положить $x = y$, $f(x) = f(y)$, и перейти к шагу 2, иначе — повторить 3 шаг.

[Код реализации градиентного спуска\(gradient\)](#)

Задание 2.

В ходе выполнения лабораторной работы был реализован выбранный метод одномерного поиска (метод Дихотомии).

Метод Дихотомии - это улучшенный алгоритм одномерного поиска, который использует принцип деления отрезка пополам.

Метод определяет длину шага градиентного спуска.

Для этого выбираются две точки на линии, соединяющей x_0 и $x_0 - \alpha \nabla f(x_0)$, и вычисляется значение функции в этих точках. Затем отбрасывается половина отрезка, в которой значение функции больше, и процесс повторяется до тех пор, пока не будет достигнута требуемая точность.

Таким образом, градиентный спуск с методом Дихотомии позволяет быстро и эффективно находить оптимальное значение функции, используя информацию о ее градиенте и метод деления отрезка пополам.

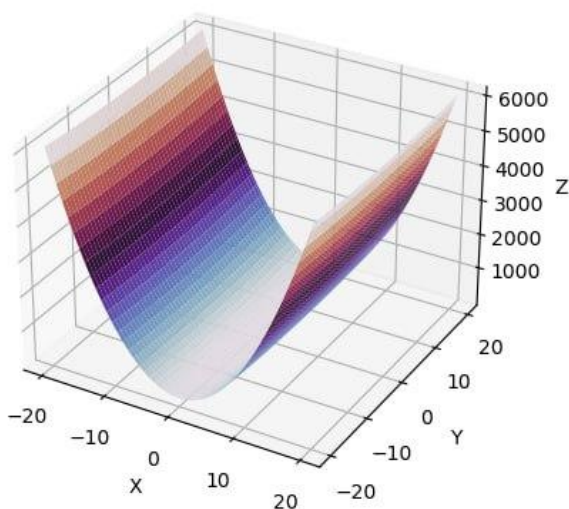
Код метода(one_dim + dichotomy_search)

Задание 3.

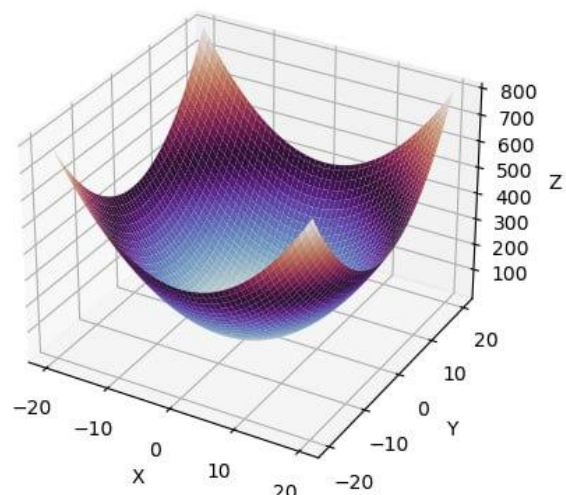
Проанализируйте траекторию градиентного спуска на примере квадратичных функций.

Для анализа мы взяли 2 функции:

$$z(x, y) = x^2 + y^2$$



$$z(x, y) = 15x^2 + 0.3y^2$$



Они достаточно подробно иллюстрируют различия в подходах методов оптимизации. Первая - канонический параболоид с единственным минимумом в точке

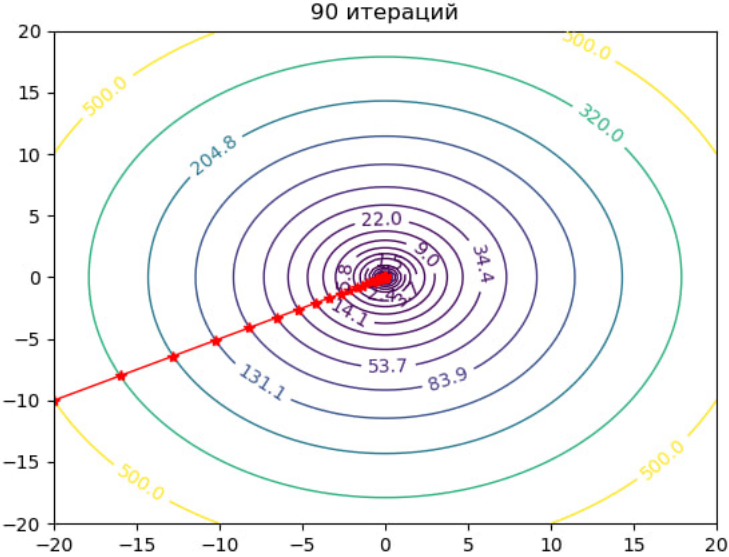
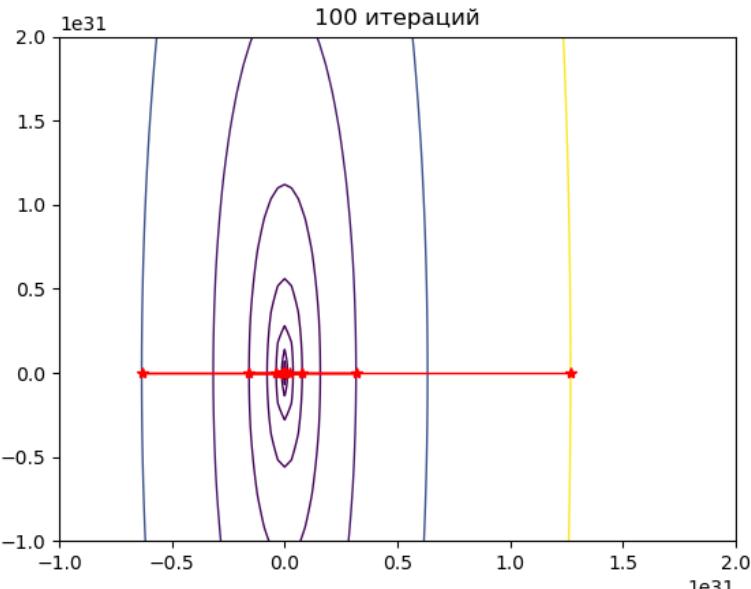
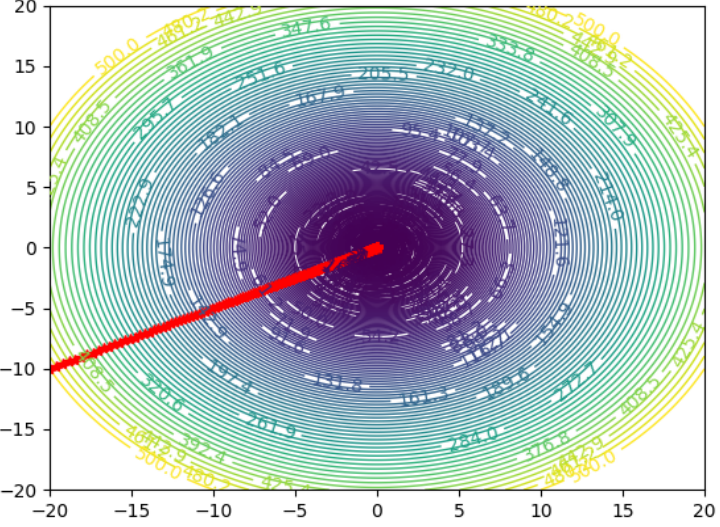
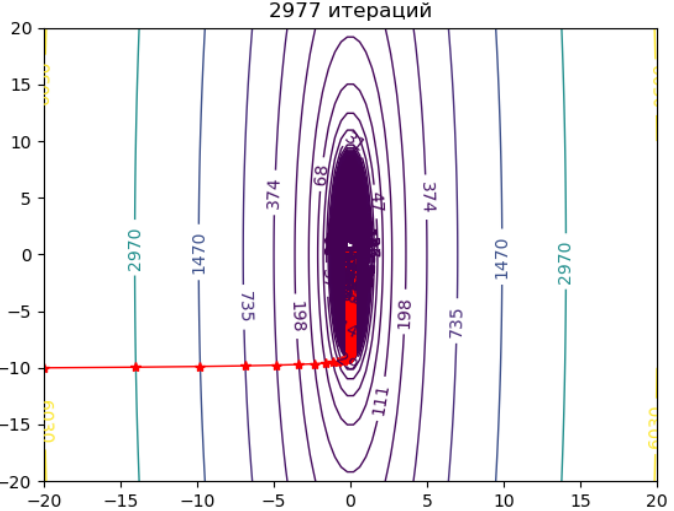
$(0; 0)$. Вторая - параболоид с единственным минимумом в точке $(0; 0)$, но с растянутыми осями координат. Мы решили не сдвигать координатные оси, потому что особого смысла в этом не наблюдается.

