Методы оптимизации 1

Назаров Владимир Целиков Владислав

19 июня 2023 г.

1 Градиентный спуск с постоянным шагом (learning rate)

Описание решения: Градиентный спуск с постоянным шагом - это метод оптимизации функции, который использует производную (градиент) функции для поиска минимума. Шаг, с которым мы двигаемся в направлении градиента, остается постоянным на протяжении всего процесса оптимизации.

В коде, который приведен ниже, мы начинаем с некоторой начальной точки x. Затем мы вычисляем градиент функции f в этой точке, используя метод $grad()$, и перемещаемся на некоторое расстояние в направлении антиградиента. Расстояние, на которое мы перемещаемся, определяется постоянным шагом α.

Мы затем проверяем, достигнут ли критерий остановки, который устанавливает, что норма градиента должна быть меньше некоторого значения ϵ. Если это условие не выполняется, мы повторяем процесс, начиная с новой точки y.

Градиентный спуск с постоянным шагом может быть эффективным методом оптимизации в тех случаях, когда функция имеет единственный минимум и этот минимум достигается в пределах области определения. Однако, если функция имеет множество локальных минимумов, градиентный спуск может "застрять"в одном из них.

```
while True:
    y = x -  alpha * f.grad(x)
    metr = get_metric2(f.grad(y) - f.grad(x))

    if metr < eps:
        break

    x = y
```

2 Метод одномерного поиска (метод дихотомии, метод Фибоначчи, метод золотого сечения) и градиентный спуск на его основе

Для решения данной задачи мы выбрали метод дихотомии

Для реализации метода дихотомии была написана функция $dichotomy()$, которая принимает на вход функцию , и начальные значения a и b. Цикл проходит 10 раз, этого должно хватать для нормальной точности α.

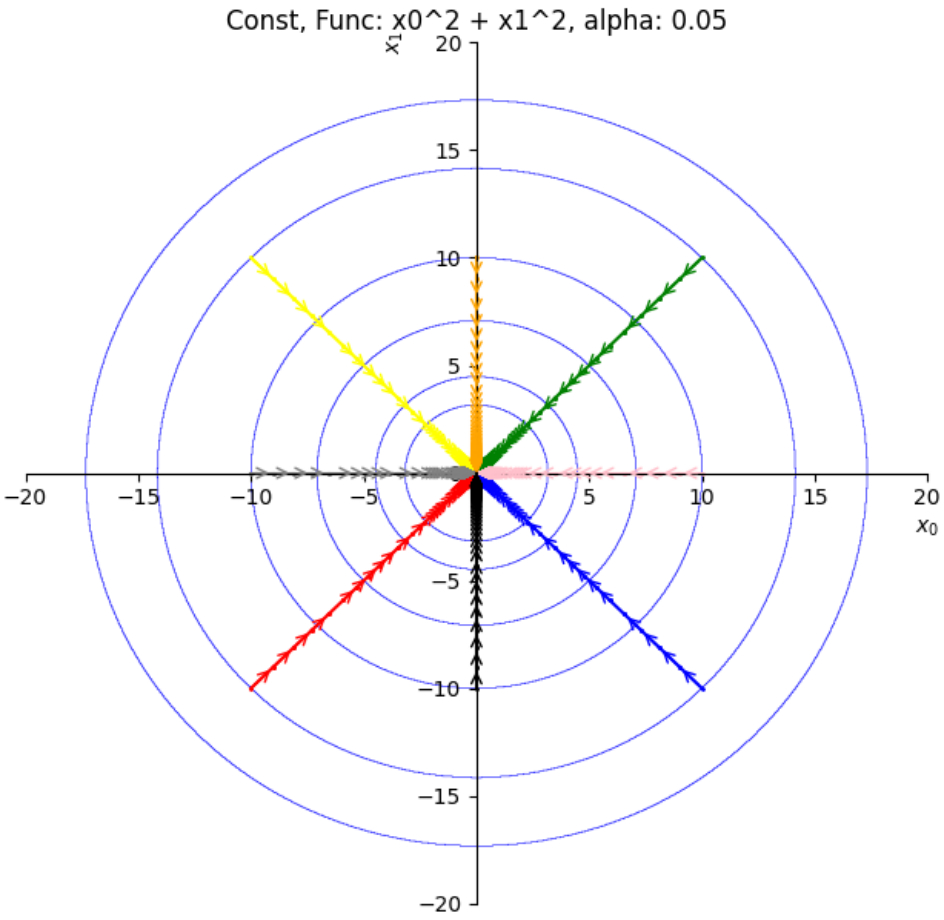
Для градиентного спуска была написана основная функция, в которой используется метод дихотомии для определения оптимального шага, на который стоит сместиться при поиске точки. Основной цикл повторяется до тех пор, пока норма градиента не станет меньше заданной точности ϵ.

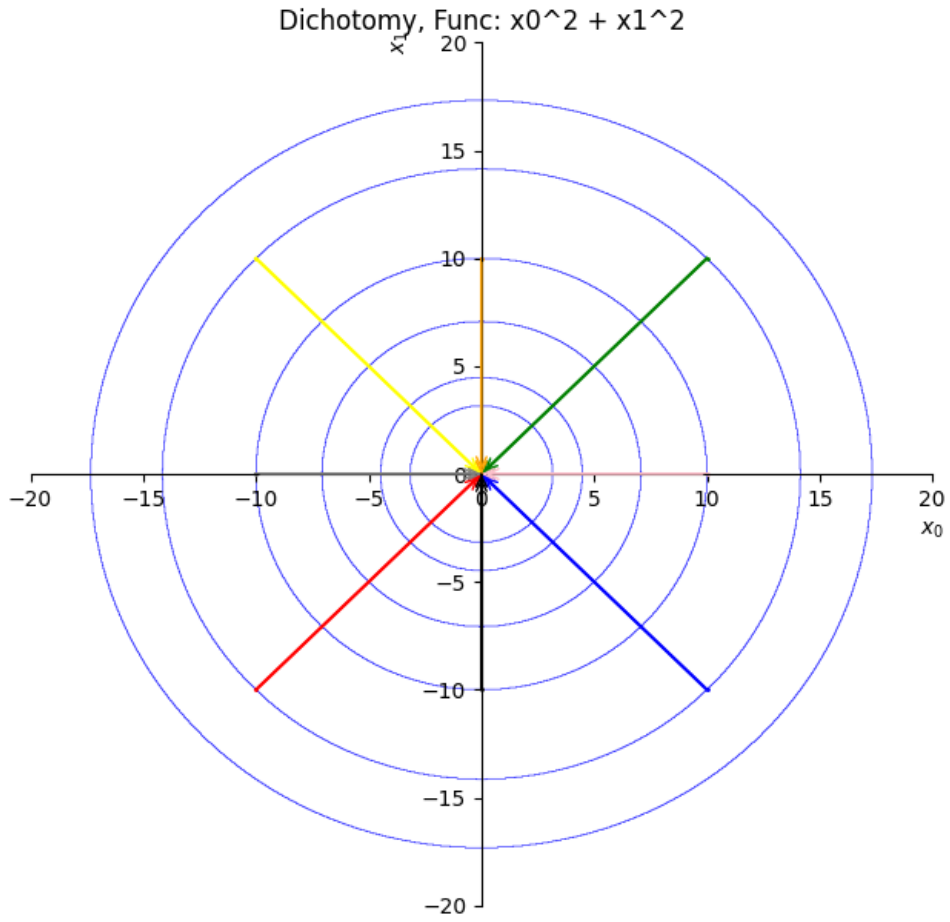
```
def dichotomy(f, a=0, b=10):
    for i in range(10):
        c = (a + b) / 2;
        if f(c) <= 0:
            b = c;
        else
            a = c;
    return (a + b) / 2


while True:
    alpha = dichotomy(lambda a: f.eval(x - a * f.grad(x)))

    y = x - alpha * f.grad(x)
    metr = get_metric2(f.grad(y) - f.grad(x))

    if metr < eps:
        break

    x = y
```

