

Лабораторная Работа №2

Методы первого и высших порядков.

Колтаков Максим М3234

Рязанова Екатерина М3234

Хайруллин Артур М3234

Оглавление

[Метод Ньютона с постоянным шагом.](#)

[Метод Ньютона с одномерным поиском](#)

[Метод Newton-CG и один-два квазиньютоновских метода](#)

[Сравнения методов](#)

[Дополнительное Задание 1.](#)

[Дополнительное Задание 2.](#)

[Ссылка на гит-репо с кодом.](#)

Метод Ньютона с постоянным шагом.

Описание метода:

1. Инициализируется начальная точка cur_x и градиент cur_grad.
2. Выполняется цикл, пока не будет достигнута заданная точность eps или не будет достигнуто максимальное количество итераций 1000.
3. В каждой итерации вычисляется Гессиан в текущей точке, который затем делается положительно определенным с помощью функции positive_matrix.
4. Вычисляется направление поиска, используя обратную матрицу Гессиана и градиент.
5. Обновляется текущая точка, вычитая от нее произведение направления на постоянный шаг.
6. Процесс повторяется до тех пор, пока не будет достигнута заданная точность или не будет достигнуто максимальное количество итераций.

Выберем функцию: функция Розенброка

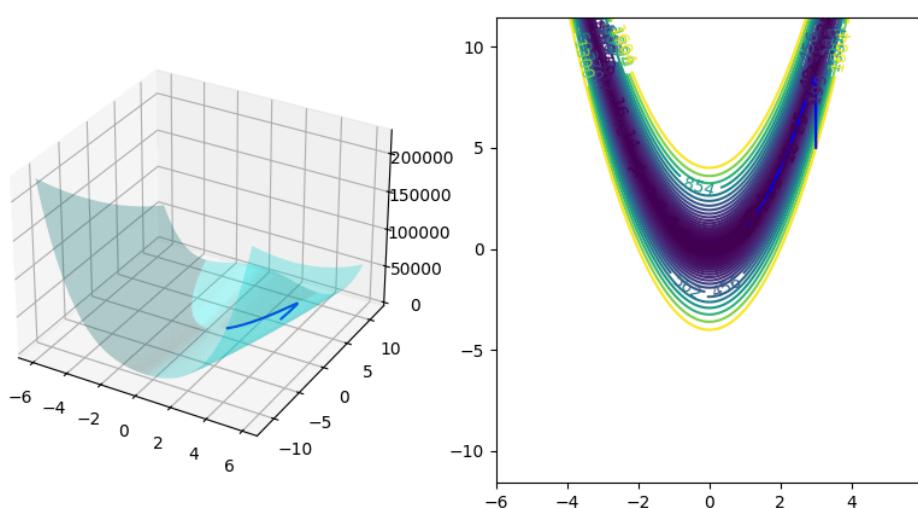
$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \text{ минимум в точке } (1, 1)$$

```

Method: Newton with constant step
Start point: [3 5]
x: [1. 1.]
y: 6.987510298775049e-19
time: 0.04356980323791504
memory: 41121
function_calls: 0
gradient_calls: 298
hessian_calls: 298
iterations: 298

```

Newton with constant step для функции Rosenbrock, epsilon = 1e-08, начальная точка: [3 5]



Выберем функцию: $f(x, y) = 5xye^{(-x^2-y^2)}$

Local minima
Approximate form

$$\min\{5xye^{-x^2-y^2}\} = -\frac{5}{2e} \text{ at } (x, y) = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$$

$$\min\{5xye^{-x^2-y^2}\} = -\frac{5}{2e} \text{ at } (x, y) = \left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right)$$

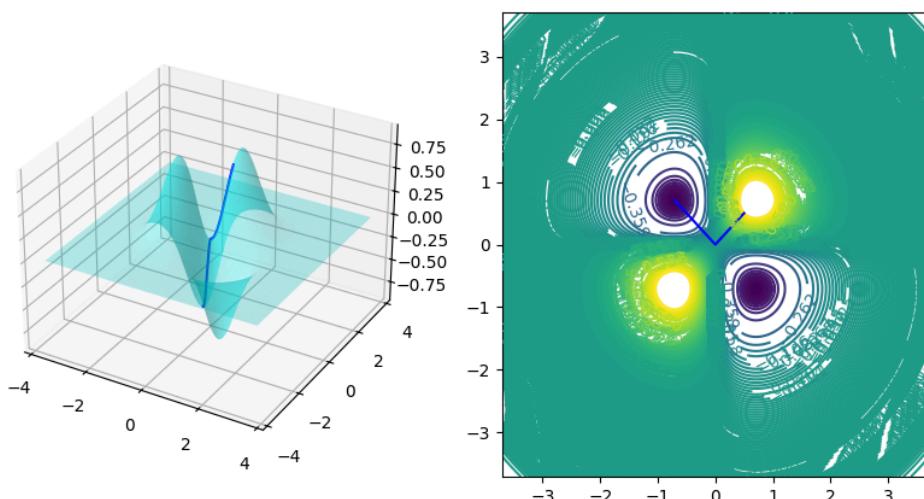
Enlarge
 Data
 Customize
 Plain Text

```

Method: Newton with constant step
Start point: [0.5 0.5]
x: [-0.70710678  0.70710678]
y: -0.9196986029286057
time:          3.748478412628174
memory:        4122588
function_calls: 0
gradient_calls: 572
hessian_calls: 572
iterations:    572

```

Newton with constant step для функции notpolinom, epsilon = 1e-08, начальная точка: [0.5 0.5]



Метод Ньютона с одномерным поиском

Описание метода:

1. Инициализируется начальная точка `cur_x` и градиент `cur_grad`.
2. Выполняется цикл, пока не будет достигнута заданная точность `eps` или не будет достигнуто максимальное количество итераций 1000.
3. В каждой итерации вычисляется Гессиан в текущей точке, который затем приводится к положительному определению с помощью функции `positive_matrix`.

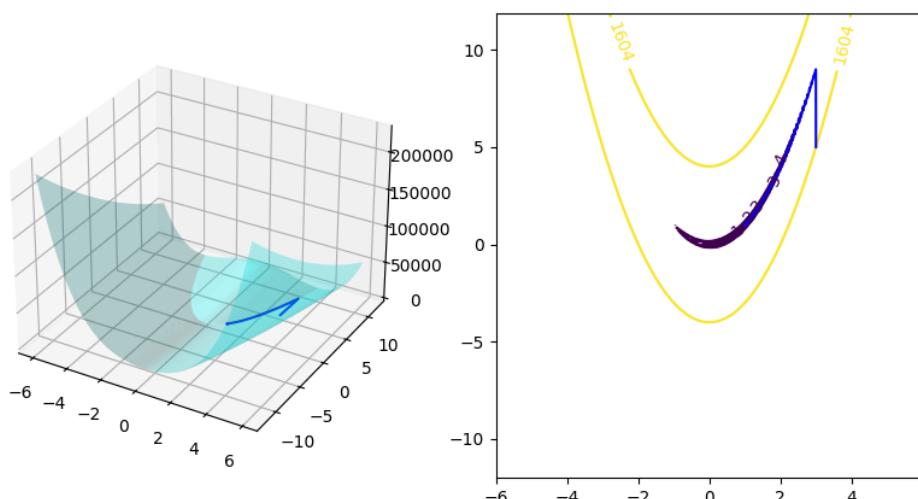
4. Вычисляется направление поиска, используя обратную матрицу Гессиана и градиент.
5. Обновляется текущая точка, вычитая от нее произведение направления на *шаг*, значение которого считается по методу золотого сечения.
6. Процесс повторяется до тех пор, пока не будет достигнута заданная точность или не будет достигнуто максимальное количество итераций.

Выберем функцию: функция Розенброка

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

```
Method: Newton
Start point: [3 5]
x: [1. 1.]
y: 2.2137409152764644e-29
time: 0.013134241104125977
memory: 6356
function_calls: 602
gradient_calls: 14
hessian_calls: 14
iterations: 14
```

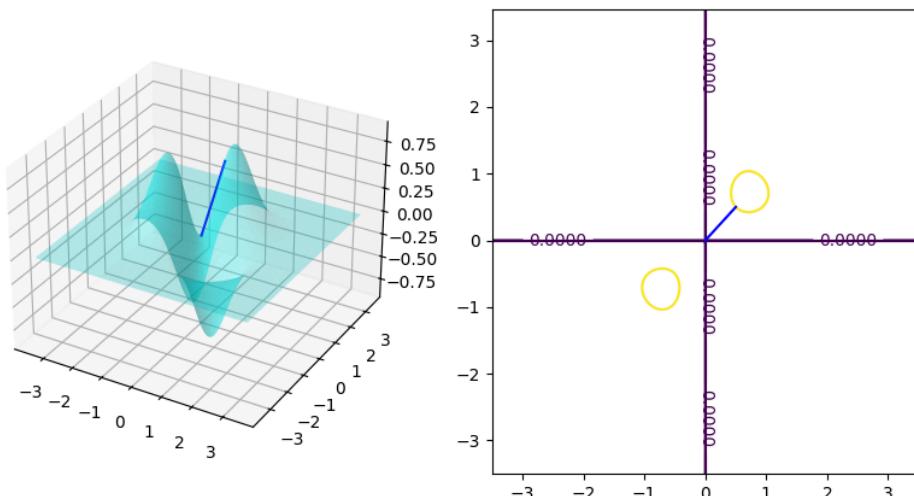
Newton для функции Rosenbrock, epsilon = 1e-08, начальная точка: [3 5]



Выберем функцию: $f(x, y) = 5xye^{(-x^2-y^2)}$

```
Method: Newton
Start point: [0.5 0.5]
x: [-1.21547800e-15 1.44905728e-15]
y: -8.806486172676584e-30
time: 0.02695918083190918
memory: 73826
function_calls: 127
gradient_calls: 3
hessian_calls: 3
iterations: 3
```

Newton для функции notpolinom, epsilon = 1e-08, начальная точка: [0.5 0.5]



Метод Newton-CG и один-два квазиньютоновских метода

Описание метода:

Newton-CG: метод Ньютона с использованием метода сопряжённых градиентов (conjugate gradients). На каждой итерации для получения направления шага решается система линейных уравнений, которая получается из гессиана функции и градиента функции. Использование метода сопряженных градиентов дополнительно способствует эффективному решению этой системы

уравнений, что обеспечивает быструю сходимость к оптимуму.