Задание 4.

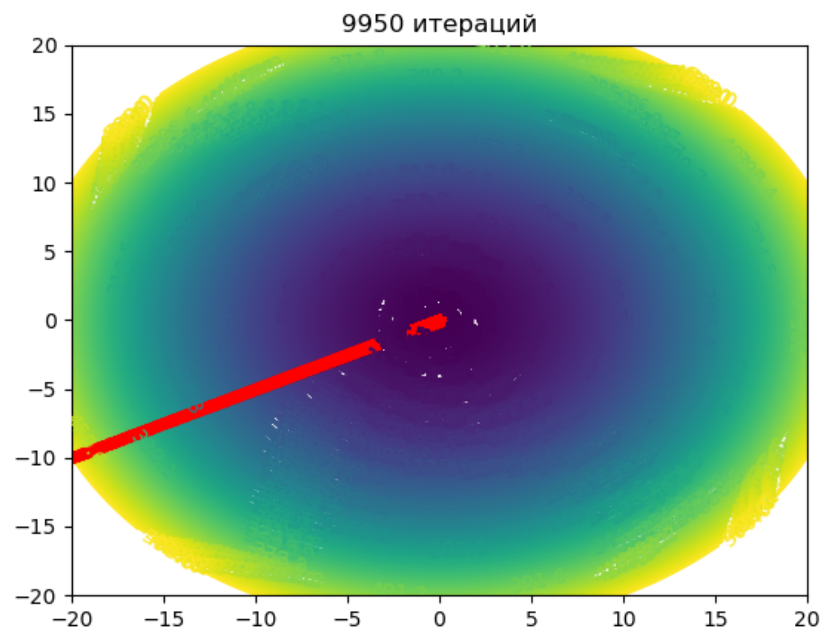
Все необходимые для тестирования функции можно посмотреть [здесь](#)

4.а Исследуем сходимость градиентного спуска с постоянным шагом и сравним полученные результаты для выбранных функций.

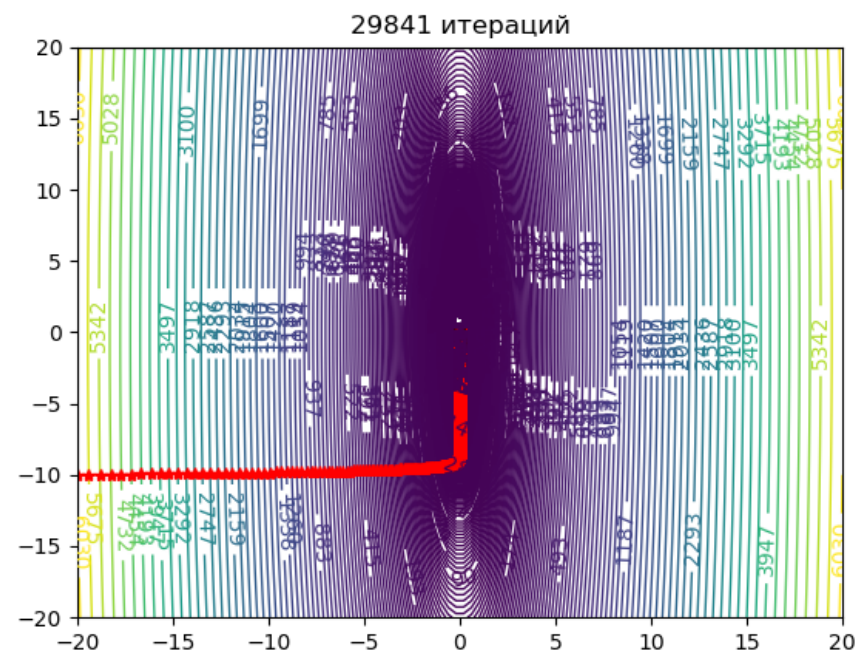
ПОСТОЯННЫЙ ШАГ

lr	$z(x, y) = x^2 + y^2$	$z(x, y) = 15x^2 + 0.3y^2$
1e-1	<p>90 итераций</p>  <p>min = [-3.55580767e-08 3.55580767e-08]</p>	<p>100 итераций</p>  <p>разошёлся</p>
1e-2	<p>986 итераций</p>  <p>min = [3.57951599e-08 3.57951599e-08]</p>	<p>2977 итераций</p>  <p>min = [-4.94065646e-324 -1.66678647e-007]</p>

1e-3



min = [-3.5743748e-08 3.5743748e-08]