3 Траектория градиентного спуска на примере квадратичных функций

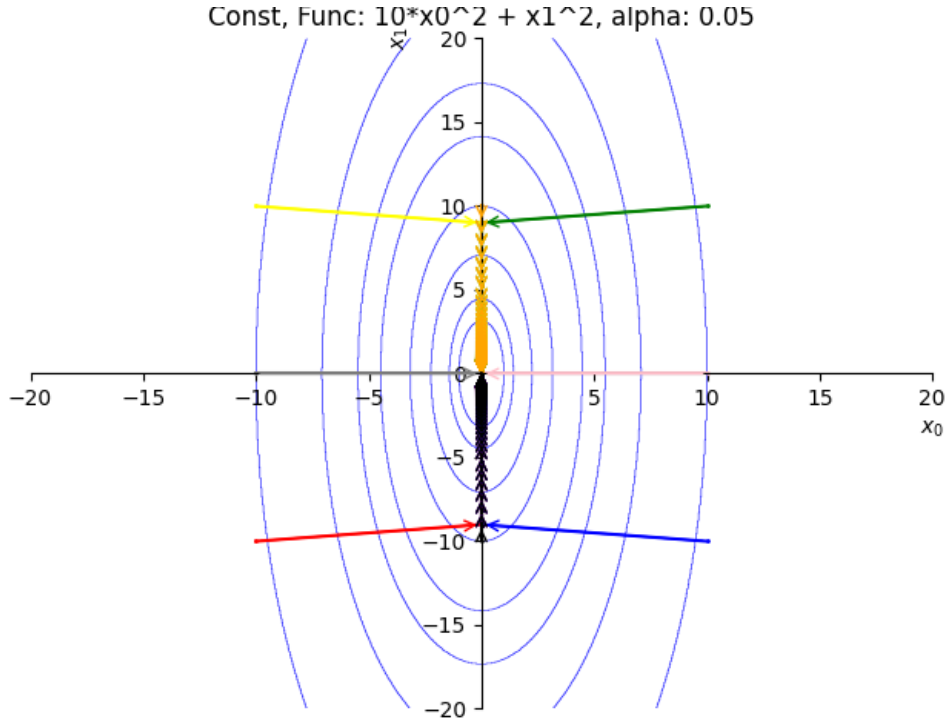
Выбранные функции:

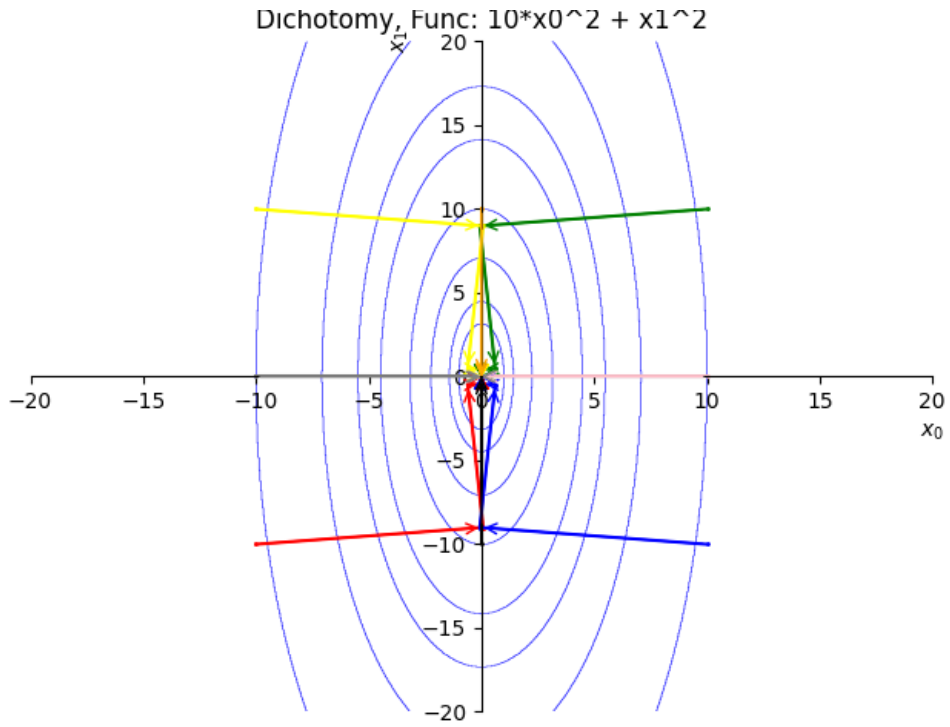
1. $x^2 + y^2 = 0$


Const, Func: x0^2 + x1^2, alpha: 0.05
x_1
x_0
20
15
10
5
−5
−10
−15
−20
−20
−15
−10
−5
5
10
15
20

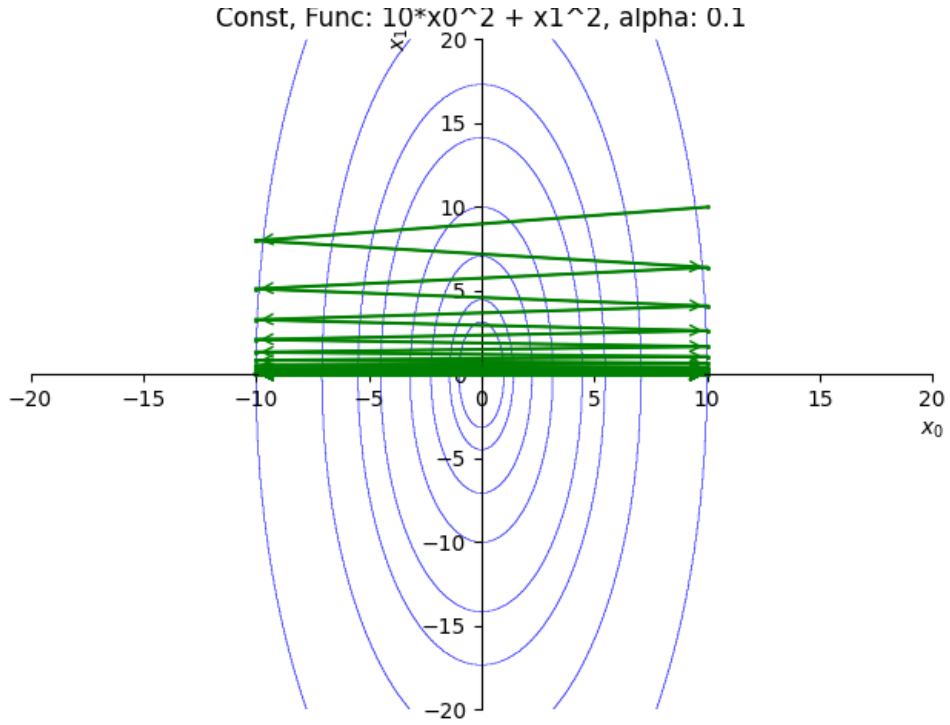

Dichotomy, Func: x0^2 + x1^2
x_1
x_0
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

2. $10x^2 + y^2 = 0$


Const, Func: 10*x0^2 + x1^2, alpha: 0.05
x_1
x_0
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

