

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

# Методы оптимизации

Отчет по лабораторной работе №3

⟨Собрано 28 июня 2023 г.⟩

**Работу выполнили:**

Бактурин Савелий Филиппович М32331

Вереня Андрей Тарасович М32331

Сотников Максим Владимирович М32331

**Преподаватель:**

Шохов Максим Евгеньевич

# Решение задачи нелинейной регрессии

Часто решая задачу создания регрессионной модели мы сталкиваемся с тем, что по жизни очень немногие рассматриваемые функции оказываются не представимы в виде обобщенной линейной зависимости или полиномиальной некоторой конечной степени  $k$ . Такая же ситуация часто случается и с некоторым набором данных, который нужно как-то обобщить. Именно в таких случаях к нам на помощь приходит более частный случай регрессионного анализа – *нелинейная регрессия*.

Идея построения нелинейной регрессии как и в случае с полиномиальной заключается в том, чтобы найти математическую функцию, которая максимально точно описывает зависимость между независимой переменной и зависимой от нее. Например, для построения нелинейной регрессии можно использовать функции типа полинома, логарифмической или экспоненциальной зависимости.

В целом весь процесс нахождения нелинейной регрессионной модели можно поделить на два этапа:

- ▷ Определить регрессионную модель  $f(w, x)$ , которая зависит от параметров  $w = (w_1, \dots, w_W)$  и свободной переменной  $x$ .
- ▷ Решить задачу по нахождению минимума суммы квадратов регрессионных остатков:

$$S = \sum_{i=1}^m r_i^2, \quad r_i = y_i - f(w, x_i)$$

Однако, решая в лоб такую задачу, мы сталкиваемся с оптимизационной задачей нахождения параметров нелинейной регрессионной модели. Тут к нам и приходят на помощь различные методы нахождения, в том числе и рассматриваемые ниже: *Gauss-Newton* и *Powell Dog Leg*.

## Gauss-Newton

Напомним, что мы решаем следующую задачу: дана нелинейная модель  $f(w, x)$ , где  $w \in \mathbb{R}^m$ , тогда сумма квадратов регрессионных остатков высчитывается как

$$S = \sum_{i=1}^{\text{sizeof } X} (f(w, x_i) - y_i)^2 \rightarrow \min$$

Итак, пусть  $n = \text{sizeof } X$  и введем некоторые новые объекты для решения задачи, пусть  $w^0 = (w_0^0, w_1^0, \dots, w_m^0)$  – начальное приближение, и

$$\mathbf{J} = \left( \frac{\partial f}{\partial w_j}(w^i, x_i) \right)_{n \times m} \quad - \text{Якобиан, или матрица первых производных}$$

$$\vec{f}_i = (f(w^i, x_i))_{n \times 1} \quad - \text{вектор значений функции } f$$

$$\delta_i = \text{const} \quad - \text{размера шага}$$

Тогда, формула  $i$ -й итерации рассматриваемого метода будет высчитываться как

$$w^{i+1} \leftarrow w^i - \underbrace{\delta_i \cdot (\mathbf{J}_i^T \mathbf{J}_i)^{-1}}_{\beta} \mathbf{J}_i^T (\vec{f}_i - y),$$

где  $\beta$  – это псевдообратная матрица к матрице  $\mathbf{J}_i$ , или решение некоторой задачи многомерной линейной регрессии, где мы ищем такой вектор  $\beta$ , что

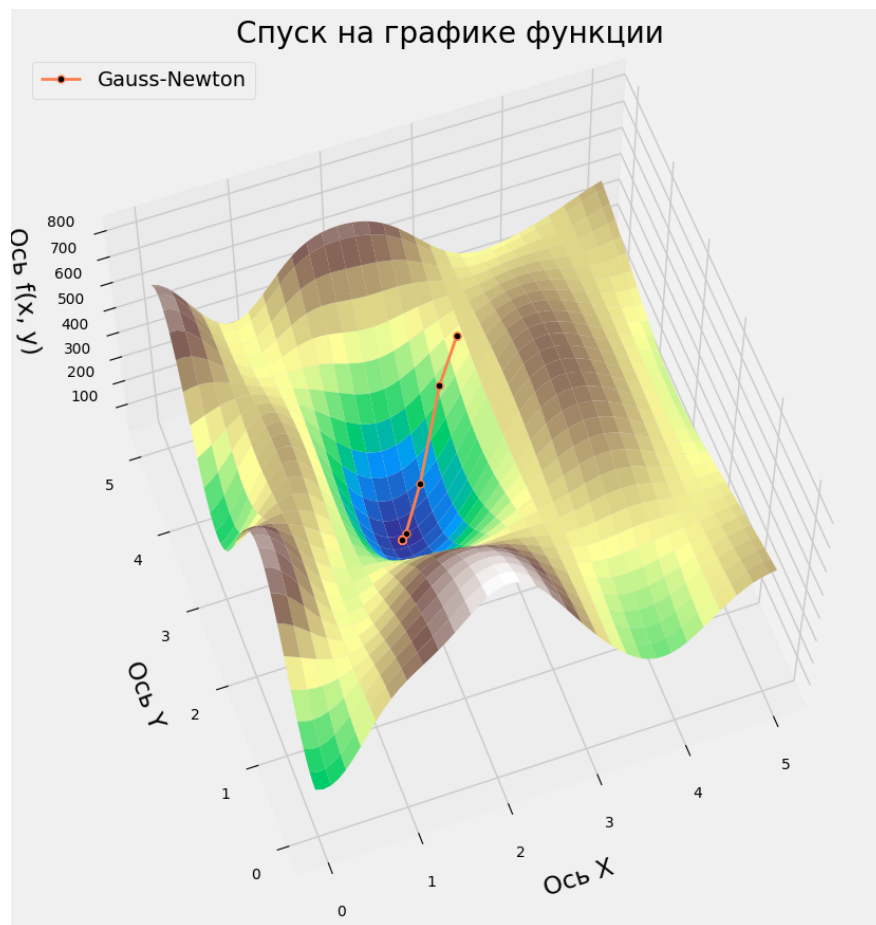
$$\left\| \mathbf{J}_i \beta - (\vec{f}_i - y) \right\|^2 \rightarrow \min,$$