BFGS: алгоритм Бройдена — Флетчера — Гольдфарба — Шанно нахождения локального оптимума. Это квазиньютоновский метод: обратная матрица Гессе не вычисляется в явном виде, вместо этого вычисляется матрица - её приближение.

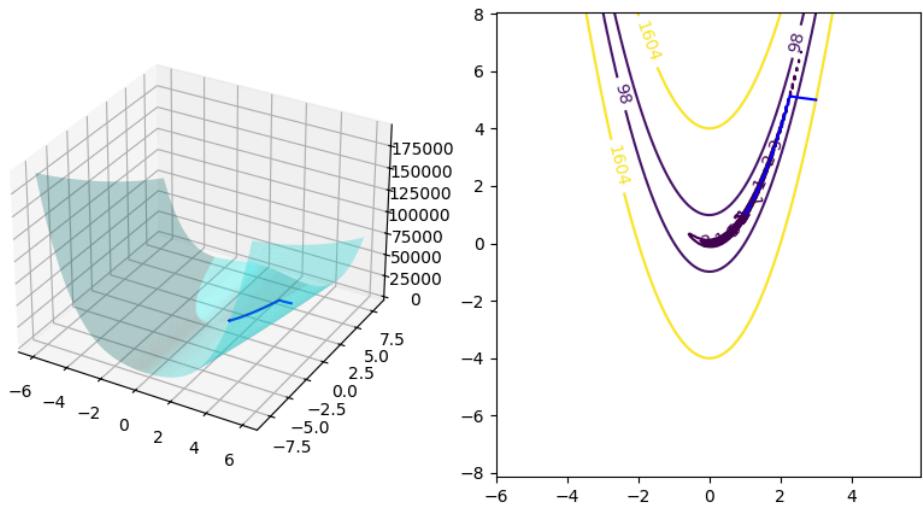
L-BGFS: вариация BFGS, использующая меньше памяти за счёт того, что хранит только несколько последних итераций метода и хранит не полную матрицу приближённого обратного гессиана (как в BFGS), а вектор, представляющий произведение обратного гессиана на градиент. В программе использован метод L-BFGS-B, который позволяет решать задачу с ограничениями на переменные (ограничения не применялись).

Выберем функцию: функция Розенброка

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

```
Method: Newton-CG
Start point: [3 5]
x: [0.9999999 0.9999998]
y: 9.929182434526365e-17
time: 0.025303125381469727
memory: 45497
function_calls: 72
gradient_calls: 163
hessian_calls: 0
iterations: 43
```

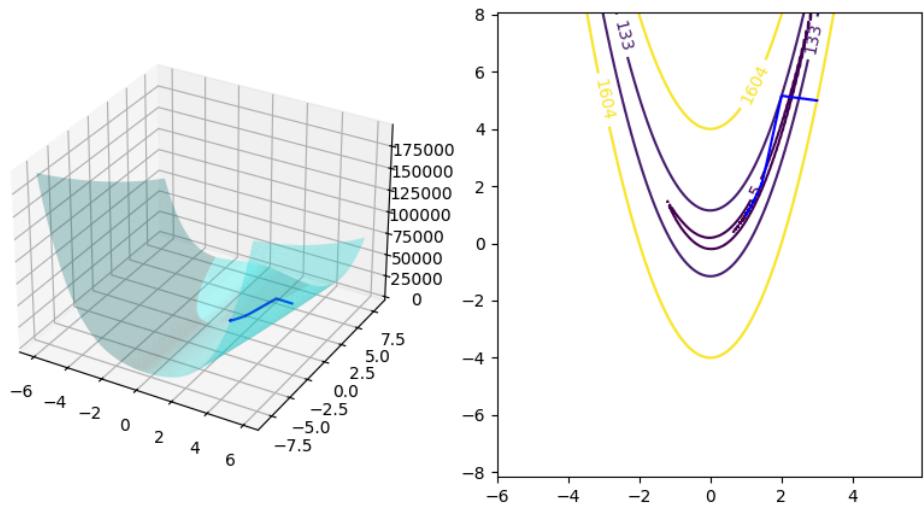
Newton-CG для функции Rosenbrock, epsilon = 1e-08, начальная точка: [3 5]



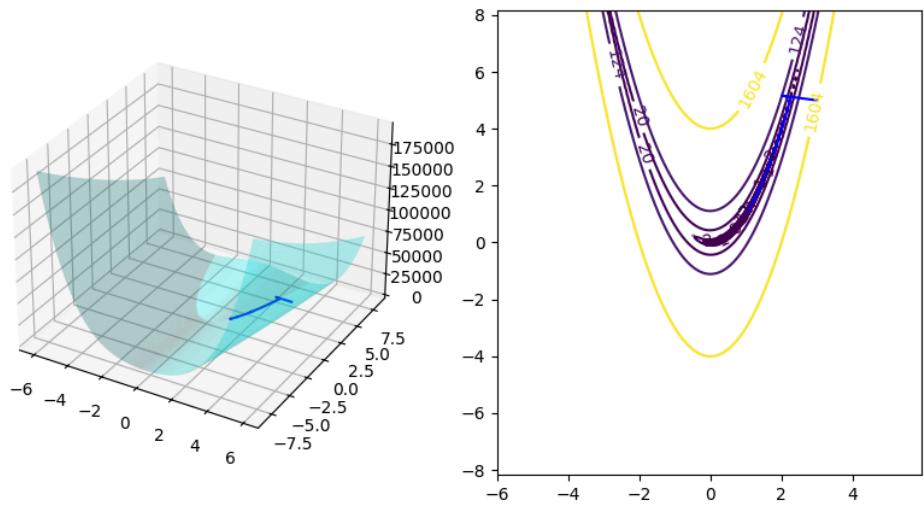
```
Method: Quasinewton (BGFS)
Start point: [3 5]
x: [1. 1.]
y: 7.274519186119575e-22
time: 0.009336709976196289
memory: 12999
function_calls: 34
gradient_calls: 34
hessian_calls: 0
iterations: 24

Method: Quasinewton (L_BFGS-B)
Start point: [3 5]
x: [1.00000384 1.00000879]
y: 1.378622430641048e-10
time: 0.0060045719146728516
memory: 27120
function_calls: 38
gradient_calls: 38
hessian_calls: 0
iterations: 34
```

Quasinewton (BGFS) для функции Rosenbrock, epsilon = 1e-08, начальная точка: [3 5]



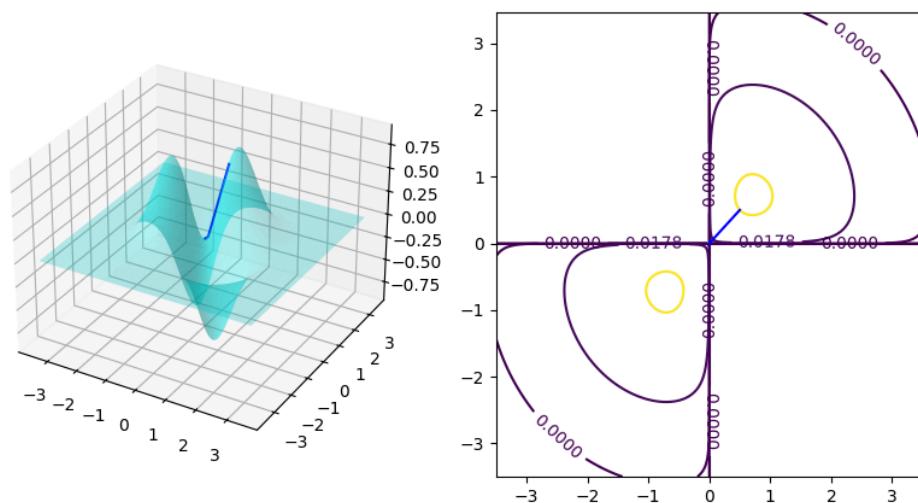
Quasinewton (L_BFGS-B) для функции Rosenbrock, epsilon = 1e-08, начальная точка: [3 5]



Выберем функцию: $f(x, y) = 5xye^{(-x^2-y^2)}$

```
Method: Newton-CG
Start point: [0.5 0.5]
x: [-4.50957032e-18 -4.50957032e-18]
y: 1.0168112253358593e-34
time: 0.004000663757324219
memory: 33193
function_calls: 8
gradient_calls: 13
hessian_calls: 0
iterations: 5
```