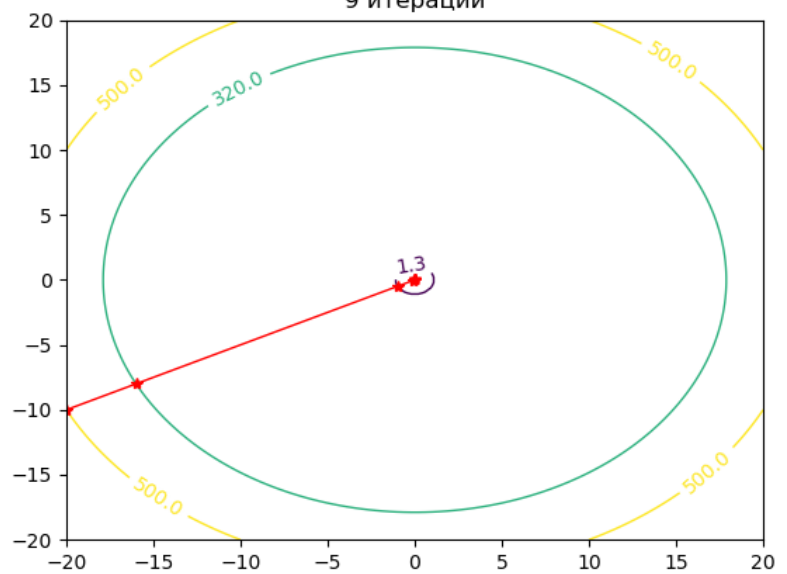
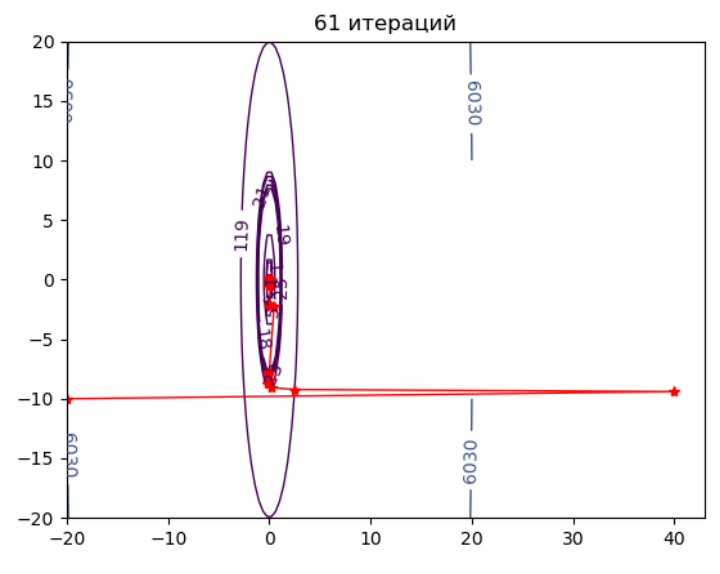
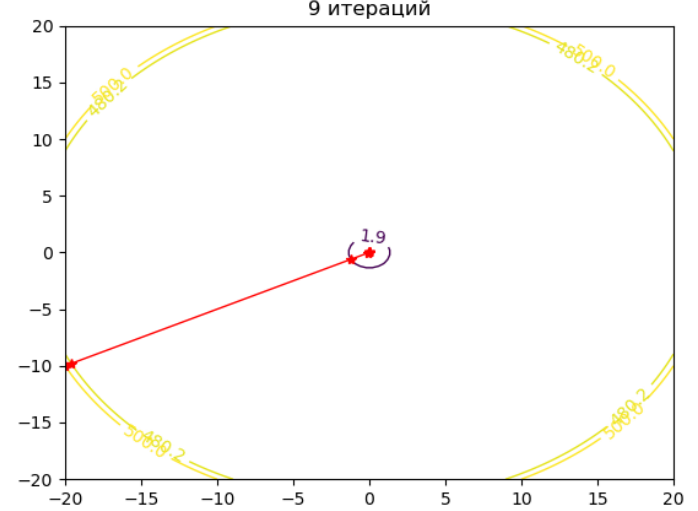
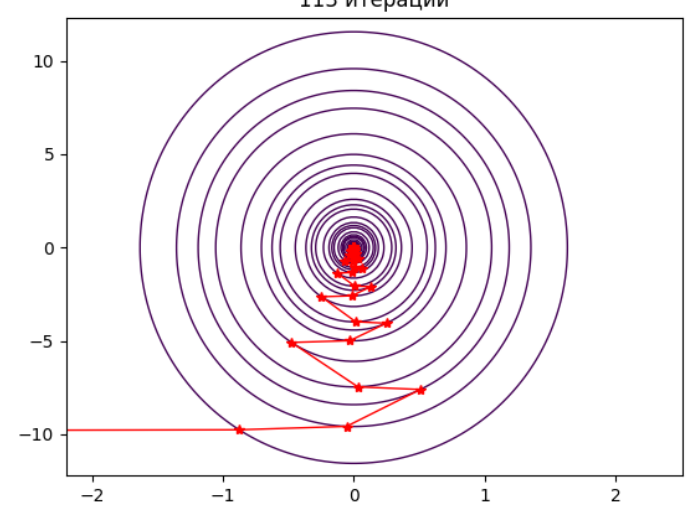
min = [-7.90505033e-323 -1.66746981e-007]

Заметим, что при одинаковых стартовых условиях [start_point = [-20;-10]] градиентный спуск с постоянных шагом работает неоптимально, ему требуется достаточно много шагов, чтобы достигнуть минимума даже канонической кривой второго порядка. Также можно отметить, что траектории сходимости (1) и (2) функций принципиально отличаются, на это влияет обусловленность функции. Градиентному спуску сложнее достигнуть минимума с плохо обусловленной функцией (2).

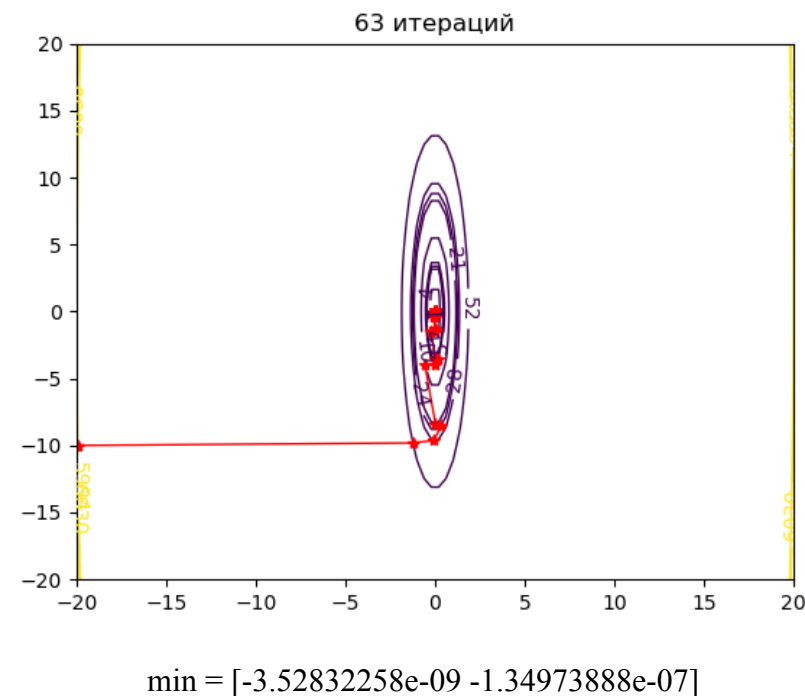
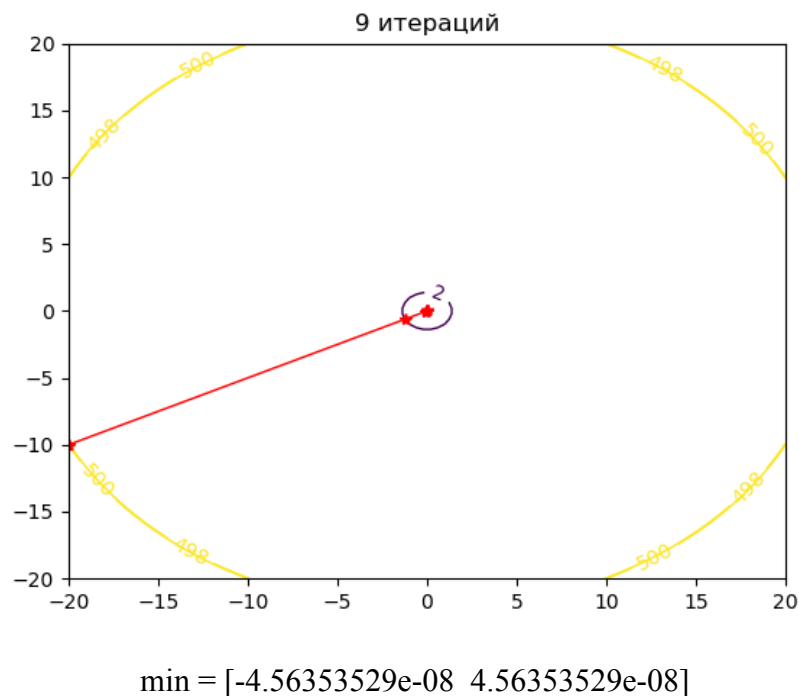
Если сравнивать зависимость сходимости функции от взятого шага, можно заметить, что если шаг слишком большой, минимум может не найтись, метод просто “перешагнёт” через него, в то время как выбрав слишком маленький шаг, мы увеличим количество итераций, за которое наш метод найдёт минимум, что довольно очевидно.

4.b Сравним эффективность градиентного спуска с использованием одномерного поиска с точки зрения количества вычислений минимизируемой функции и ее градиентов.