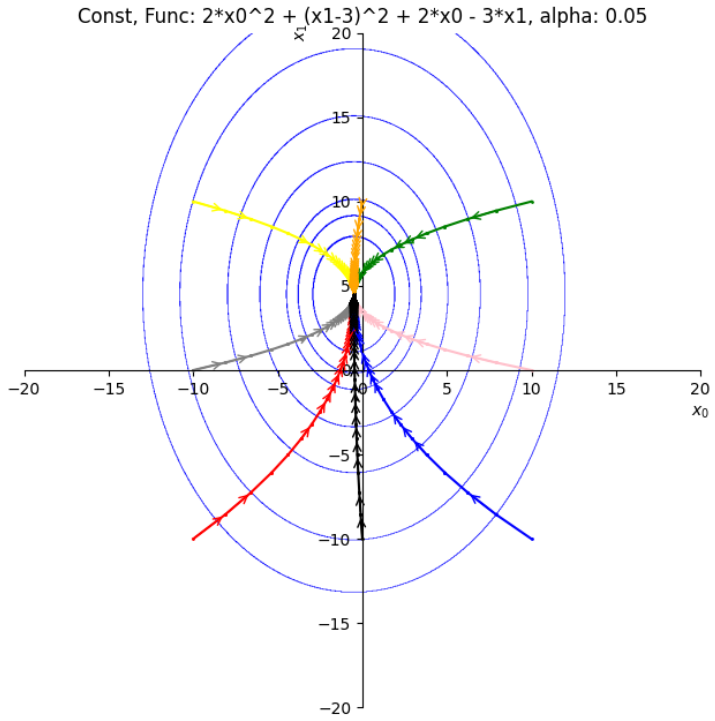


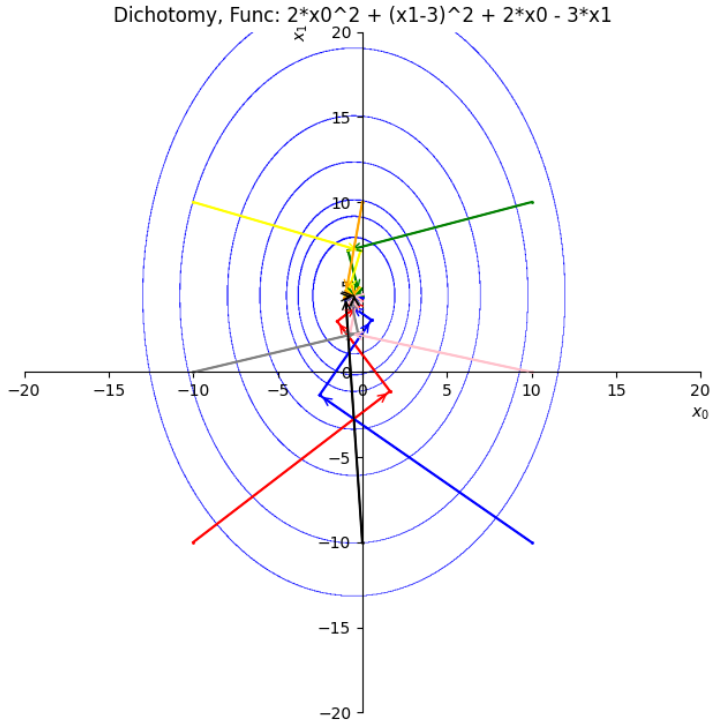
Что происходит, когда метод расходится



Из за большого шага, метод просто не может приблизиться к точке

3. $2 * x^2 + (y-3)^2 + 2 * x - 3 * y - 10 = 0$





Можно увидеть, что методы, при увеличении числа обусловленности k пути начинают сходиться к центральной линии.

Это лучше заметно, на этом примере рисунка путей градиентного спуска постоянного шага для функции $10x^2 + y^2 = 0$ и для последней функции.

4 Исследование методов градиентного спуска

4.1 Сходимость градиентного спуска с постоянным шагом

Для исследования возьмём функцию $20x^2 + y^2$, зафиксируем точку $(20, 20)$. Будем менять значение α и значение ϵ, что-бы узнать как они влияют на сходимость градиентного спуска, и как меняется кол-во итераций.

Таблица 1: $20x^2 + y^2$ $(20, 20)$

Значение α	**Значение ϵ**	**Количество итераций**
0.1	0.0001	Не сошелся
0.2	0.0001	Не сошелся
0.01	0.001	332
0.01	0.0001	446
0.02	0.0001	239
0.03	0.0001	165
0.04	0.0001	126
0.05	0.0001	Не сошелся
0.09	0.0001	Не сошелся
0.001	0.0001	3340
0.001	0.00001	4491
0.04	0.00001	154
0.04	0.0000000001	292
0.001	0.0000000001	10241

Можно заметить, что чем меньше значение α, тем больше требуется кол-во операций. В нашем случае α выступает константой для шага алгоритма. Если шаг слишком большой, то алгоритм начинает рассходиться из-за того, что не может подойти близко к точки минимума.

Так же можно заметить, что существует верхний порог α после которого алгоритм начинает рассходится. И наименьшее кол-во итераций при α рядом с этим порогом.

Значение ϵ не влияет на сходимость алгоритма. Это значение описывает точность найденной точки. И лишь увеличивает или уменьшает кол-во итераций.

Проверим тезисы для выбранных функций:

Таблица 2: $x^2 + y^2$ $(10, 10)$

Значение α	**Значение ϵ**	**Количество итераций**
0.9999999	0.0001	> 100000
0.9	0.0001	60
0.8	0.0001	27
0.5	0.0001	2
0.001	0.0001	3167
0.001	0.00000001	7768
0.5	0.00000001	2

Можно заметить, что порог α о котором говорилось выше прямопропорционален числу обусловленности.

Все тезисы подтвердились для выбранных функций. Взаимосвязь сходимости и начальной точки описано в 4.3-ем пункте этого исследования.

Таблица 3: $10x^2 + y^2$ $(10, 10)$

Значение α	**Значение ϵ**	**Количество итераций**
0.9	0.0001	Не сошелся
0.1	0.0001	Не сошелся
0.099999	0.0001	>100000
0.01	0.0001	412
0.02	0.0001	222
0.09	0.0001	69
0.001	0.0001	2994
0.001	0.0000001	6445
0.02	0.0000001	391

Таблица 4: $2 * x^2 + (y - 3)^2 + 2 * x - 3 * y - 10$ $(10, 10)$

Значение α	**Значение ϵ**	**Количество итераций**
0.9	0.0001	Не сошелся
0.1	0.0001	46
0.2	0.0001	22
0.4	0.0001	28
0.5	0.0001	Не сошелся
0.49	0.0001	3404
0.001	0.0001	2994
0.001	0.0000001	6146
0.02	0.0000001	376

4.2 Эффективности градиентного спуска с одномерным поиском

Сравним какой какое кол-во итераций нужно градиентным спускам. Для оценки эффективности будем считать кол-во вызывов вычисления градиента в точки $\sum$. Кол-во итераций будем обозначать N.

Для константного градиентного спуска $\sum = N_1 * c$

Для градиентного спуска с одномерны поиском $\sum = N_2 * c + N_2 * a$

В наших случаях $c = 3$. Для метода с одномерным поиском

$a = 10$

Для анализа возьмём результаты из пункта 4.1 . Мы знаем, что кол-во итераций для градиентного спуска с постоянным шагом зависит от выбранной α.

Таблица 5: Анализ эффективности методов градиентного спуска с постоянным шагом

Функция	α	N_1	$\sum$
$x^2 + y^2 = 0$	0.5	2	6
$x^2 + y^2 = 0$	0.9	60	180
$x^2 + y^2 = 0$	0.001	3167	9501
$10x^2 + y^2 = 0$	0.09	69	207
$10x^2 + y^2 = 0$	0.01	412	1236
$10x^2 + y^2 = 0$	0.001	2994	8982
$2 * x^2 + (y - 3)^2 + 2 * x - 3 * y - 10 = 0$	0.2	22	66
$2 * x^2 + (y - 3)^2 + 2 * x - 3 * y - 10 = 0$	0.001	2994	8982

Для следущих значений будем запускать градиентный спуск с одномерным поиском из точки,

которая была в пункте 4.1

Таблица 6: Анализ эффективности методов градиентного спуска с одномерным поиском

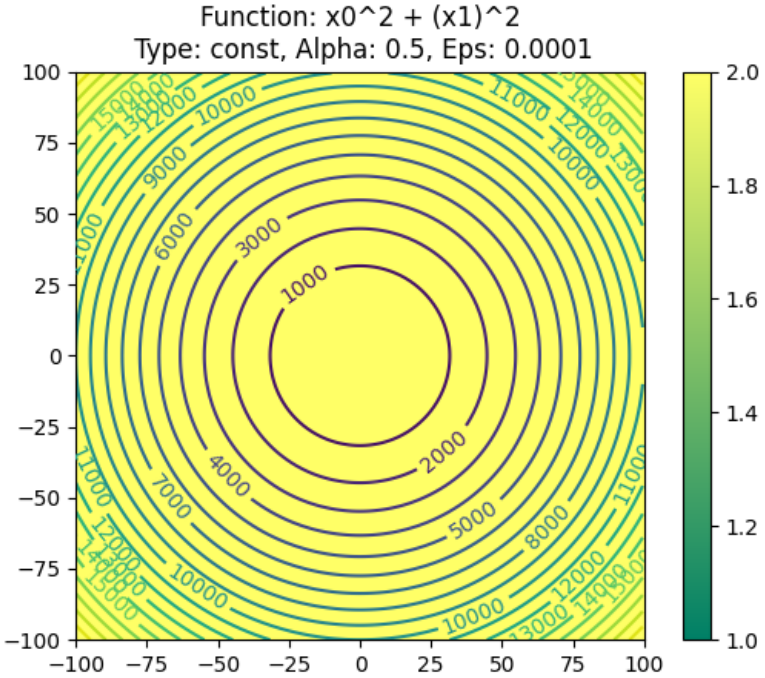
Функция	N_2	$\sum$	$\min \sum$
$x^2 + y^2 = 0$	3	39	6
$10x^2 + y^2 = 0$	12	126	207
$2 * x^2 + (y-3)^2 + 2 * x - 3 * y - 10 = 0$	9	117	66