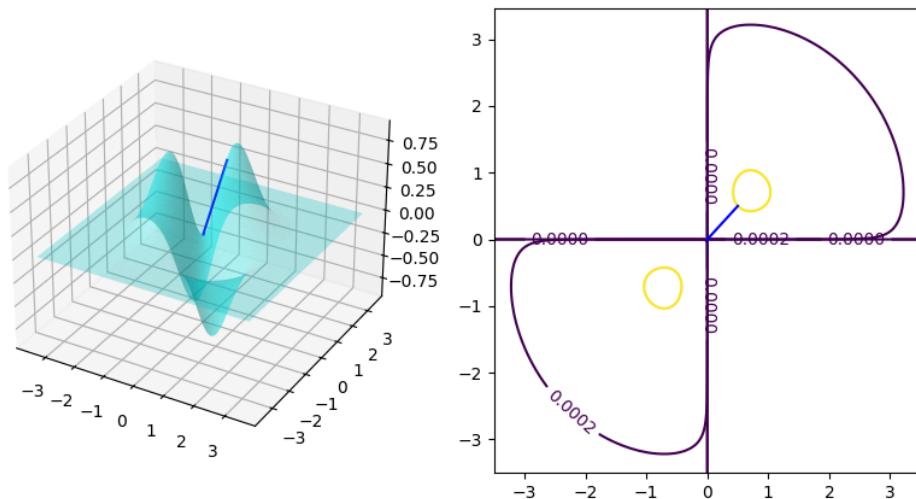
Newton-CG для функции notpolinom, epsilon = 1e-08, начальная точка: [0.5 0.5]



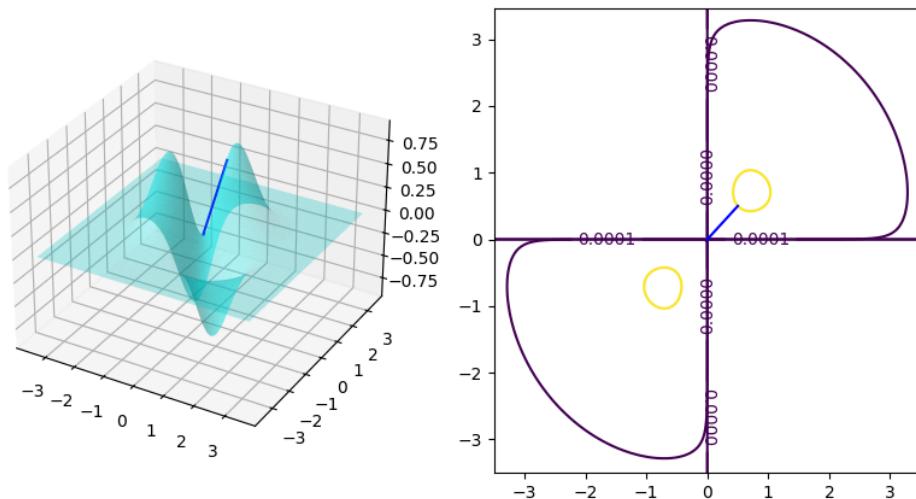
```
Method: Quasinewton (BGFS)
Start point: [0.5 0.5]
x: [2.67102809e-10 2.67102819e-10]
y: 3.5671956687343314e-19
time:          0.002000570297241211
memory:        6057
function_calls: 7
gradient_calls: 7
hessian_calls:  0
iterations:     3

Method: Quasinewton (L_BFGS-B)
Start point: [0.5 0.5]
x: [9.78550101e-11 9.78550101e-11]
y: 4.787801499134087e-20
time:          0.0009996891021728516
memory:        17609
function_calls: 7
gradient_calls: 7
hessian_calls:  0
iterations:     3
```

Quasinewton (BGFS) для функции notpolinom, epsilon = 1e-08, начальная точка: [0.5 0.5]



Quasinewton (L_BFGS-B) для функции notpolinom, epsilon = 1e-08, начальная точка: [0.5 0.5]



Сравнения методов

Сравнение своей реализации метода Ньютона с методом Newton-CG из библиотеки `scipy.optimize` (по точности и скорости).

По результатам сравнения видно, что метод Newton-CG эффективнее работает, чем собственная реализация метода Ньютона и с постоянным шагом, и с одномерным

поиском.(проверим на примере функции Розенброка в точке (200, 200))

```
Method: Newton
Start point: [200. 200.]
x: [1. 1.]
y: 2.368554867926088e-28
time: 0.12408328056335449
memory: 45862
function_calls: 8254
gradient_calls: 192
hessian_calls: 192
iterations: 192

Method: Newton with constant step
Start point: [200. 200.]
x: [ 43.56080589 1897.38055882]
y: 1814.087286433547
time: 0.14606952667236328
memory: 142820
function_calls: 0
gradient_calls: 1000
hessian_calls: 1000
iterations: 1000

Method: Newton-CG
Start point: [200. 200.]
x: [1.00000042 1.00000085]
y: 1.7921258713389796e-13
time: 0.0753633975982666
memory: 58003
function_calls: 238
gradient_calls: 526
hessian_calls: 0
iterations: 130
```

Метод	Точность	Время работы
newton(const step)	[43.56080589 1897.38055882]	0.14606952667236 328
newton(golden ratio)	[1. 1.]	0.12408328056335 449
newton-cg	[1.00000042 1.00000085]	0.07536339759826 66

Точность может разниться у собственной реализации и Newton-CG из-за разного взятого эпсилон (в собственной

реализации с эпсилон сравнивается норма градиента в данной точке, в реализации Newton-CG условие останова может быть другое), но по времени Newton-CG выигрывает, т.к. метод сопряжённых градиентов позволяет быстрее достигать локального оптимума в полученном направлении.

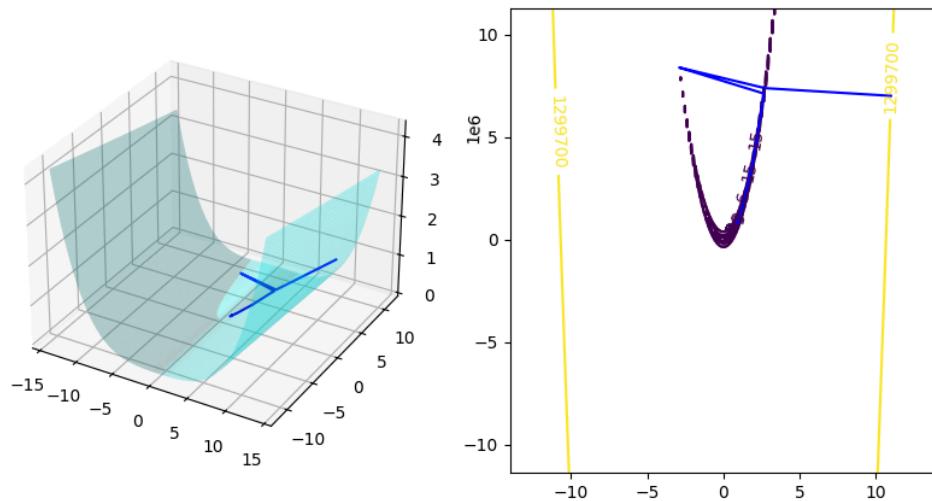
Сравнение эффективности методов нулевого порядка и градиентного спуска из лаб 1., метода Ньютона, квазиньютоновских методов.

Сравним это на примере функции Розенброка в точке (11, 7)

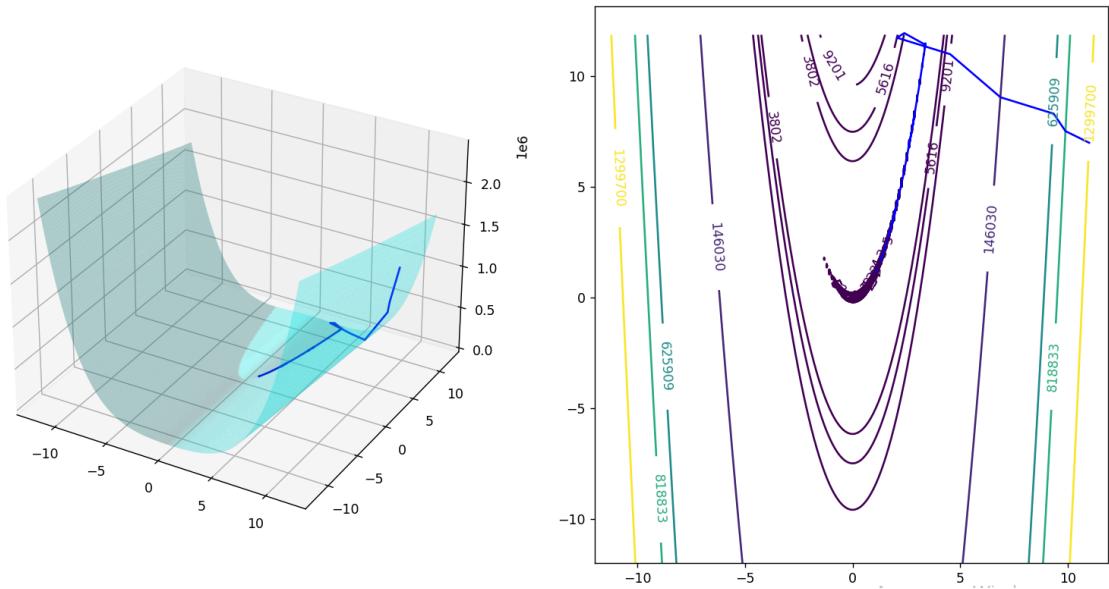
Метод	Точность	Время работы	Количество вычислений значений функции	Кол-во итераций
Нелдера-Мида	[1. 1.]	0.2424201 96533203 12	287	154
Град.спуск	[1. 1.]	0.0478224 75433349 61	3866	89
Ньютона	[1.000001 13 1.0000022 5]	0.0293345 45135498 047	86	53
Квази-Ньютона	[1. 1.]	0.0315964 22195434 57	126	95
Квази-Ньютона 2	[0.999999 24 0.9999984	0.0052452 08740234 375	52	42