ОДНОМЕРНЫЙ ПОИСК

lr	$z(x, y) = x^2 + y^2$	$z(x, y) = 15x^2 + 0.3y^2$
1e-1	<p>9 итераций</p>  <p>min = [-5.84130767e-08 5.84130767e-08]</p>	<p>61 итераций</p>  <p>min = [1.78068671e-08 -8.45412518e-08]</p>
1e-2	<p>9 итераций</p>  <p>min = [9.12705691e-08 9.12705691e-08]</p>	<p>113 итераций</p>  <p>min = [6.87961545e-09 -1.42804189e-07]</p>

1e-3



Градиентный спуск с использованием одномерного поиска работает лучше, чем градиентный спуск с постоянным шагом, так как мы на каждой итерации подбираем оптимальный шаг, что позволяет нам быстрее добраться до минимума функции.

Различие в скорости очевидно, особенно хорошо видна разница на меньших параметрах l_r и плохо обусловленной функции.

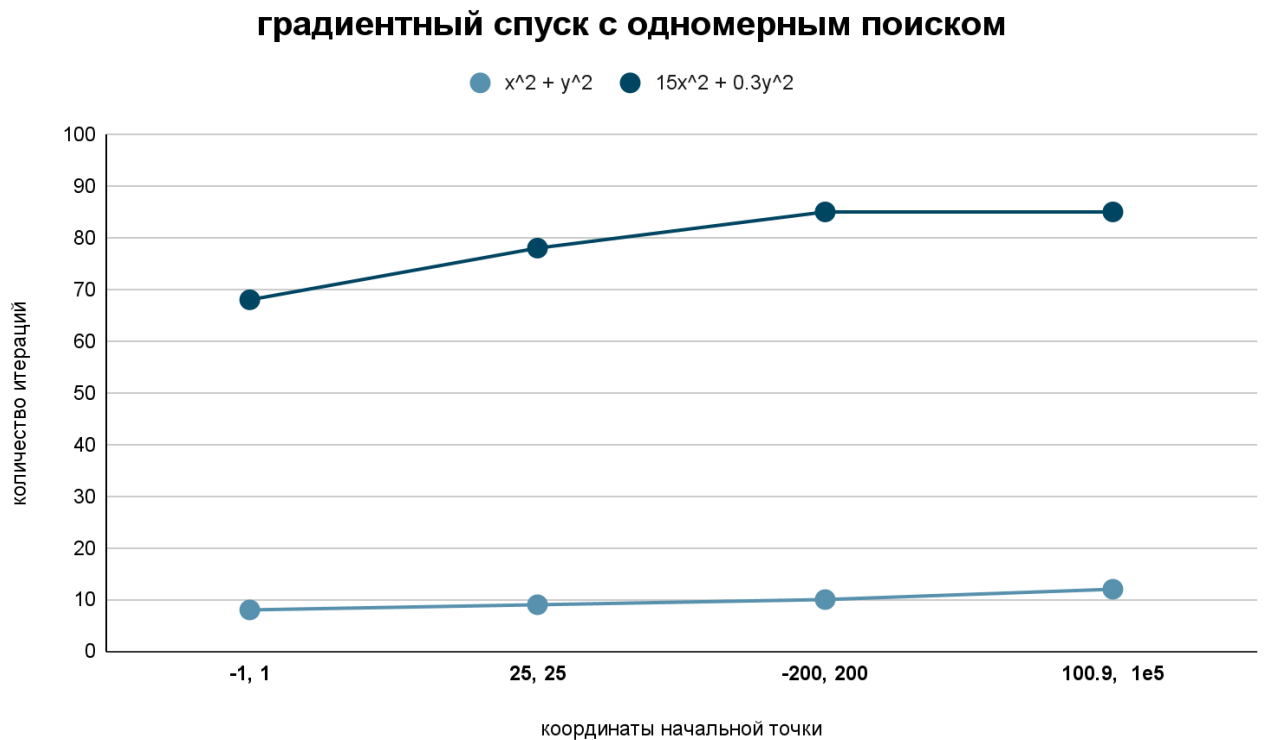
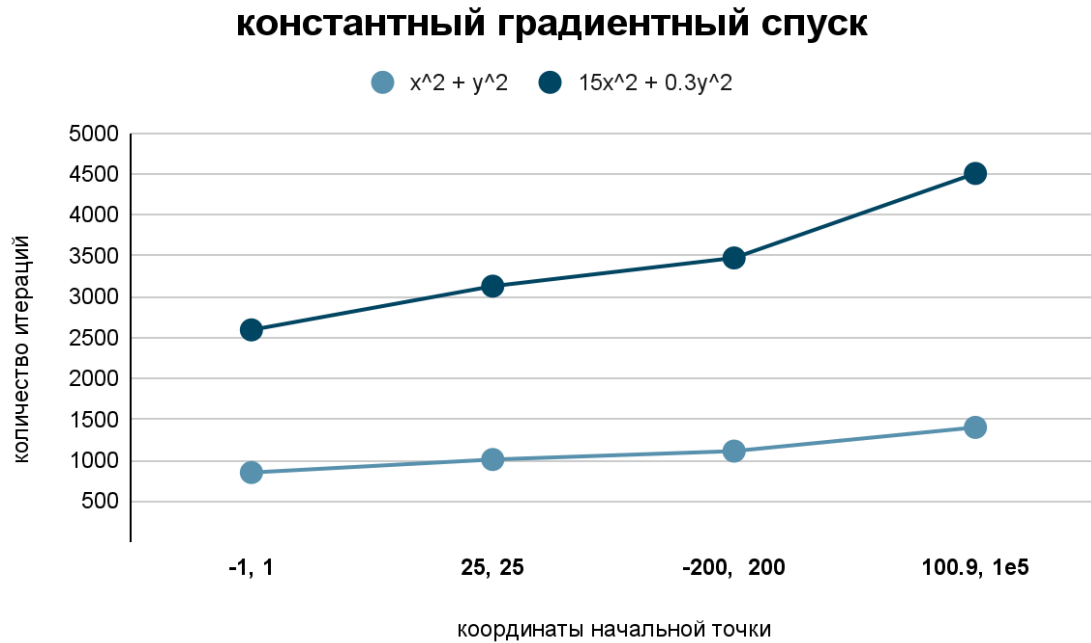
Также мы можем видеть, что хорошо обусловленная функция (1) сошлась за 9 итераций с использованием метода дихотомии, в то время как градиентный спуск с постоянным шагом сошелся за 90 | 986 | 9950. Кроме того, плохо обусловленная функция (2) с использованием метода дихотомии сошлась за 61 | 113 | 63 итераций, тогда как градиентный спуск с постоянным шагом сойдется за ∞ | 2977 | 29841 итераций. Помимо скорости сходимости мы видим, что при достаточно большом шаге градиентный спуск с использованием одномерного поиска не расходится.

Заметим, что метод одномерного поиска на функции (2) сработал оптимальнее на начальных значениях шага, равных $1e-1$ и $1e-3$, тогда как с $l_r = 1e-2$, метод работал чуть медленнее. Это обусловлено принципом работы метода Дихотомии, самим коэффициентом обучения, выбором начальной точки и видом функции, минимум которой мы хотим найти

НЕПРАВИЛЬНО ВЫБРАН ИЗНАЧАЛЬНЫЙ РАЗМЕР ШАГА. ОН ДОЛЖЕН БЫТЬ СИЛЬНО БОЛЬШЕ, ЧЕМ У ПОСТОЯННОГО, СКАЖЕМ 20, 50, 100

4.с Исследуем работу методов в зависимости от выбора начальной точки;

Интуитивно понятно, что чем дальше начальная точка от минимума, тем большее количество итераций градиентного спуска нам потребуется, чтобы этот минимум достичь.