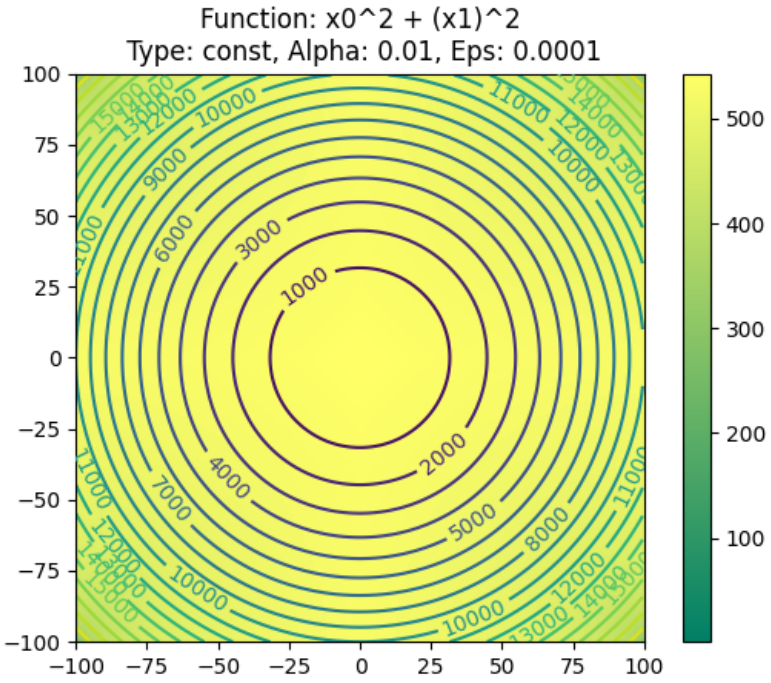
Можно увидеть, что $\sum$ для градиентного спуска с одномерным поиском близко к значению наилучшей константы α. Но если константа подобрана плохо, то эффективность заметна сразу.

4.3 Работа методов в зависимости от начальной точки

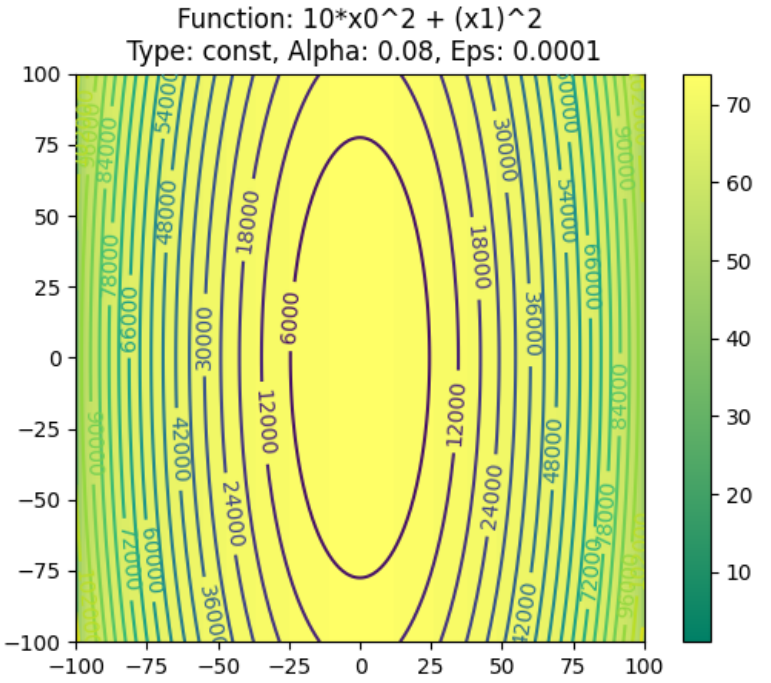
4.3.1 Метод градиентного спуска с постоянным шагом

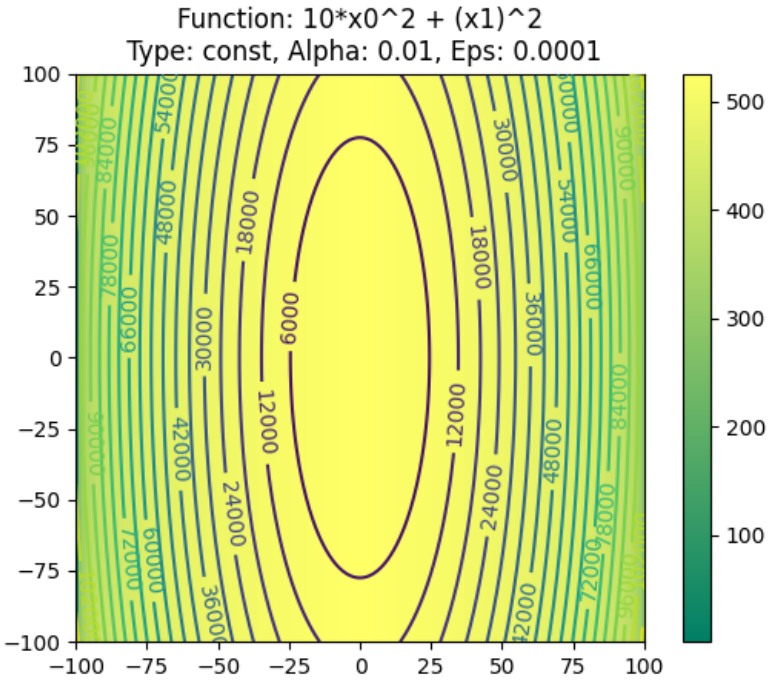
Рассмострим функции и разные α.



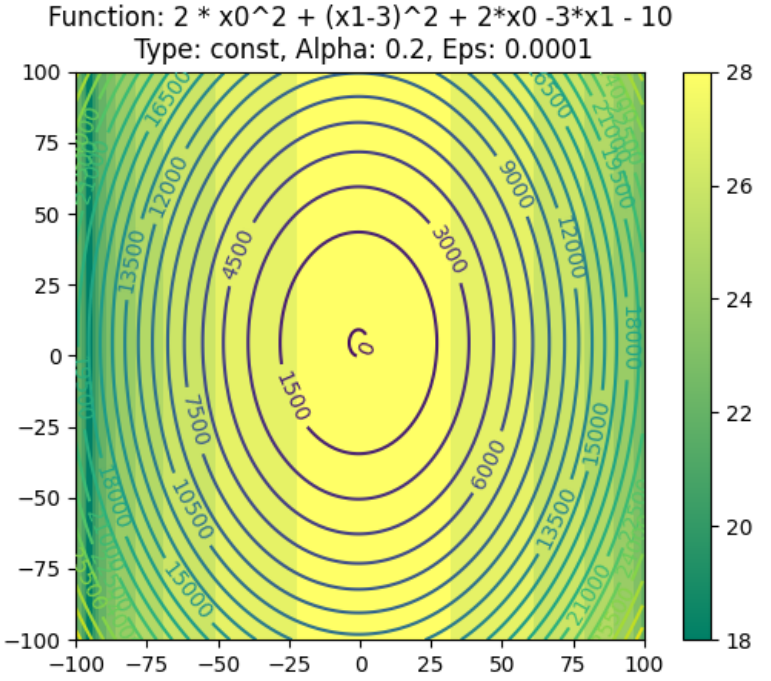


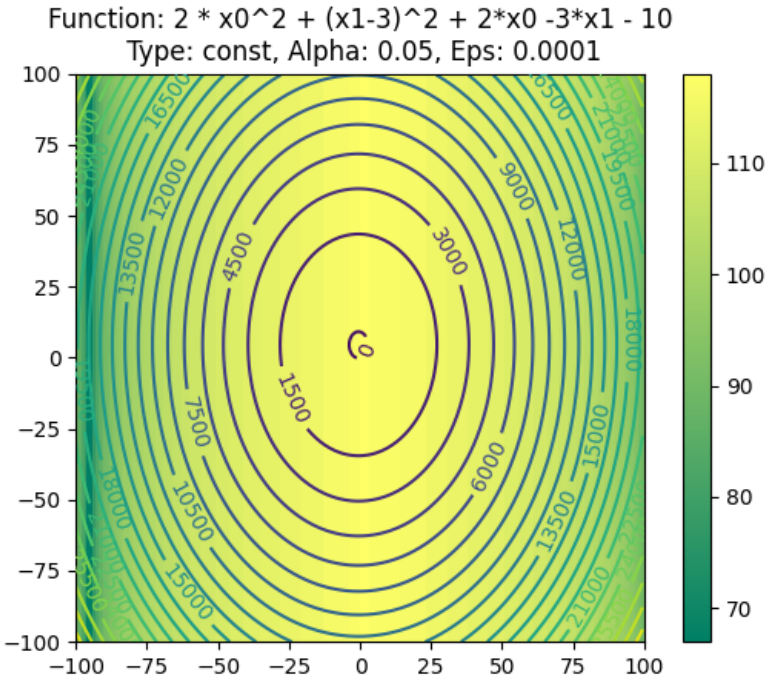
При уменьшении α видно, что чем дальше точка от минимума функции, тем плавно уменьшается кол-во итераций. Чем больше линии уровня, тем больше кол-во итераций.