	6]			
--	----	--	--	--

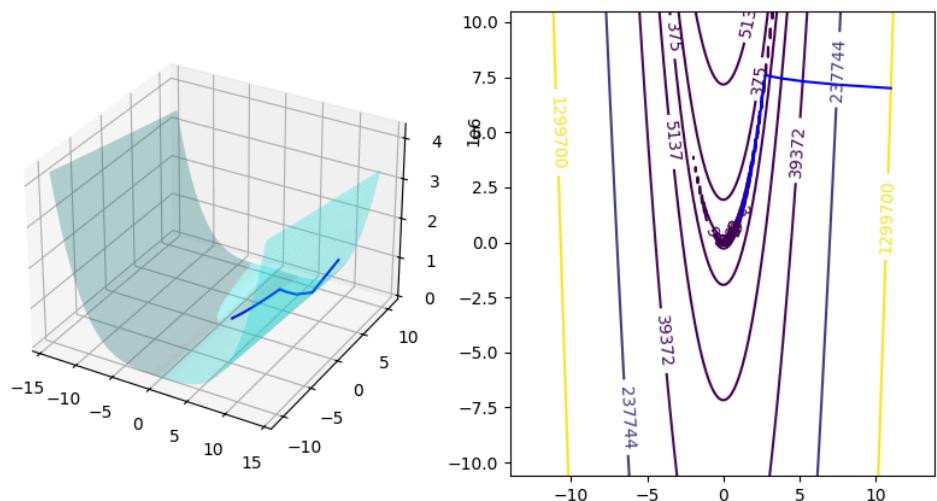
Gradient descend для функции Rosenbrock, epsilon = 1e-08, начальная точка: [11. 7.]



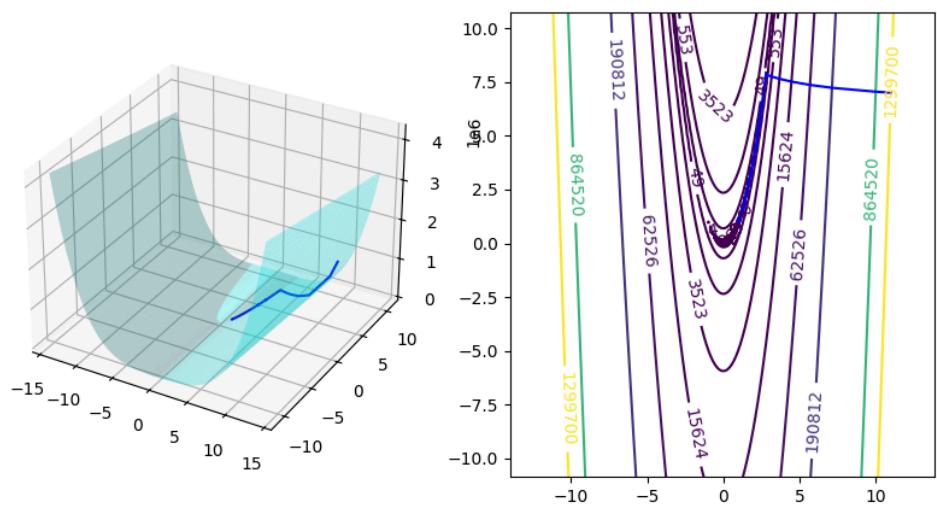
Нэлдер мид для Rosenbrock, epsilon = 1e-08, начальная точка: [11 7]



Newton-CG для функции Rosenbrock, epsilon = 1e-08, начальная точка: [11. 7.]



Quasinewton (L_BFGS-B) для функции Rosenbrock, epsilon = 1e-08, начальная точка: [11. 7.]



```
Method: Gradient descend
Start point: [11. 7.]
x: [0.99999999 0.99999998]
y: 7.515358993420253e-17
time: 0.04782247543334961
memory: 17692
function_calls: 3866
gradient_calls: 89
hessian_calls: 0
iterations: 89
```

```
Method: Newton-CG
Start point: [11. 7.]
x: [1.00000113 1.00000225]
y: 1.2680681876681843e-12
time: 0.029334545135498047
memory: 46511
function_calls: 86
gradient_calls: 193
hessian_calls: 0
iterations: 53
```

```
Method: Quasinewton (BFGS)
Start point: [11. 7.]
x: [1. 1.]
y: 6.034212569790773e-21
time: 0.03159642219543457
memory: 29737
function_calls: 126
gradient_calls: 126
hessian_calls: 0
iterations: 95
```

```
Method: Quasinewton (L_BFGS-B)
Start point: [11. 7.]
x: [0.99999924 0.99999846]
y: 6.579158820187162e-13
time: 0.005245208740234375
memory: 30649
function_calls: 52
gradient_calls: 52
```

Вывод: В целом метод Ньютона оказывается эффективнее градиентного спуска и нулевого порядка. Ему требуется меньше итераций, соответственно меньше времени и вычислений функции. Однако, стоит заметить, что при оптимизации сложных функций (с большой размерностью пространства и наличием сложностей при вычислении второй производной) может возникнуть просадка производительности, поскольку вычисление гессиана может быть многократно более трудоёмкой операцией чем вычисление самой функции. У функции Розенброка в районе $y = x^2$ есть ложбинка. Для градиентного спуска если начать поиск из точки в ложбинке, то траектория пойдет по ней к минимуму функции. Но шаги будут небольшие, потому что значение градиента маленькое в этой ложбине.

Что касается квазиньютоновских методов, BFGS хорошо себя показывает, L-BFGS-B показывает наилучшую производительность. Это обуславливается тем, что второй метод хранит меньше информации о предыдущих итерациях и в целом использует меньше памяти, что может повысить вычислительную эффективность.

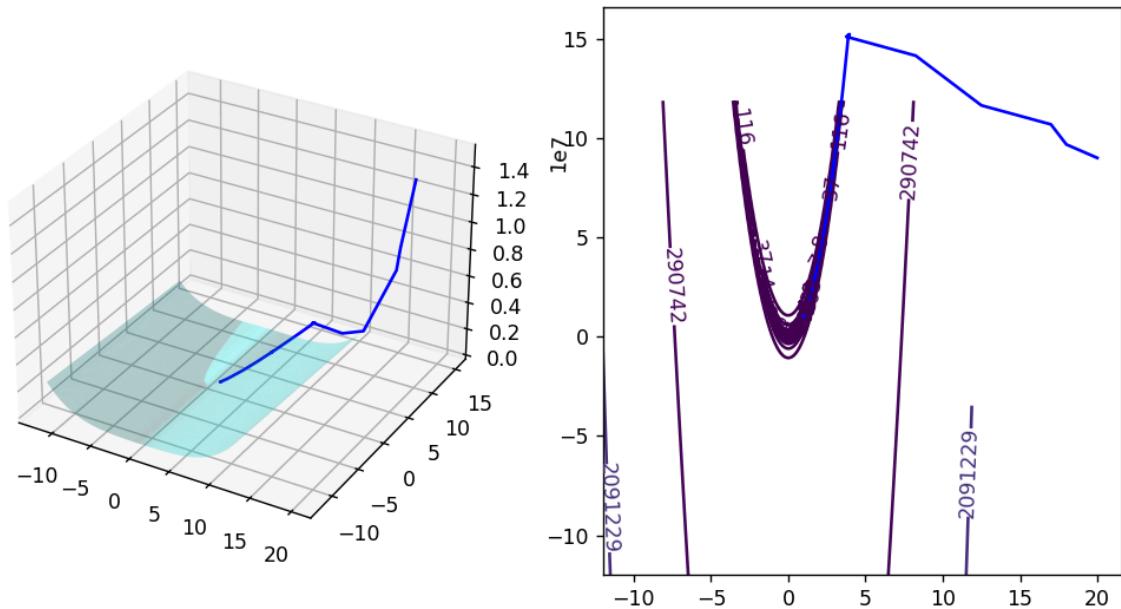
Сравнение эффективности методов нулевого порядка с квазиньютоновскими методами, если в последних производная вычисляется разностным методом

Разностный метод это способ аппроксимировать производную с помощью конечных разностей. Мы будем использовать метод 2-point из scipy. Благодаря тому, что этому методу не нужен градиент он может работать и для зашумленных функций.

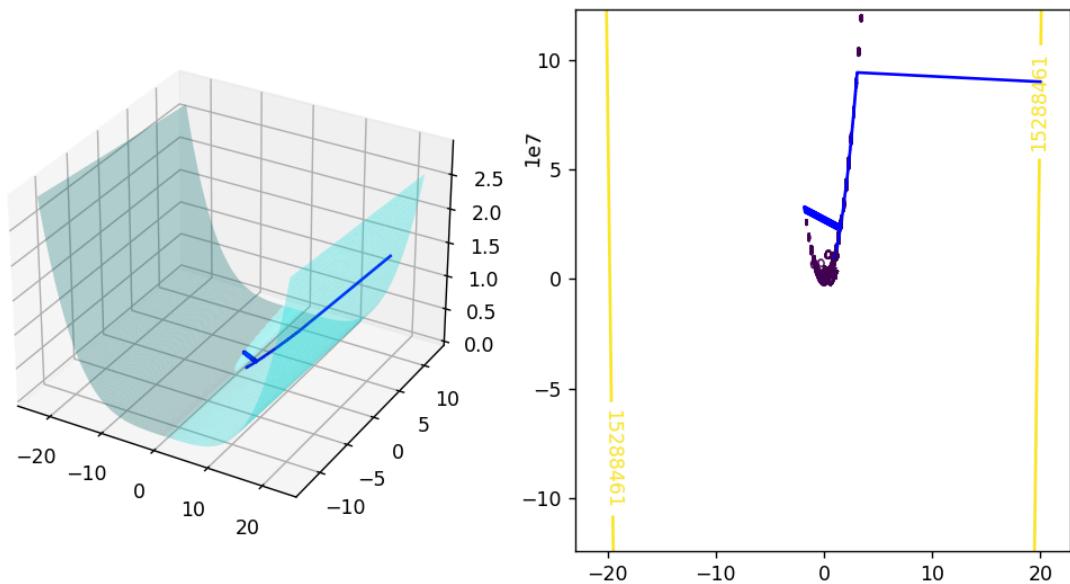
Возьмем функцию Розенброка и начальную точку (20, 9)