координаты начальной точки	(-1, 1)	(25, 25)	(-200, 200)	(100.9, 1e5)
количество итераций для константного град.спуска $f(x) = x^2 + y^2$	850	1009	1112	1402
количество итераций для град.спуска с одномерным поиском $f(x) = x^2 + y^2$	8	9	10	12
количество итераций для константного град.спуска $f(x) = 15x^2 + 0.3y^2$	2594	3129	3474	4507
количество итераций для град.спуска с одномерным поиском $f(x) = 15x^2 + 0.3y^2$	68	78	85	85

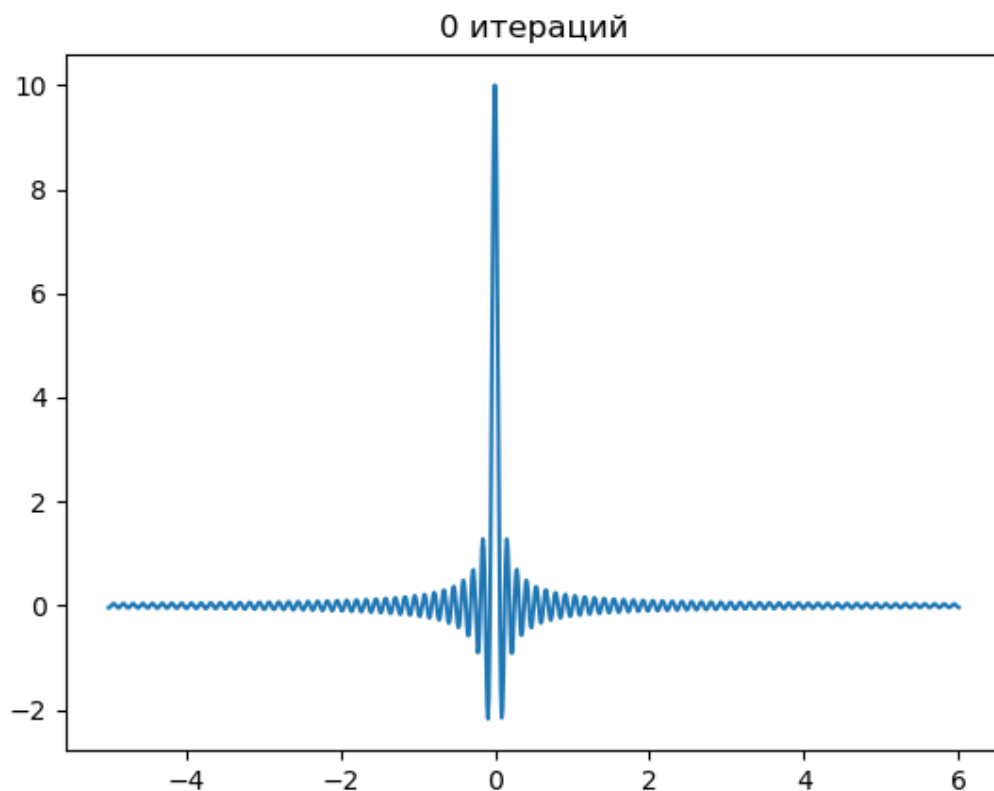
Наше предположение подтвердилось. Чем дальше от минимума начальная точка, тем больше усилий (итераций) потребуется градиентному спуску для нахождения минимума функции. Также можно заметить большую разницу в количестве операций между хорошо обусловленной функцией (1) и плохо обусловленной функцией (2). Кроме того данный график наглядно иллюстрирует преимущества градиентного поиска с использованием одномерного поиска, количество необходимых для поиска минимума итераций сильно сокращается.

4.d Исследуем влияние нормализации (scaling) на сходимость на примере масштабирования осей плохо обусловленной функции.

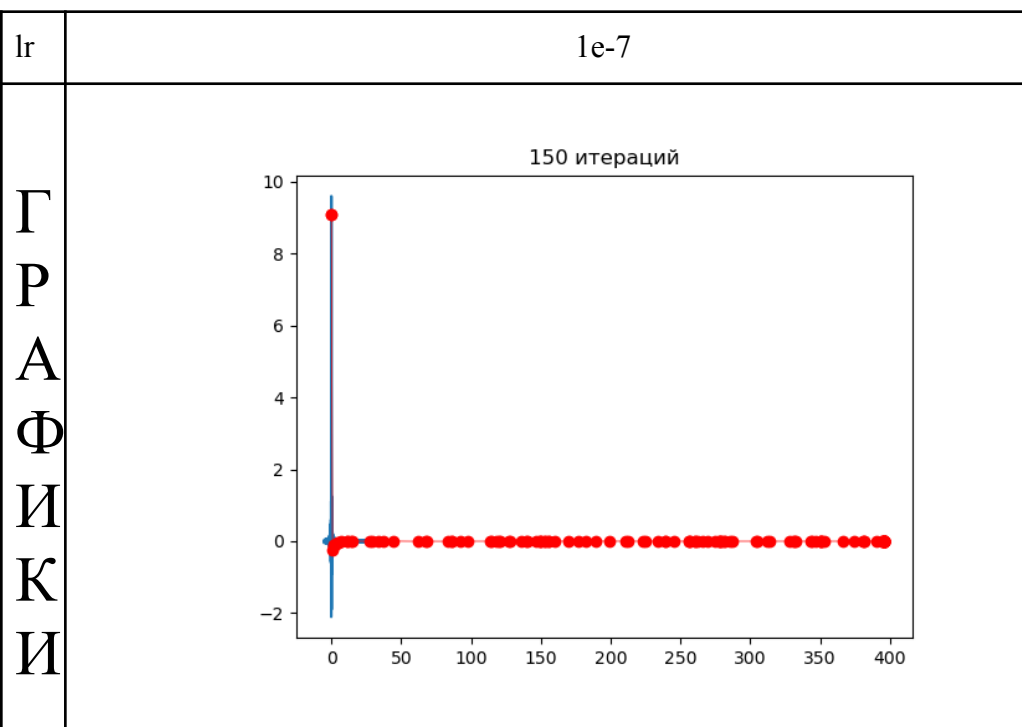
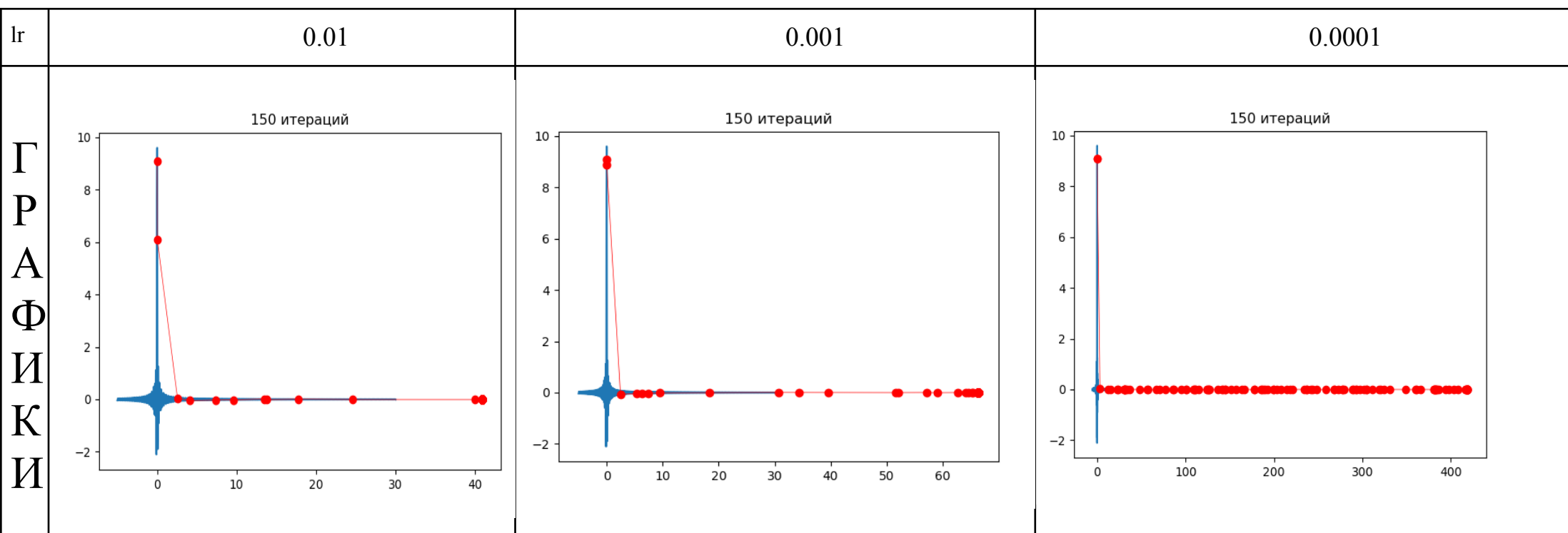
Рассмотрим результаты работы алгоритма в пунктах 4.a и 4.b. В рамках нашей лабораторной работы плохо обусловленной функцией выступает третья функция. Основываясь на наших результатах можно сделать вывод: чем хуже обусловлена функция, тем больше алгоритму требуется итераций, чтобы достигнуть минимума.