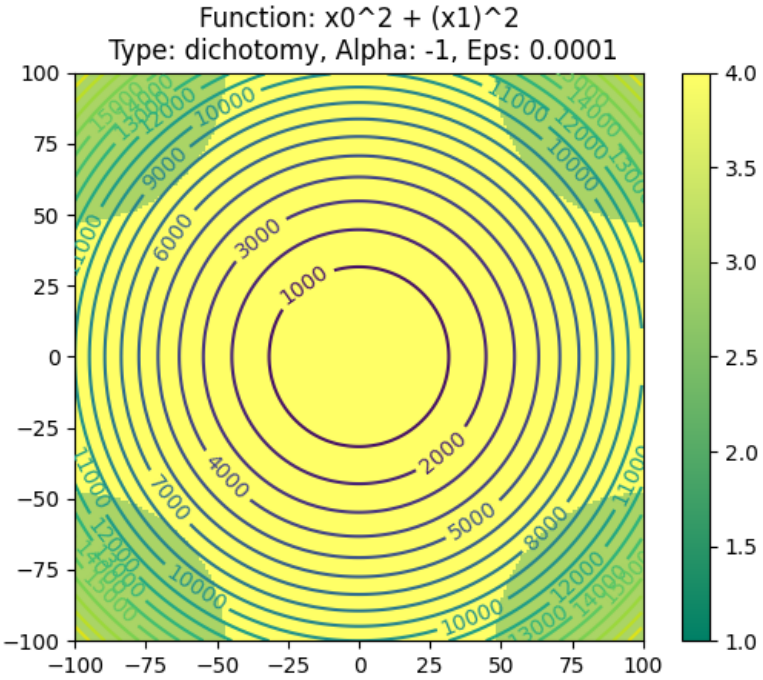
Видно, что при уменьшении α, увеличивается кол-во итераций не по радиусу, как у первой функции, а в первую очередь по краям, где вогнута функция.



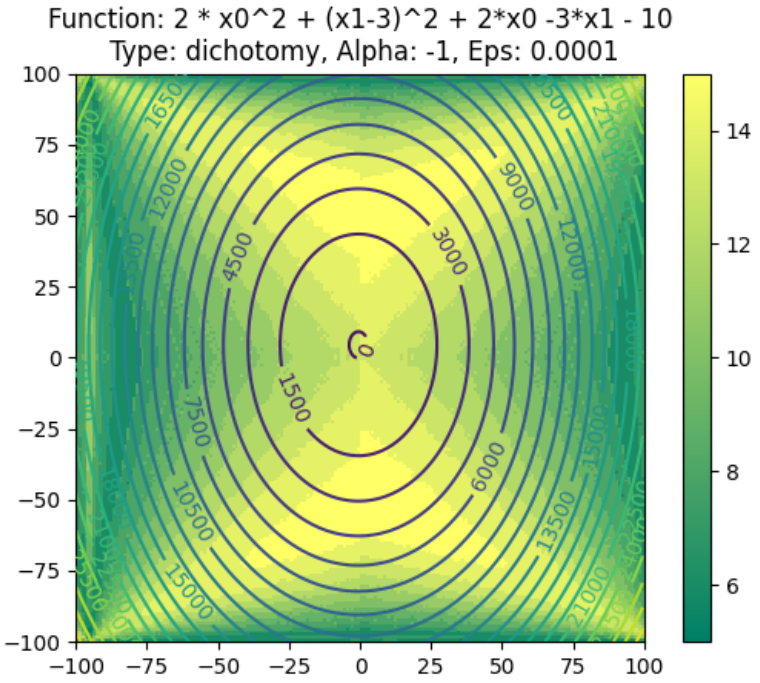


Для этой функции видно нагляднее, что области это прямоугольники. И чем лучше α тем явнее их видно. Так же видно, что существует наилучшая область, полоса, она находится справа.

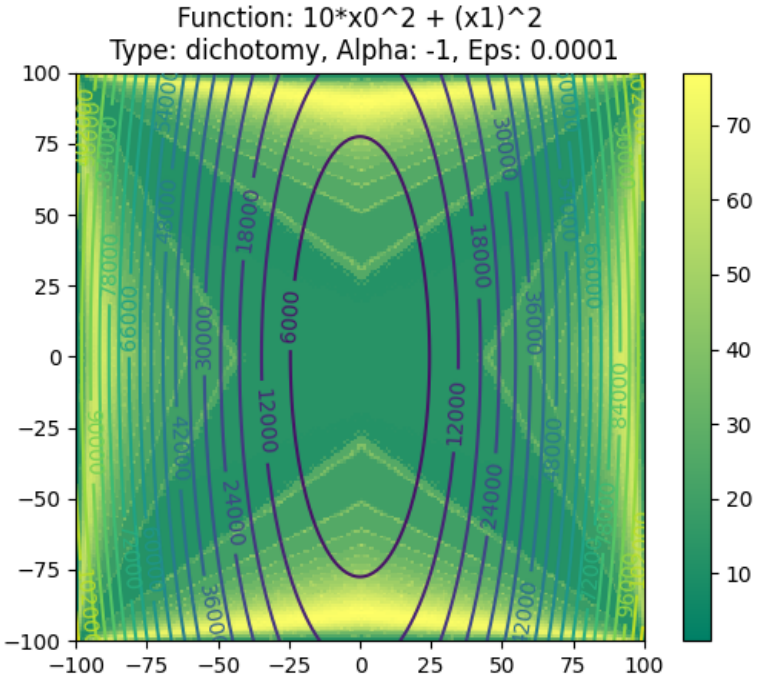
4.3.2 Метод градиентного спуска с одномерным поиском



При хорошей обусловленности можно увидеть, что по углам сущетсвуют окружности с меньшим кол-вом итераций.



При изменении числа обусловленности видно, что области становятся треугольниками.



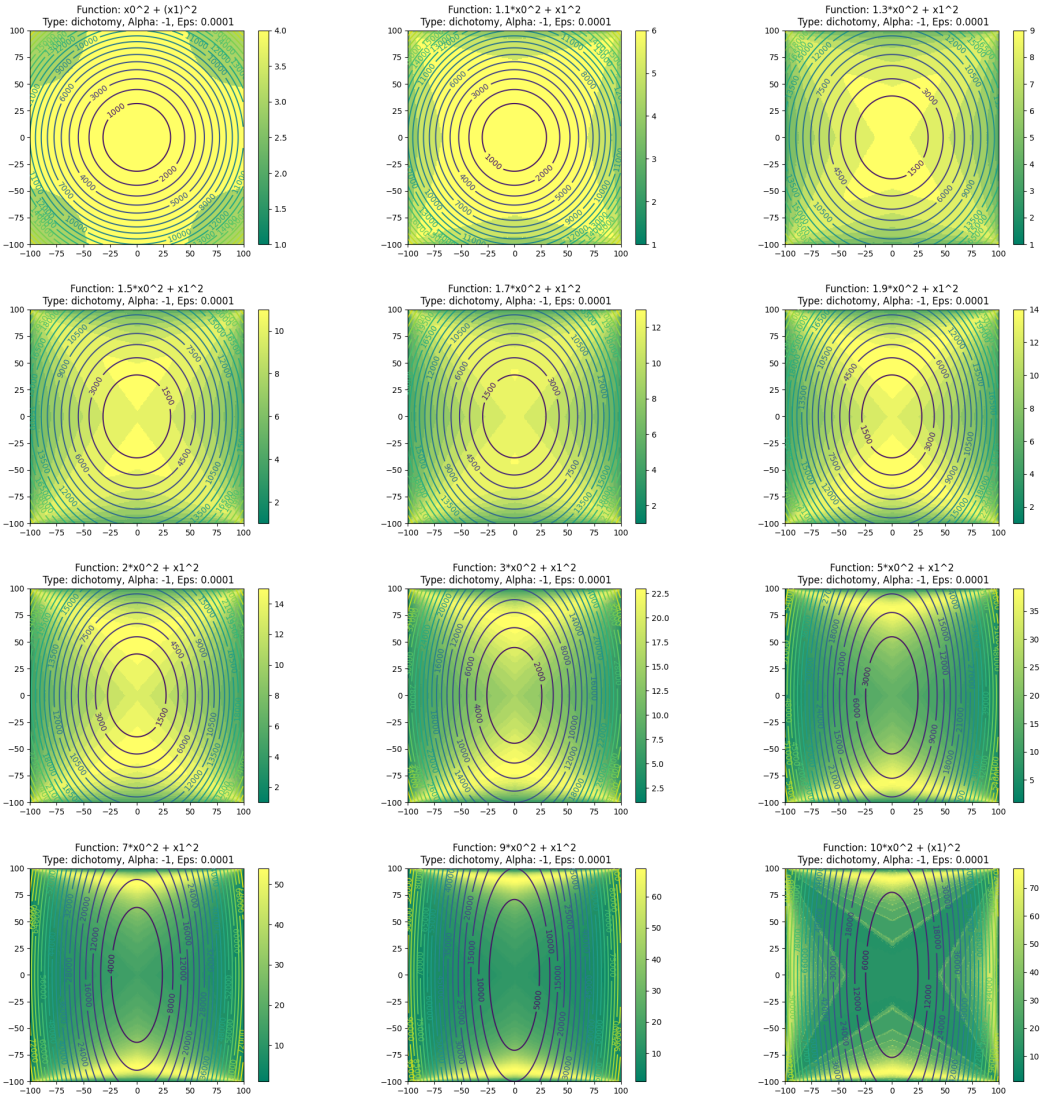
При увеличении числа обусловленности, видно, что области начинают инвертироваться.