Метод	Полученная точка	Количество итераций	Время работы	Количество вычислений функции
Нелдер-Мид	[1, 1]	158	0.2233722 20993042	298
Градиентный спуск	[1.000000 01 1.000000 01]	866	0.5820286 27395629 9	48552
BFGS	[0.999999 47 0.999999 08]	137	0.0992243 28994750 98	615
L_BFGS_B	[0.999995 74 0.999991 48]	47	0.0259993 07632446 29	177

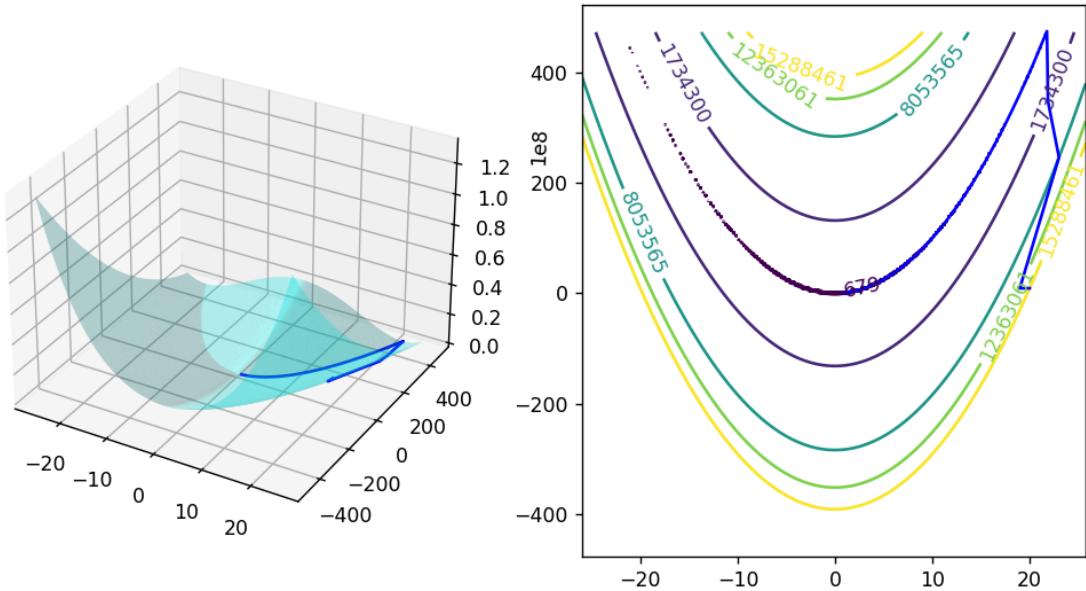
Нелдер мид для Rosenbrock, epsilon = 1e-08, начальная точка: [20 9]



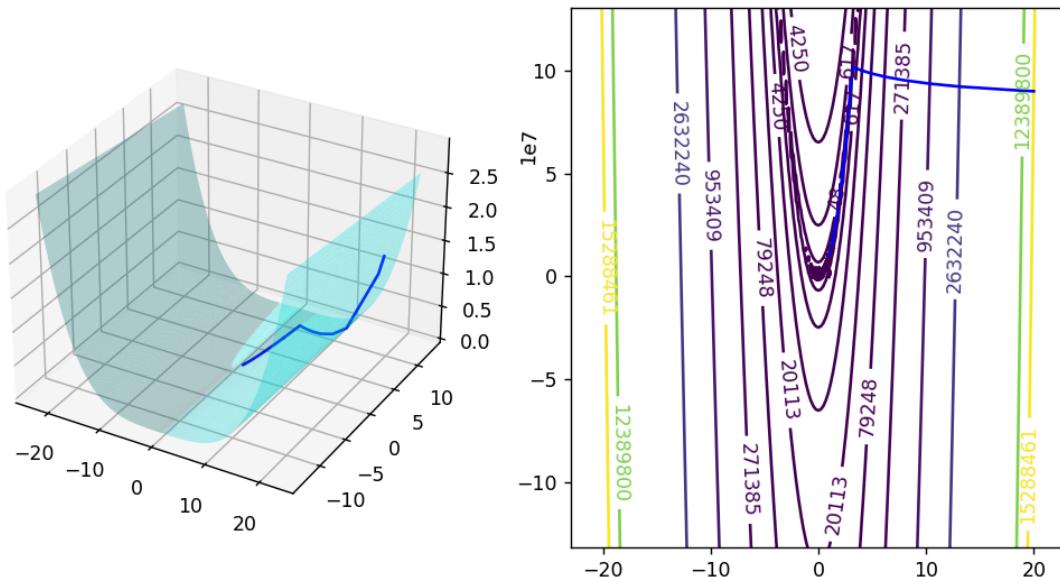
Gradient descend для функции Rosenbrock, epsilon = 1e-08, начальная точка: [20 9]



Quasinewton (BGFS) для функции Rosenbrock, epsilon = 1e-08, начальная точка: [20 9]



Quasinewton (L_BFGS-B) для функции Rosenbrock, epsilon = 1e-08, начальная точка: [20 9]



Квази-Ньютоновские методы отработали лучше.
Градиентный спуск для функции розенброка движется зигзагами это происходит из-за того, что градиент по краям ложбинки $y = x^2$ большой, а в ней он маленький, поэтому ему нужно больше итераций, чтобы сойтись.

Дополнительное Задание 1.

Постановка задачи:

Реализовать одномерный поиск по усиленному правилу Вольфе (Wolfe) и метод Ньютона на его основе. Сравнить с реализованным методами Ньютона и методом NewtonCG

Описание метода:

1. Выбираем начальное значение шага. и параметры b, c1, c2 из [0, 1].
2. Проверяем подходит ли данное альфа под условия армихо и условие кривизны.
3. Если удовлетворяет, то возвращаем альфа, инача a = b * a.

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

$$|\nabla f(x_k + \alpha_k p_k)^T p_k| \leq c_2 |\nabla f_k^T p_k|$$

Тут первое условие это условие армихо, которое проверяет, что новое значение a, будет давать достаточное убывание для функции. То есть значение $f(x_k + \alpha_k p_k)$ (это функция от альфа) в этой точке будет меньше чем значение в её производной приподнятой на коэффициент c_1

Второе условие это усиленное условие кривизны, которое проверяет, что градиент в новой точке меньше по модулю, текущего градиента умноженного на какой-то параметр c_2

Выберем функцию: функция Розенброка

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \text{ с эпсилон } 1e-8$$

- 1) Точка (80, 80)

Method: Newton
Start point: [80. 80.]
x: [1. 1.]
y: 1.1290571705975731e-29
time: 0.10320448875427246
memory: 33825
function_calls: 4556
gradient_calls: 106
hessian_calls: 106
iterations: 106

Method: Newton with wolf
Start point: [80. 80.]
x: [1. 1.]
y: 3.187984133224414e-28
time: 0.0666356086730957
memory: 27084
function_calls: 1410
gradient_calls: 1596
hessian_calls: 186
iterations: 186

Method: Newton-CG
Start point: [80. 80.]
x: [1.00000002 1.00000003]
y: 2.58295256976374e-16
time: 0.04134011268615723
memory: 32490
function_calls: 170
gradient_calls: 359
hessian_calls: 0
iterations: 95

Точка (200, 200)

```
Method: Newton
Start point: [200. 200.]
x: [1. 1.]
y: 2.368554867926088e-28
time: 0.1959984302520752
memory: 27688
function_calls: 8254
gradient_calls: 192
hessian_calls: 192
iterations: 192
```

```
Method: Newton with wolf
Start point: [200. 200.]
x: [1. 1.]
y: 1.8168896457630615e-26
time: 0.12243247032165527
memory: 47406
function_calls: 2572
gradient_calls: 2911
hessian_calls: 339
iterations: 339
```

```
Method: Newton-CG
Start point: [200. 200.]
x: [1.00000042 1.00000085]
y: 1.7921258713389796e-13
time: 0.05406928062438965
memory: 21691
function_calls: 238
gradient_calls: 526
hessian_calls: 0
iterations: 130
```