Приведём пример другой функции.

$f(x) = 10 * \sin(ax) \div ax$, где параметр a отвечает за растяжение оси OX. Начальным значением $a = -50$.

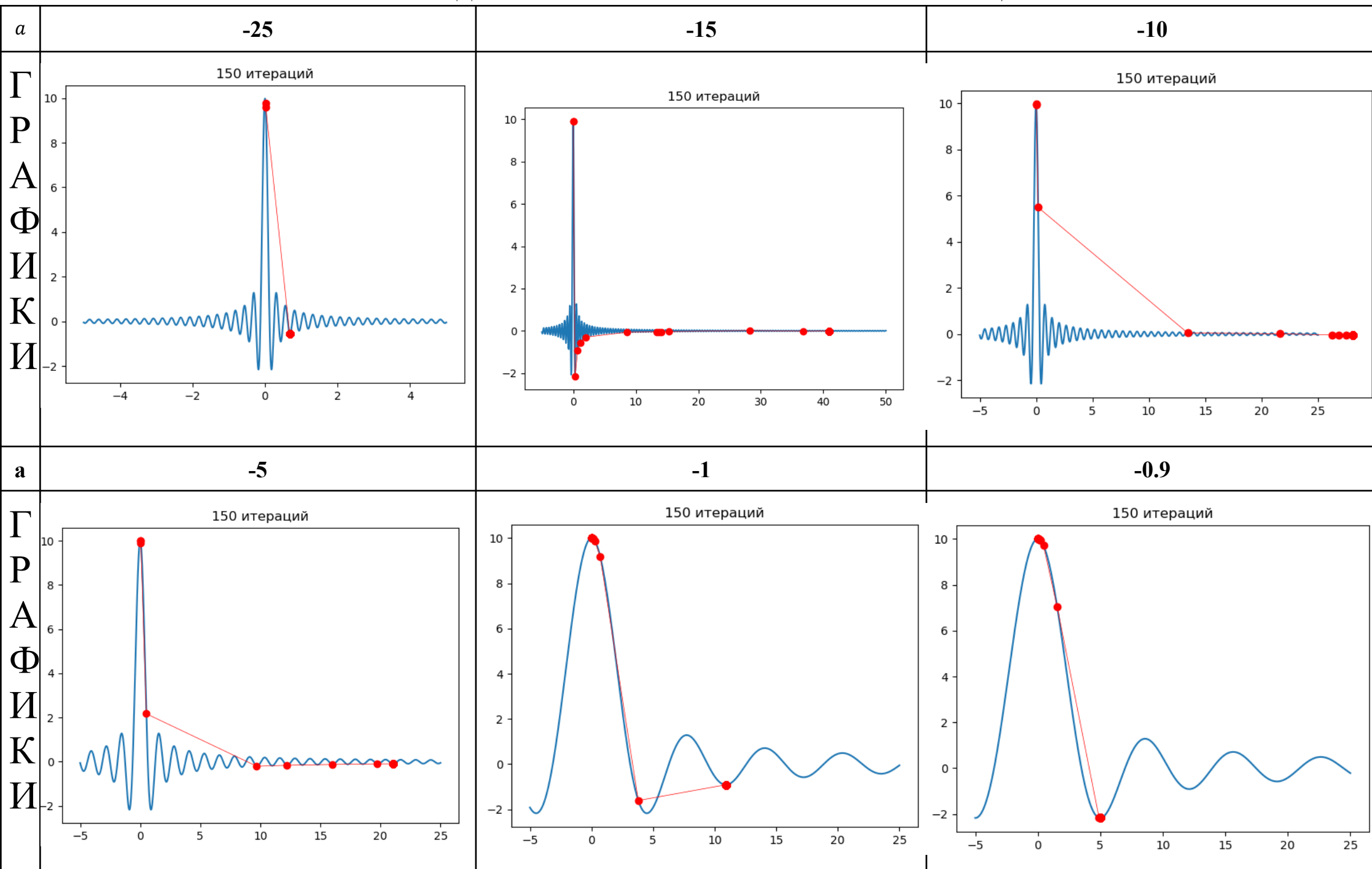


Нетрудно заметить, что в окрестности нуля функция очень чувствительна к малым изменениям x . Посмотрим, как справится дихотомия с поиском минимума при различных learning rate, запущенная из одной точки $x = 0.015$.



Отметим, что градиентный спуск не только пропустил минимум функции, но с уменьшением learning rate отдалается от минимума функции. Нормализуем ось x постепенно и посмотрим как работает наш алгоритм при одинаковых условиях. [learning rate = 0.01, $x_0 = 0.015$]

СХОДИМОСТЬ В ЗАВИСИМОСТИ ОТ НОРМАЛИЗАЦИИ



Как мы и говорили, основываясь на предыдущих пунктах, градиентный спуск сходится и находит минимум лучше тогда, когда функция как можно лучше обусловлена. Это прекрасно видно на приведённом примере.

Задание 5.

Цель: реализовать генератор случайных квадратичных функций n переменных с числом обусловленности k .

Как известно, любую квадратичную форму можно представить в виде:

$$f(x) = \sum_{i=1}^n \sum_{j=1}^n (a_{ij} x_i x_j) = \sum_{i=1}^n (a_{ij} x_i) + 2 \sum_{1 \leq i < j \leq n} (a_{ij} x_i x_j) \quad (1)$$

где x - вектор переменных, а a_{ij} - коэффициент формы.

Нетрудно заметить, что между пространством квадратичных функций и пространством симметричных матриц $\mathbb{R}^{n \times n}$. Тогда саму квадратичную форму можно переписать в виде:

$$f(x) = x^T A x \quad (2)$$

Тогда в рамках лабораторной работы нам достаточно создать функцию, которая сможет сгенерировать матрицу A и на основе её создать квадратичную форму и ее градиент.