4.4 Влияние нормализации - масштабирование осей для плохо обусловленной функции

Масштабирование осей плохо обусловленной функции, это понижениее её число обусловленности. Мы хотим сделать замену $y = 2y$, что бы растянуть какую то ось, для того, чтобы наши линии уровня стали больше похожи на круги, чем на овалы.

Для градиентного спуска с постоянным шагом, понятно из пункта 4.3, как происходит масштабирование. Появляются хорошо и плохо сходящиеся вертикальные или горизонтальные области.

Для метода с одномерным поиском не совсем понятно. Что происходит при масшатбировании осей. Давайте посмотрим на то, как меняется график при $k \subset [1, 10]$



Можно увидеть, что плохо обусловленная функция имеет минимальное количество итераций, намного меньше, чем минимальное среднее колличество итераций, для хорошо обусловленной функции. Так же можно увидеть, как области с максимальной и минимальной зоной менялись между собой.

5 Генератор случайных квадратичных функций

```
def fast_generate_quadratic_func(n: int, k: float) -> FastQFunc:
    """
    :param n: Count of vars
    :param k: Number of cond
    :return: Random QFunc
    """

    if k < 1:
```

```
        raise AssertionError("k must be >= 1")
    if n == 1:
        raise AssertionError("n must be > 1")

    if int(k) == 1:
        a_min = Decimal(1)
    else:
        a_min = Decimal(randint(1, int(k) - 1))
    a_max = Decimal(Decimal(k) * a_min)
    a_max, a_min = max(a_max, a_min), min(a_max, a_min)
    v = [float(a_min)] + [uniform(float(a_min), float(a_max)) for _ in range(n - 2)] + [
    A = np.diag(v)

    C = np.matrix(np.random.randint(1, INF, (n, n)))
    Q, R = np.linalg.qr(C)

    B = np.matmul(np.matmul(Q, A), Q.transpose())

    return FastQFunc(n, B, np.matrix([0 for _ in range(n)]), 5)
```

Первым делом мы генерируем диагональную матрицу A с любыми собственными значениями. Главное чтобы выполнялось условие $\max \alpha / \min \alpha = k$.

Матрица A уже задаёт квадратичную функцию. Но она не будет иметь поворотов и смещений.

Для добавления поворотов и смещений мы можем изменить базис A. Для этого мы можем воспользоваться ортоганальным преобразованием. $B = Q * A * Q^T$ Оно сохраняет инварианты. А собственные числа это и есть инварианты. Значит число обусловленности останется тем же самым.

Для ортоганального преобразования нам нужна рандомная ортоганальная матрица Q. Для её нахождения мы генерируем рандомную матрицу C. И получаем матрицу Q с помощью qr-разложения.

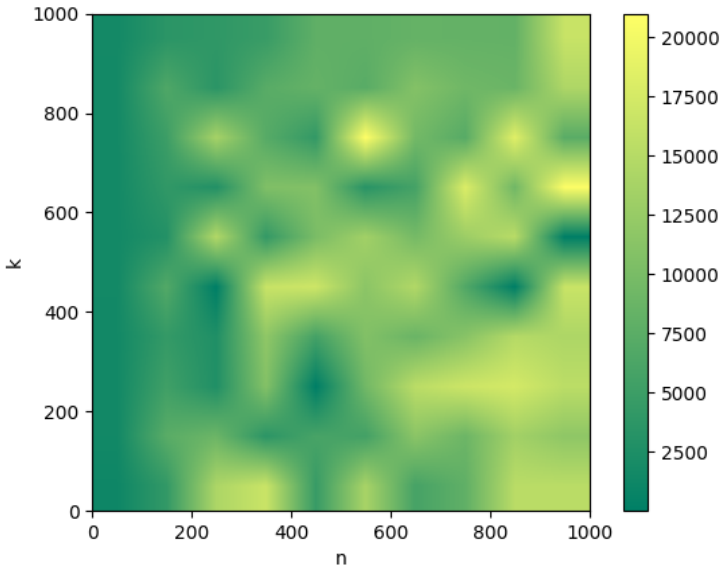
6 Исследуйте зависимость числа итераций $T(n, k)$ необходимых градиентному спуску для сходимости в зависимости от размерности пространства $2 \leq n \leq 10^3$ и числа обусловленности оптимизируемой функции $1 \leq k \leq 10^3$

6.1 Градиентный спуск с постоянным шагом

Как мы выяснили из 3-его пункта, сходимость функции связано со значение α. Поэтому для исследования, мы решили взять разную α для разного числа обусловленности k.

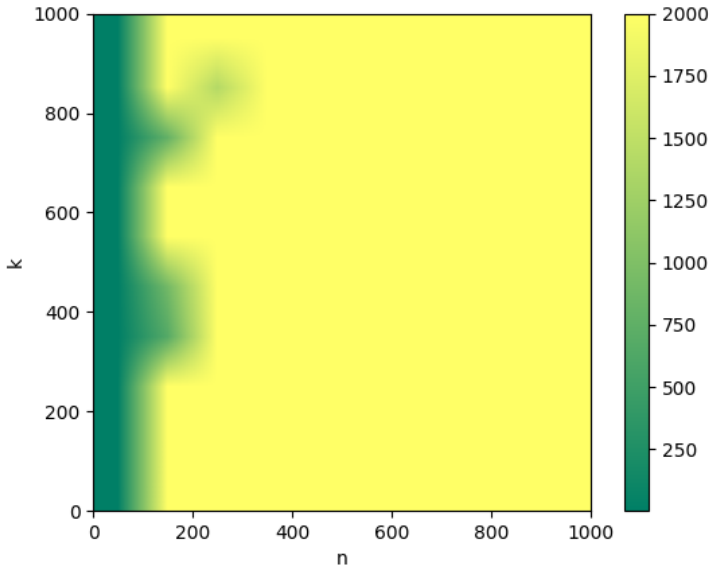
Таблица 7: Значение α

k	a	α
$<= 100$	1	1 / (80 * a * k + 1)
< 100	1.4	1 / (80 * a * k + 1)
> 200	2	1 / (80 * a * k + 1)
> 300	3	1 / (80 * a * k + 1)
> 500	3.3	1 / (80 * a * k + 1)
> 600	4	1 / (80 * a * k + 1)
> 800	5	1 / (80 * a * k + 1)
> 900	6	1 / (80 * a * k + 1)