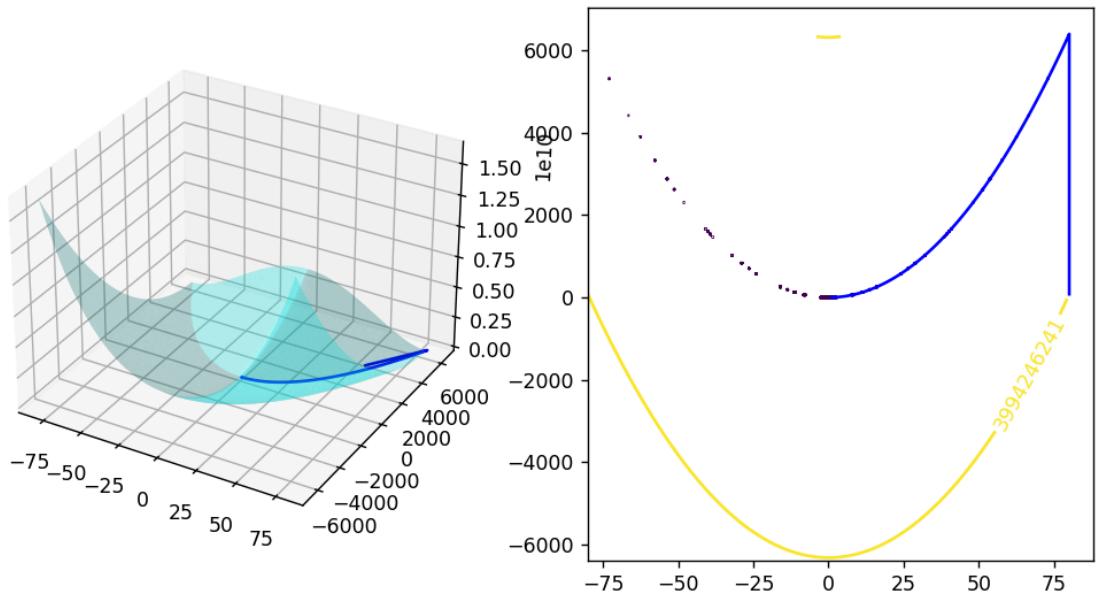
Точка (1000, 1000)

```
Method: Newton
Start point: [1000. 1000.]
x: [1. 1.]
y: 1.67632942359465e-30
time: 0.5227816104888916
memory: 80843
function_calls: 23906
gradient_calls: 556
hessian_calls: 556
iterations: 556
```

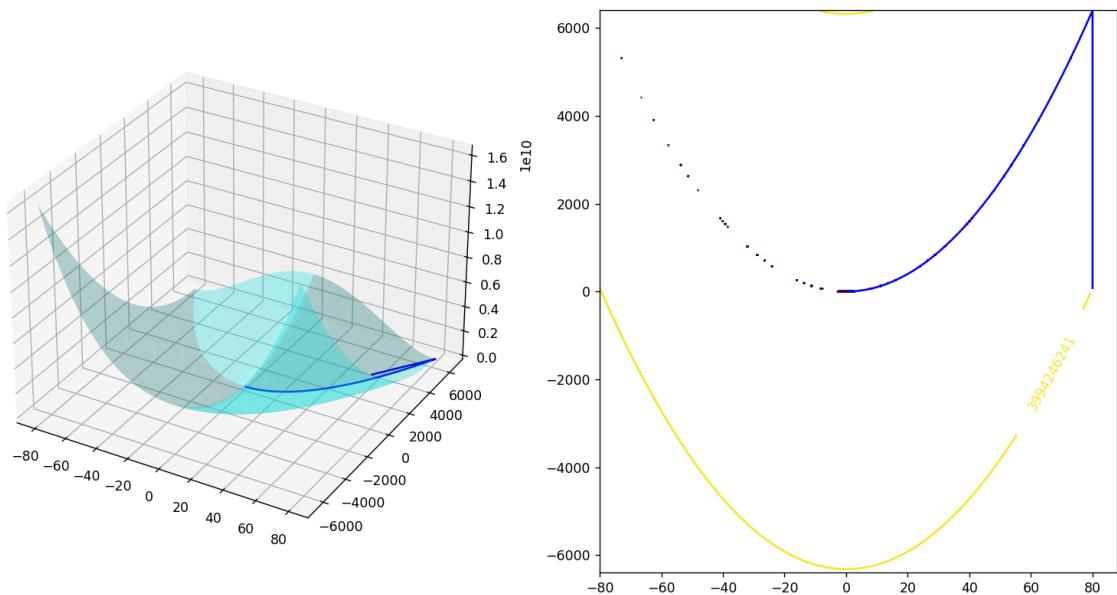
Method: Newton with wolf
Start point: [1000. 1000.]
x: [1. 1.]
y: 1.0842736602676927e-23
time: 0.3699829578399658
memory: 134956
function_calls: 7490
gradient_calls: 8471
hessian_calls: 981
iterations: 981

Method: Newton-CG
Start point: [1000. 1000.]
x: [-26.99092133 728.51498321]
y: 783.494328087954
time: 0.05777454376220703
memory: 18446
function_calls: 272
gradient_calls: 584
hessian_calls: 0
iterations: 89

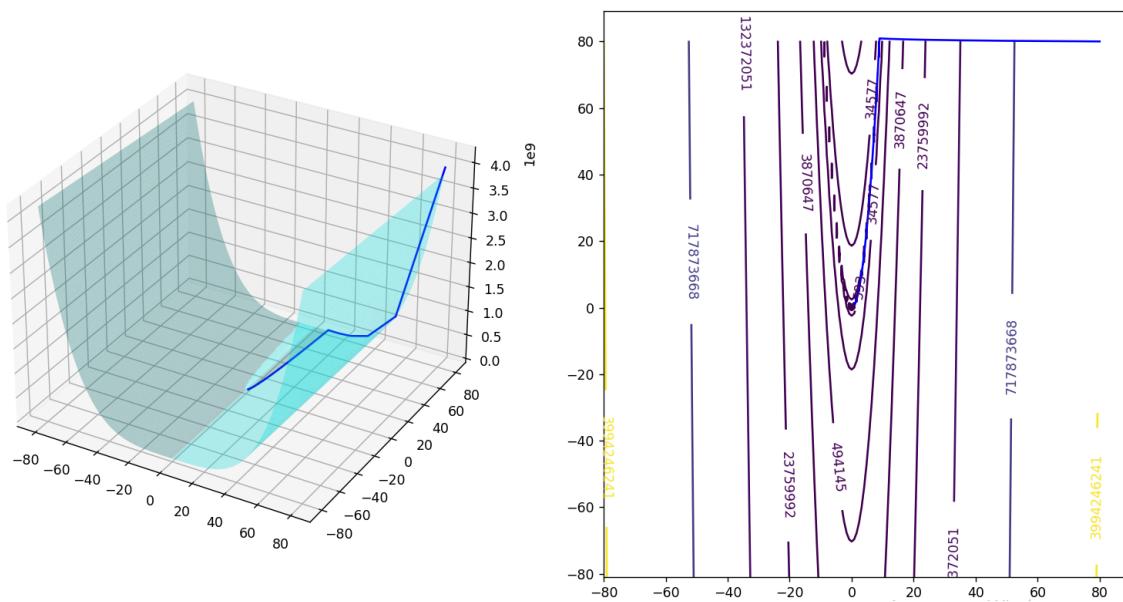
Newton для функции Rosenbrock, epsilon = 1e-08, начальная точка: [80. 80.]



Newton with wolf для функции Rosenbrock, epsilon = 1e-08, начальная точка: [80. 80.]



Newton-CG для функции Rosenbrock, epsilon = 1e-08, начальная точка: [80. 80.]



Начальная точка	Метод	Время	Число итераций
80, 80	Newton	0.103	106
	Newton wolf	0.066	186
	Newton-CG	0.041	95
200, 200	Newton	0.196	192
	Newton wolf	0.122	339
	Newton-CG	0.054	130
1000, 1000	Newton	0.522	556
	Newton wolf	0.370	981
	Newton-CG	0.058	89

Вывод: Методу Вольфа нужно больше итераций для сходимости, чем методу Ньютона с одномерным поиском.

Потому что не ищем минимум по направлению на каждом шаге, а находим какое-то допустимое альфа удовлетворяющее условию, которое может быть не минимальным. Но благодаря этому метод Ньютона с условием Вольфа работает быстрее по времени, чем наш метод Ньютона. Также ему требуется меньше вычислений значений функции, потому что он не использует одномерный поиск, а количество вычислений градиента повысилось, так как при поиске подходящего альфа мы считаем градиент, чтобы проверить условие Вольфа.

Дополнительное задание 2.

Постановка задачи:

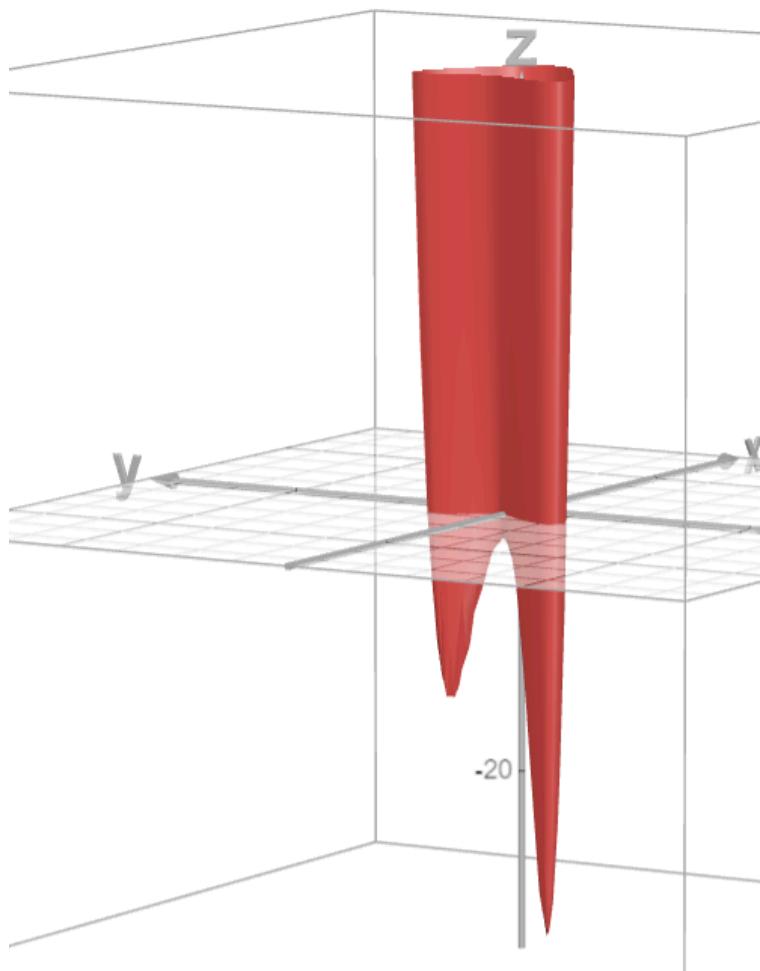
1. Найти функцию, для которой метод Ньютона не работает для некоторой начальной точки.