Для симметричных и положительно определенных матриц число обусловленности вычисляется как отношение максимального и минимального собственных чисел. Для любых других это число выражается как отношение максимального и минимального сингулярных чисел. В рамках лабораторной работы мы будем генерировать функцию от отношения ее собственных чисел.

Получается, что нам нужно найти симметричную и положительно определенную матрицу A . В силу свойств симметричной матрицы мы знаем, что существует такой базис из собственных векторов матрицы A , что $A = T \Lambda T^{-1}$. Где Λ - диагональная матрица собственных чисел A . Так как матрица симметрична, то все её собственные подпространства ортогональны, а значит $T = Q$ - ортогональная матрица, а $T^{-1} = Q^{-1} = Q^T$. Такую матрицу можно получить из QR - разложения произвольной матрицы B . В силу равносильности всех преобразований мы можем уверенно сказать, что полученная матрица A будет описывать нужную функцию.

Исходя из формулы (2) мы можем понять, как будет выглядеть код для самой квадратичной функции на основе матрицы A .

Градиент рассчитывается следующим образом: нетрудно заметить из формулы (2), что квадратура записывается в виде скалярного произведения.

$$f(x) = \langle Ax, x \rangle \\ y = Ax$$

Тогда:

$$\frac{d(x^T A x)}{dx} = \frac{\partial(x^T y)}{\partial x} + \frac{d(y(x)^T)}{dx} \frac{\partial(x^T y)}{\partial y} = y + \frac{d(x^T A^T)}{dx} x = y + A^T x = (A + A^T)x$$

В силу симметричности нашей матрицы мы получаем:

$$\nabla_x f(x) = \nabla_x (x^T A x) = 2Ax$$

Важно отметить, что градиентный спуск, запущенный из точки начала координат не сдвинется с места. Поэтому перед генерацией функции создадим сдвиг относительно начала координат, условно в пределах $[-5; 5]$.

Код реализации генератора вы можете посмотреть в файле [quadratic_form_generator.py](#)

Задание 6 и 7.

Все необходимые для тестирования функции можно посмотреть [здесь](#)

Таблица зависимости числа итераций $T(n, k)$, необходимых градиентному спуску для сходимости, от размерности пространства $2 \leq n \leq 10^3$ и числа обусловленности оптимизируемой функции $1 \leq k \leq 10^3$

n \ k	2	3	4	5	6	7	8	9
1	8	8	8	8	8	8	8	8
2	14	12	12	9	13	14	13	13
3	22	26	23	20	20	18	19	21
4	10	23	18	23	26	17	17	22
5	24	25	30	40	27	35	39	36
6	38	36	38	40	38	39	35	35
7	66	40	53	68	38	43	55	43
8	12	39	35	40	21	33	25	20
9	50	51	49	46	45	50	44	50
10	67	63	65	63	67	65	65	64
20	10000	10000	10000	10000	10000	10000	10000	10000
30	10000	10000	10000	10000	10000	10000	10000	10000
40	10000	10000	10000	10000	10000	10000	10000	10000
50	10000	10000	10000	10000	10000	10000	10000	10000
60	10000	10000	10000	10000	10000	10000	10000	10000
70	10000	10000	10000	10000	10000	10000	10000	10000
80	10000	10000	10000	10000	10000	10000	10000	10000
90	10000	10000	10000	10000	10000	10000	10000	10000
100	10000	10000	10000	10000	10000	10000	10000	10000
200	10000	10000	10000	10000	10000	10000	10000	10000
300	10000	10000	10000	10000	10000	10000	10000	10000
400	10000	10000	10000	10000	10000	10000	10000	10000
500	10000	10000	10000	10000	10000	10000	10000	10000
600	10000	10000	10000	10000	10000	10000	10000	10000
700	10000	10000	10000	10000	10000	10000	10000	10000
800	10000	10000	10000	10000	10000	10000	10000	10000
900	10000	10000	10000	10000	10000	10000	10000	10000

При проведении тестов мы использовали дихотомию и добавили к критерию остановки по максимальной итерации граничное значение нормы вектора градиента в точке.

Заметим, что между значениями числа обусловленности 10 и 20 идёт резкий скачок числа итерации. Более того видно, что от размерности пространства не изменяется количество итераций, но наблюдается зависимость числа итераций от числа обусловленности.

Для получения более корректных результатов мы провели множественный эксперимент и усреднили полученные значения числа итераций.

Условия Вольфе

Код метода(one dim + wolfe search)

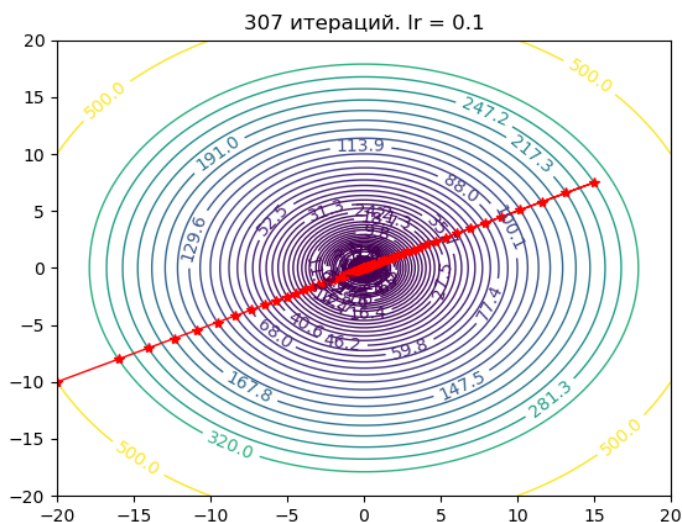
Требуется реализовать одномерный поиск с учётом условий Вольфе и исследовать его эффективность.

Условия Вольфе - набор условий, которые позволяют найти оптимальный коэффициент обучения. Высчитываются они следующим образом: пусть у нас есть некоторое приближение минимума функции x_k и направление в котором лежит минимум p_k . Тогда, чтобы найти следующее приближение нам нужно: $x_{k+1} = x_k + \alpha_k p_k$, где α_k - искомый коэффициент обучения. Он обязан удовлетворять следующим условиям:

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k,$$
$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k$$

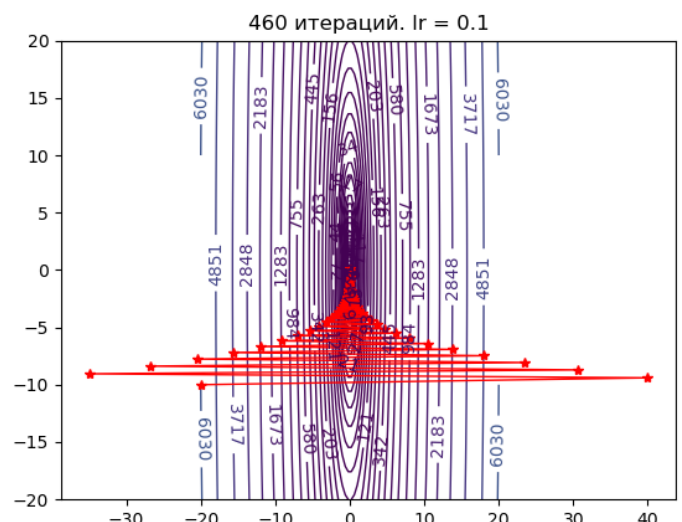
Коэффициенты c_1 и c_2 должны быть подобраны так, что $0 < c_1 < c_2 < 1$. Принято брать константы так, чтобы c_1 находилась в окрестности нуля [эта константа отвечает за уменьшение значения функции], в то время как c_2 находилась в окрестности 1 [отвечает за то, что проекция градиента в новом приближении изменяет своё направление или уменьшается].