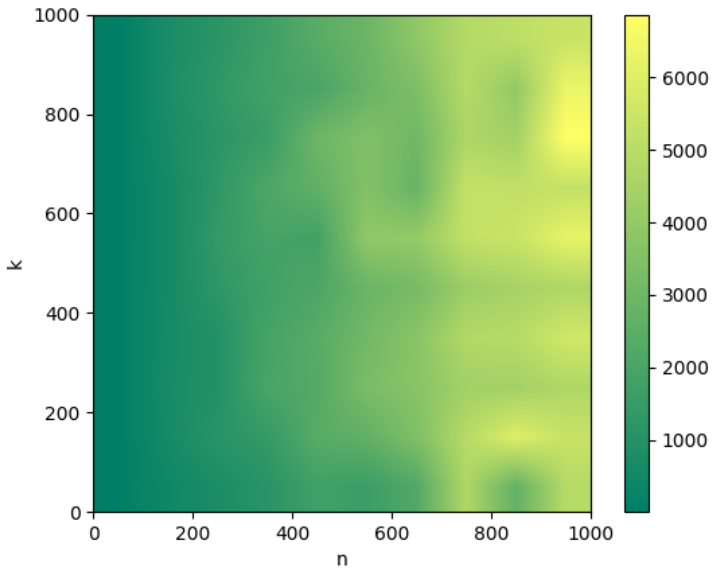
Максимальное кол-во итераций установлено 2000. Можно увидить, что с некоторыми функциями есть плохое схождение, большое кол-во итераций. Скорее всего это связано с плохой начальной точкой. Которая бралась рандомно в пределах $1000 : -1000$.

6.2 Градиентный спуск с одномерным поиском



Можно увидеть, что для больших кол-ва обусловленностей метод перестаёт сходиться. Это говорит о том, что плохо подобрано кол-во итераций для поиска α. И α находиться плохо В 7 пункте попробуем увеличить это значение.

6.3 Градиентный спуск с методом Вольфе

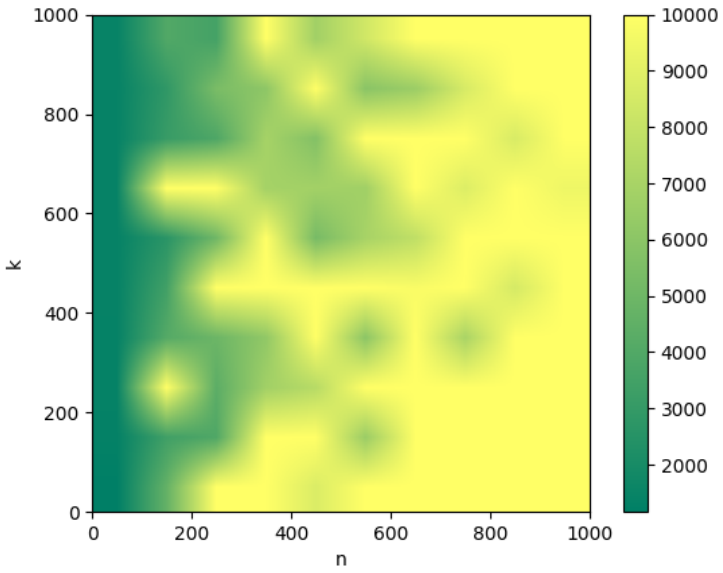


Этот метод не требует нахождение кол-ва для итераций α и этот метод сходиться намного лучше и равномерней, чем градиентный спуск с постоянным шагом и градиентный спуск с одномерным поиском.

7 Для получения более корректных результатов проведите множественный эксперимент и усредните полученные значения числа итераций

7.1 Градиентный спуск с постоянным шагом

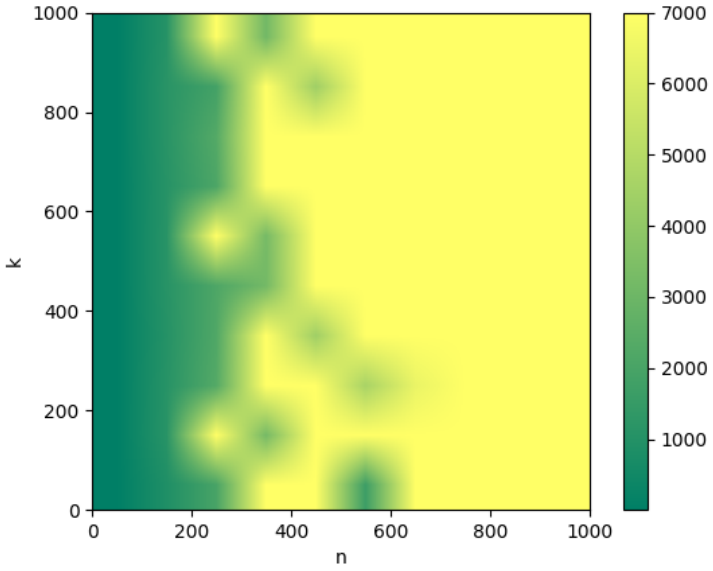
Для этого метода мы оставим теже α, что и в 6ом пункте. Но уменьшим максимальное кол-во операций до 10^4.



Видим примерно туже картину. Сходимость функции очень сильно зависит от начальной точки и значения α.

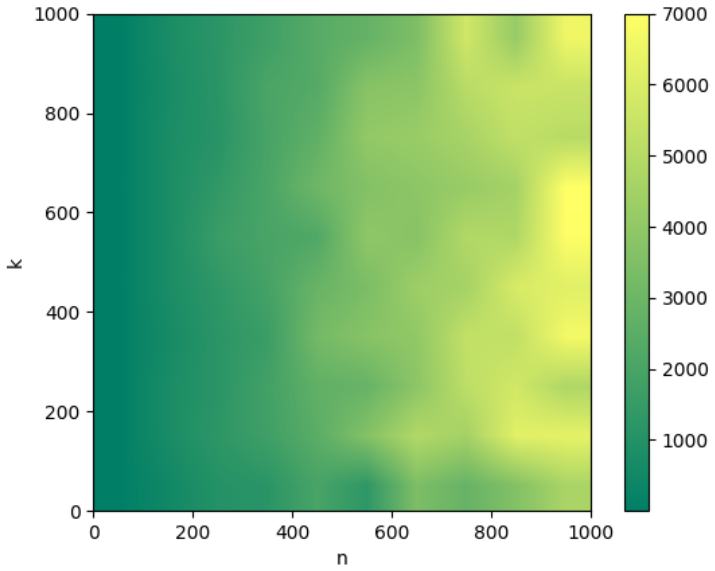
7.2 Градиентный спуск с одномерный поиском

Для этого метода мы увеличили кол-во итераций для нахождения значения α.



Тут картина уже лучше. Но такой метод требует большего времени и большего кол-ва итераций. А это в свою очередь сильно увеличивает время.

7.3 Градиентный спуск с методом Вольфе



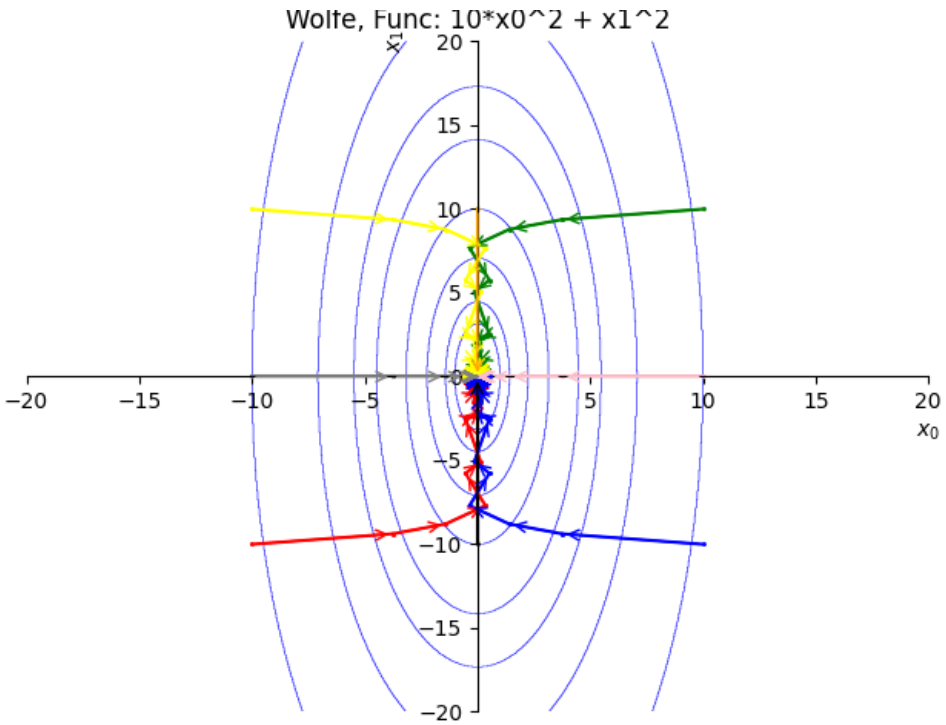
Метод Вольфе все так же стабильно сходиться с приемлемым кол-вом итераций.