$$0.1 * (x^4 + y^4) + x * y^2 + 2 * x + 2 * y$$

Минимум у нёё в этой точке

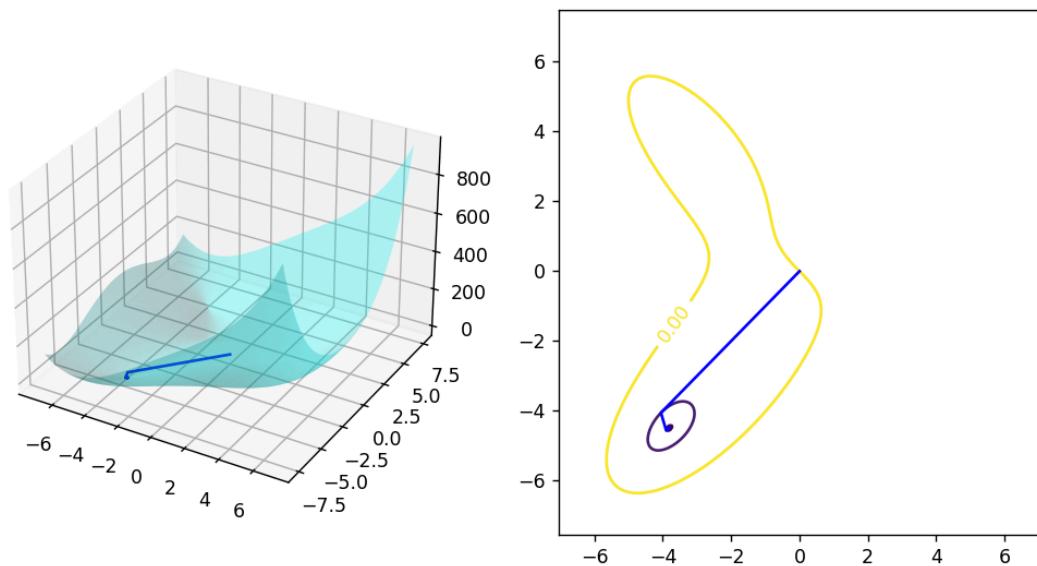
Global minimum

$$\min\{0.1(x^4 + y^4) + x y^2 + 2 x + 2 y\} \approx -31.6954 \text{ at } (x, y) \approx (-3.81326, -4.49214)$$



Возьмём эту функцию в точке $(0, 0)$. градиент в этой точке $(2, 2)$, а определитель гессиана равен 0. Поэтому это мешает воспользоваться методом ньютона. Мы можем сделать пару шагов градиентным спуском и потом продолжить метод ньютона из хорошей точки.

Newton для функции func, epsilon = 1e-08, начальная точка: [0 0]



Method: Newton

Start point: [0 0]

x: [-3.8132586 -4.49214287]

y: -31.695369815676596

time: 0.03598451614379883

memory: 14915

function_calls: 813

gradient_calls: 19

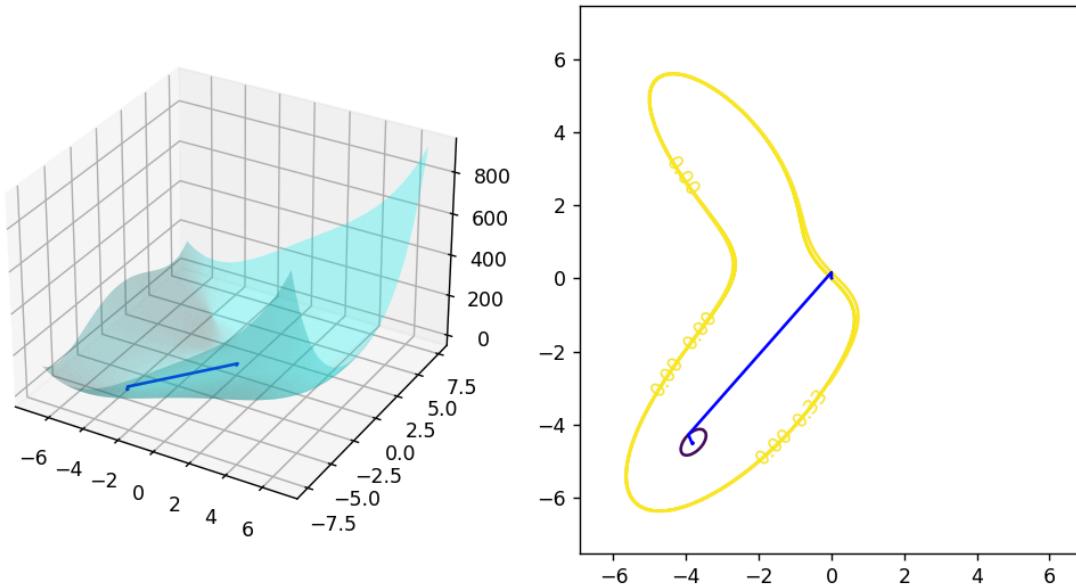
hessian_calls: 19

iterations: 19

$\Delta \kappa$

[

Newton with wolf для функции func, epsilon = 1e-08, начальная точка: [0 0]



```

Method: Newton with wolf
Start point: [0 0]
x: [-3.8132586 -4.49214287]
y: -31.695369815676585
time: 0.0683908462524414
memory: 7780
function_calls: 1776
gradient_calls: 1785
hessian_calls: 9
iterations: 9

```

A

Для метода Вольфа пришлось сделать три шага с помощью градиентного спуска, потому что он блуждал в окрестности нуля. Стратегия делать несколько шагов градиентным спуском, а потом использовать метод Ньютона очень удобна, потому что в самом начале они получают примерно равные результаты, а итерация метода Ньютона стоит гораздо больше. Также еще может возникнуть проблема с тем, что гессиан не будет положительно определен,

соответственно, направление не будет направлением убывания, это можно исправить добавив к диагональным элементам матрицы какое-то число.

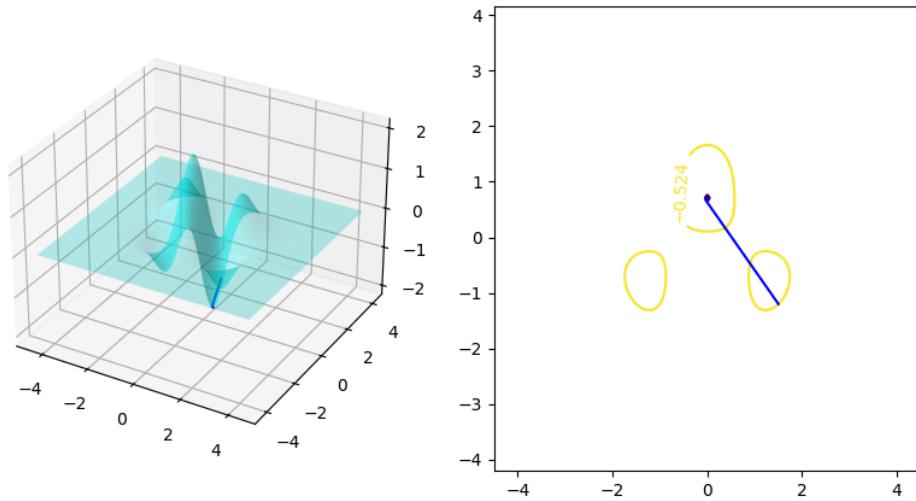
2. для функции со многими точками локального минимума найдите пример, где разные методы, из одной начальной точки, сходятся к разным точкам минимума.

Для исследования возьмем функцию

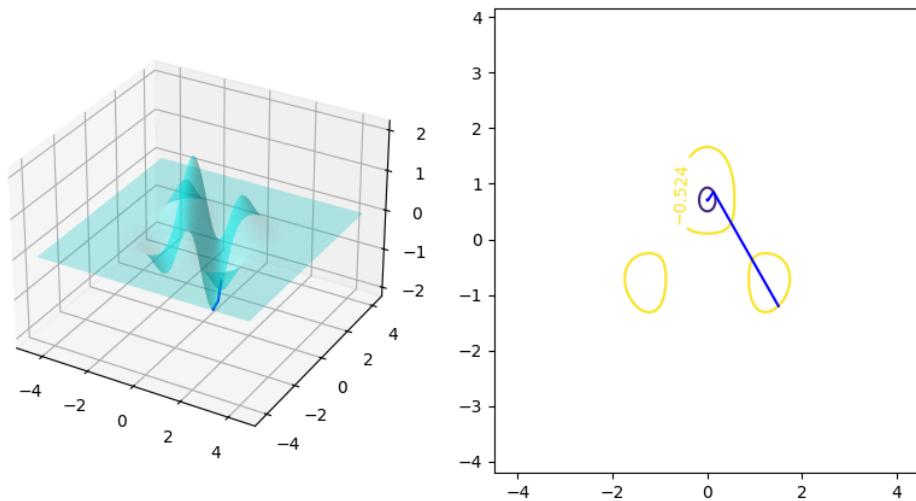
$$5(2x^2 - 1)y * e^{(-x^2-y^2)}$$

в точке (1.5, -1.2)