Канонический параболоид

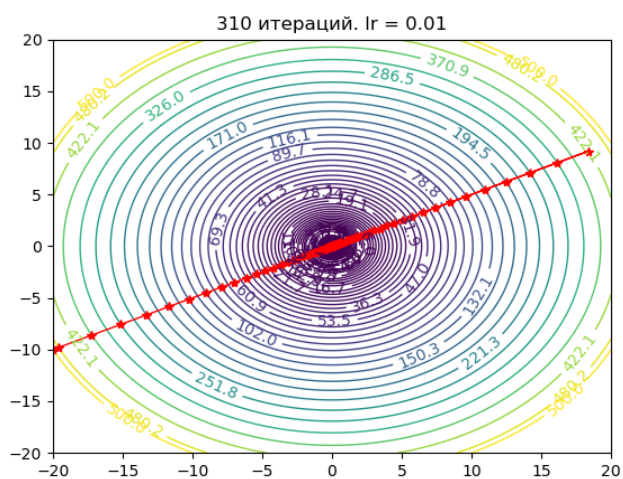


min = [4.52229927e-08 2.26114964e-08]

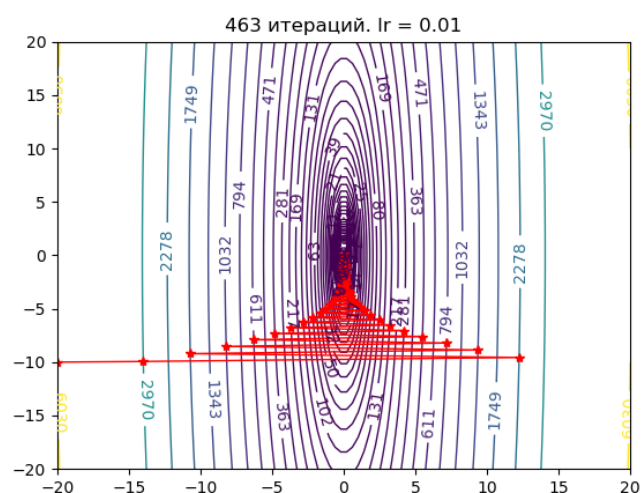
$z(x, y) = 15x^2 + 0.3y^2$



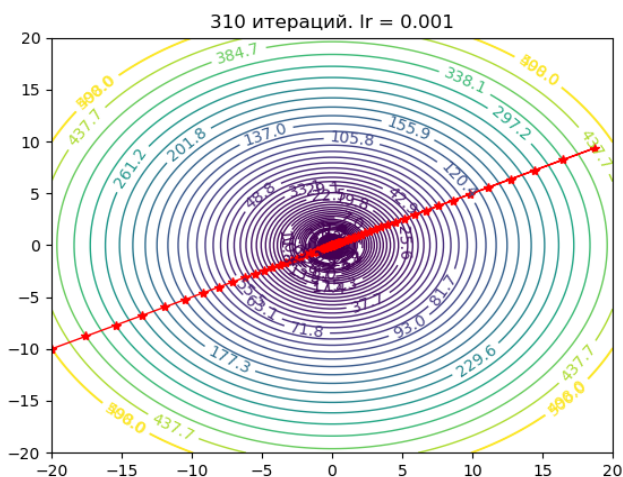
min = [3.07458803e-09 -8.44091844e-08]



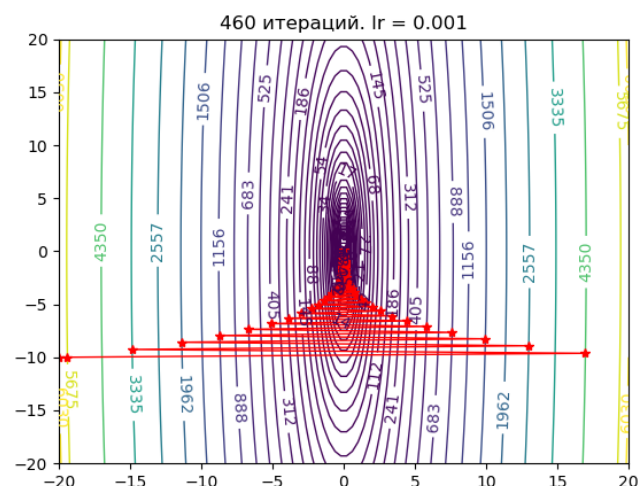
$\min = [-4.56466823e-08 \ -2.28233411e-08]$



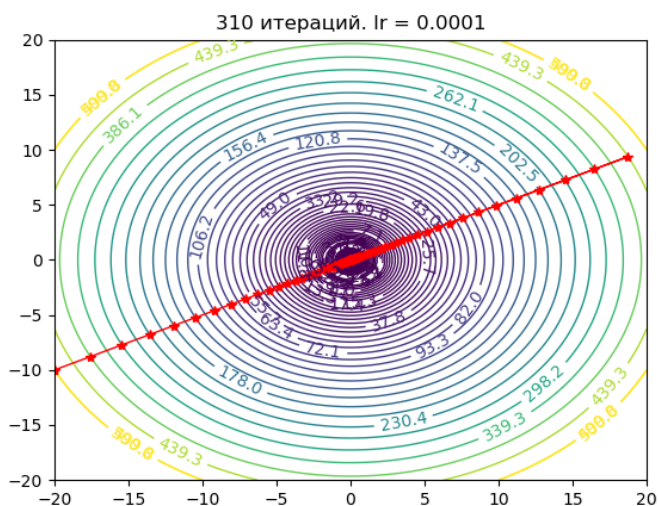
$\min = [3.09327878e-09 \ -7.65178753e-08]$



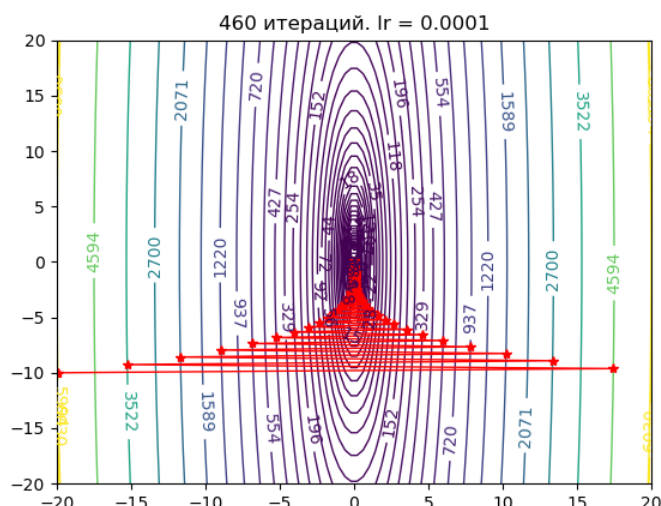
$\min = [-4.64850907e-08 \ -2.32425454e-08]$



$\min = [-3.08886290e-09 \ -8.79948838e-08]$



$\min = [-4.65689316e-08 \ -2.32844658e-08]$



$\min = [-3.17484156e-09 \ -8.80424295e-08]$

По результатам экспериментов мы видим, что одномерный поиск, основанный на условиях Вольфе действительно находит минимум функции, однако в нашем случае он значительно проигрывает обычной Дихотомии. Всё потому что условия Вольфе не гарантируют нам то, что мы найдём минимум быстрее любого другого метода, однако он гарантирует, что шаг будет достаточно маленьким, чтобы градиентный спуск не проскочил через минимум или же не застрял в локальном минимуме.

Например, на $y = |x|$ Дихотомия, запущенная из точки $x = -2$ будет ходить около минимума и даже находиться в нём, а Вольфе сойдётся за 1 итерацию.