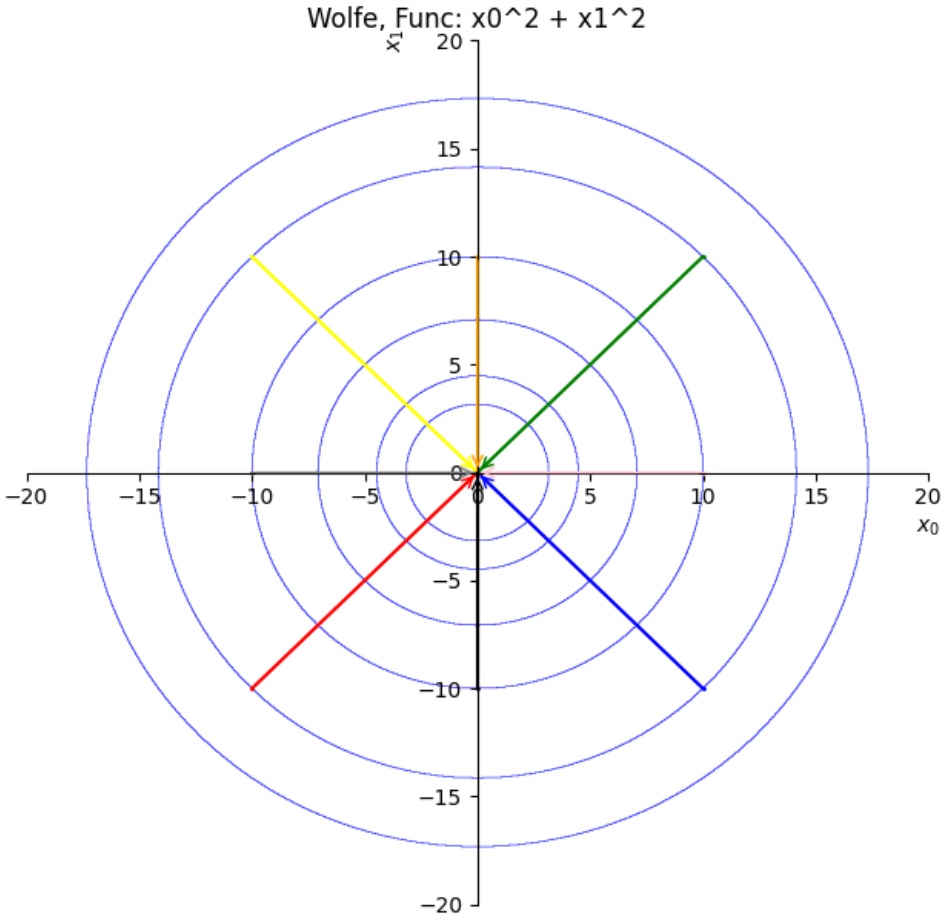
8 Метод Вольфе

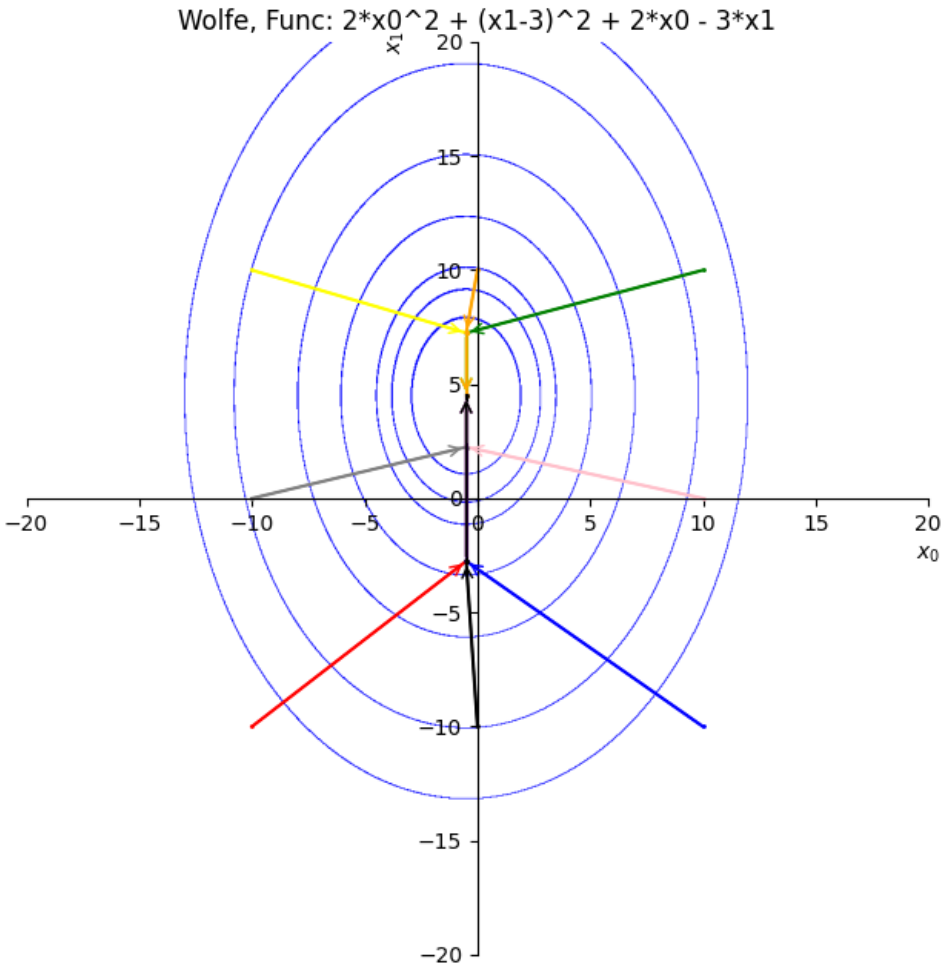
метод вольфе представляет собой более простой и быстрый выбор коэффициента в функции $wolfe\_condition$ показаны два условия Вольфе, если они не выполняются то α в функции $find\_alpha\_with\_wolfe$ уменьшается вдвое, по итогу приходим к α которая удовлетворяет уловию вольфе и находится за log

```
def wolfe_conditions(f: Func, x: sp.Matrix, alpha: float, c1=0.01, c2=0.99):
    grad_fx = f.grad(x)
    p = f.grad(x)
    fx_alpha = f(x - alpha * p)
    grad_fx_alpha = f.grad(x - alpha * p)
    cond1 = fx_alpha <= f(x) + c1 * alpha * (grad_fx * p.transpose())[0]
    cond2 = (grad_fx_alpha * p.transpose())[0] >= c2 * (grad_fx_alpha * p.transpose())[0

    return cond1 and cond2


def find_alpha_with_wolfe(f: Func,
                          start_point: sp.Matrix,
                          c1=0.001, c2=0.99):
    alpha = 1
    while not wolfe_conditions(f, start_point, alpha, c1=c1, c2=c2):
        alpha *= 0.5
    return alpha

while True:
    y = x - alpha * f.grad(x)
    metr = get_metric2(f.grad(y) - f.grad(x))

    if metr < eps:
        break

    x = y
```

Траектория:


Wolfe, Func: 10*x0^2 + x1^2
x_1
x_0
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
5
10
15
20


Wolfe, Func: x0^2 + x1^2
x_1
x_0
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20



как видно по графикам из 7го пункта градиентный спуск с использованием условий Вольфе, гораздо стабильнее и быстрее находит ответ, нежели ГС с постоянным шагом и с одномерным поиском, из-за простоты условий и рационального выбора α

9 Вывод

Нами была проделана работа по сравнению градиентного спуска с постоянным шагом, с использованием одномерного поиска на основе метода дихотомии и с учетом условий Вольфе. По полученным результатам, можно сделать вывод, что метод одномерного поиска увеличивает множество точек, на которых градиентный спуск успешно находит минимум, однако он может уступать в эффективности постоянному шагу. Также, на плохо обусловленных функциях использование условий Вольфе улучшают эффективность работы градиентного спуска.

10 Ссылки

https://github.com/mgvts/methodsOptimizationLab1