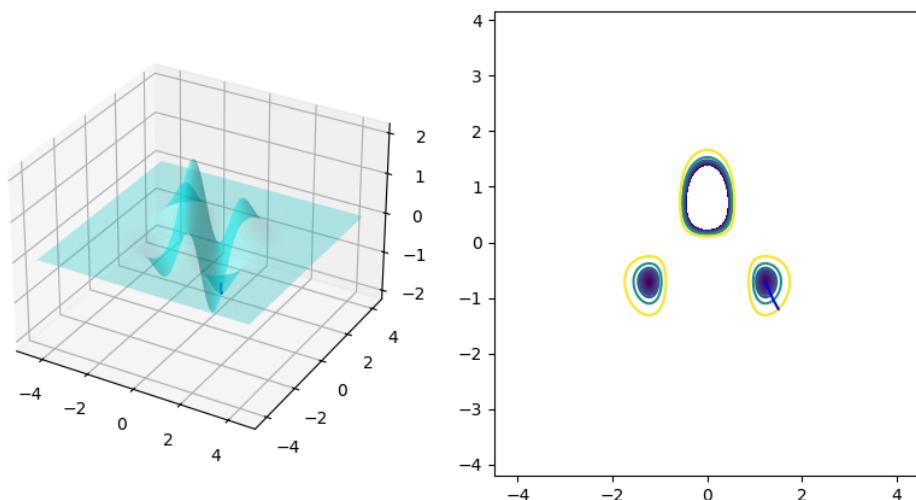
Gradient descend для функции $5 * (2x^2 - 1)y * \exp(-x^2-y^2)$, epsilon = 1e-08, начальная точка: [1.5 -1.2]



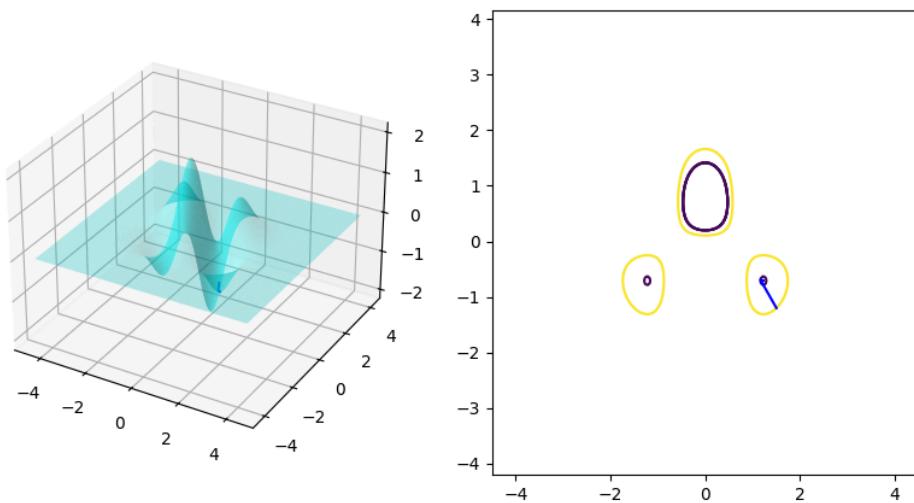
Newton для функции $5 * (2x^2 - 1)y * \exp(-x^2-y^2)$, epsilon = 1e-08, начальная точка: [1.5 -1.2]



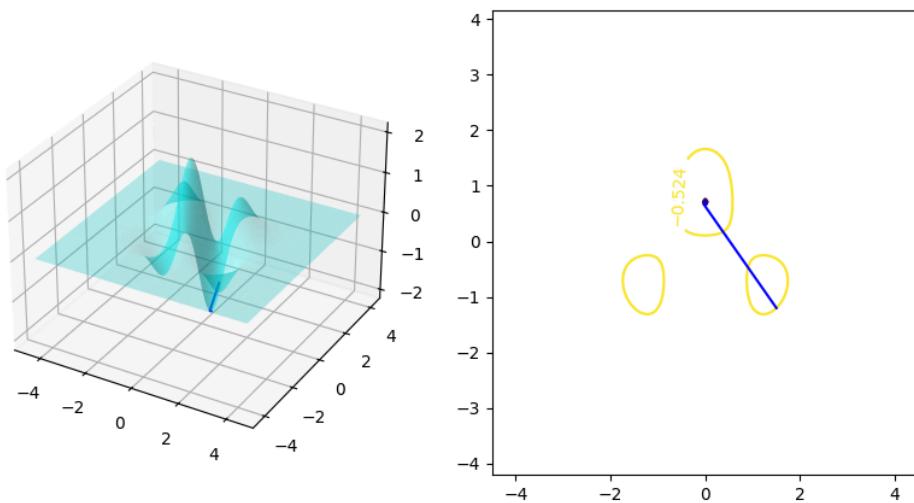
Newton with constant step для функции $5 * (2x^2 - 1)y * \exp(-x^2-y^2)$, epsilon = 1e-08, начальная точка: [1.5 -1.2]



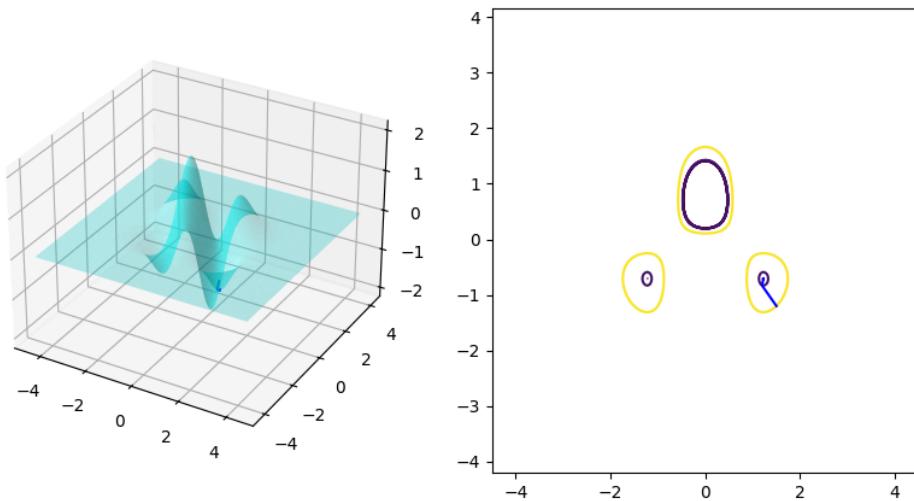
Newton with wolf для функции $5 * (2x^2 - 1)y * \exp(-x^2-y^2)$, epsilon = 1e-08, начальная точка: [1.5 -1.2]



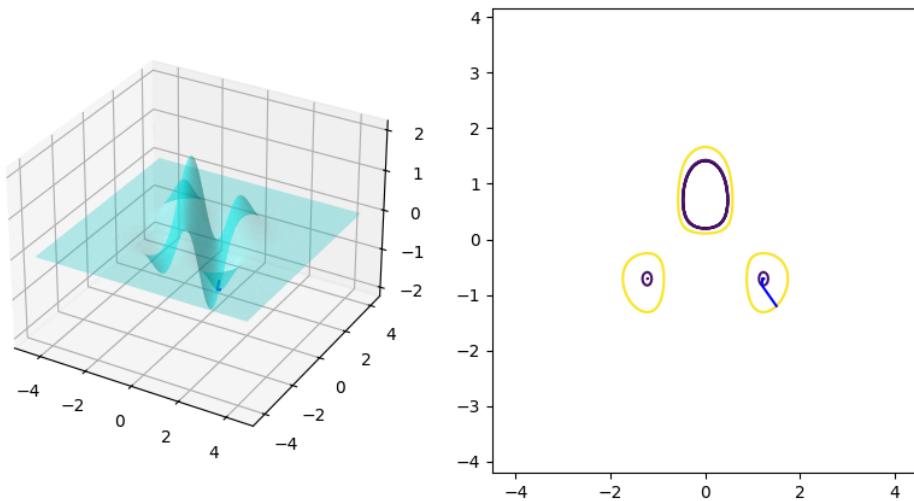
Newton-CG для функции $5 * (2x^2 - 1)y * \exp(-x^2-y^2)$, epsilon = 1e-08, начальная точка: [1.5 -1.2]



Quasinewton (BGFS) для функции $5 * (2x^2 - 1)y * \exp(-x^2-y^2)$, epsilon = 1e-08, начальная точка: [1.5 -1.2]



Quasinewton (L_BFGS-B) для функции $5 * (2x^2 - 1)y * \exp(-x^2-y^2)$, epsilon = 1e-08, начальная точка: [1.5 -1.2]



Заметим, что на методе Ньютона с постоянным шагом мы нашли локальный минимум, к которому мы были ближе, вместо глобального минимума. Это объясняется тем, что мы неоптимально ищем направление спуска. Так же происходит и с методом Ньютона с Вольфе, потому что он не использует одномерный поиск. В основе квазиньютоновского метода лежит не вычисление матрицы Гессе, а лишь ее аппроксимация в окрестности данной точки, поэтому в этих случаях в результате тоже получаем не глобальный экстремум, а локальный.

Вывод:

Метод Ньютона использует второе приближение функции для ее аппроксимации и нахождении минимума. Поэтому часто он работает эффективнее обычного градиентного спуска. Особенно это заметно на квадратичных функциях и на функциях, которые хорошо аппроксимируются квадратичными функциями. Для них метод сходится за пару итераций. Квазиньютоновские методы могут оказаться более эффективными с точки зрения использования памяти и вычислений, например, когда матрица Гессе плохо обусловлена или её вычисление трудозатратно.

Ссылка на гит-репо с кодом

<https://github.com/samiaFife/optimization-methods-lab2>