где  $y$  – вектор правильных/настоящих ответов нашей модели. Получается, для решения задачи, мы, так называемую, *невязку* пытаемся приблизить линейной комбинацией вектора из матрицы Якобиана так, что при следующем шаге итерации получить такой  $w^{i+1}$ , который бы сократил нам расстояние невязки. Причем, заметим, что на каждом шаге, задача будет новой, так как  $\mathbf{J}_i$  зависит от текущего приближения, чтобы решить задачу многомерной регрессии.

Заметим, что здесь, по алгоритму, мы видим достаточно очевидное ограничение:  $m \geq n$ , в ином случае для  $\mathbf{J}_i^T \mathbf{J}_i$  не будет существовать обратной матрицы и, в следствии, решения к уравнению.

## Пример

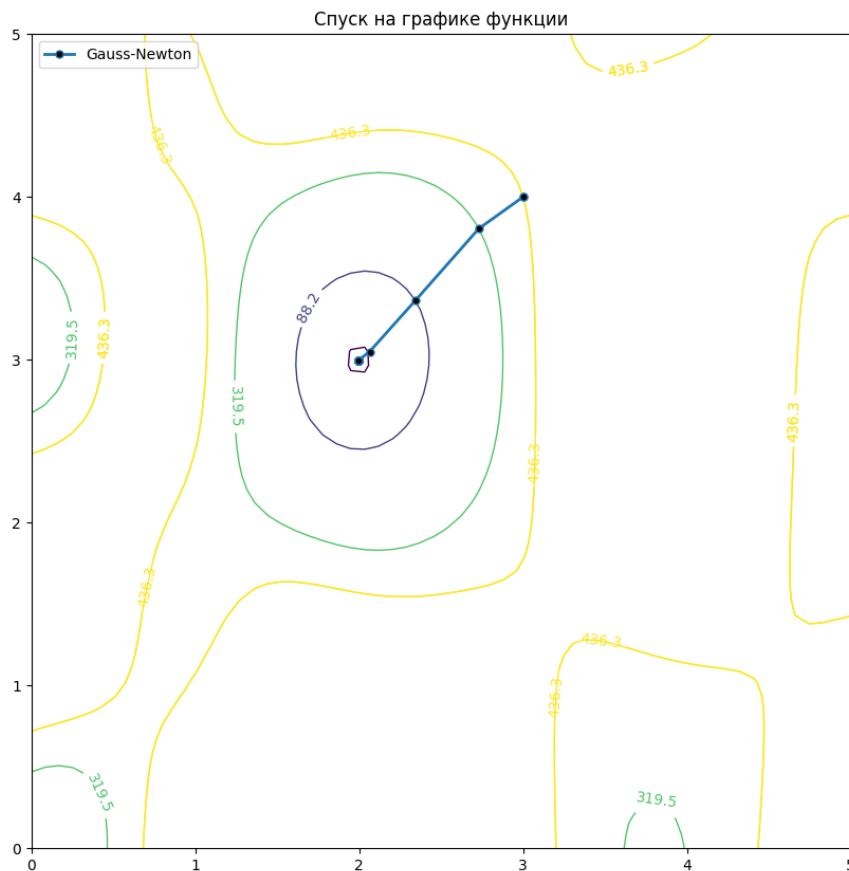
Возьмем в качестве примера функцию  $f(x, w) = \sin(x \cdot w_0 + w_1)$ , где  $w_0$  и  $w_1$  – это подбираемые коэффициенты. Здесь минимум является 5.5. В качестве начальной точки возьмем  $\langle 3, 4 \rangle$  и запустим зверя.



Example of Gauss-Newton

Метод сошелся за 10 шагов, при этом он нашел в качестве аргументов точку

$\langle 1.997695, 2.998286 \rangle$ , а – значения 5.532247. Тот же путь алгоритма в виде линий уровня:



Example of Gauss-Newton (lines)

## Powell Dog Leg

*Trust-region method* — это метод решения оптимизационных задач, который основывается на вычислении региона, в котором квадратичная модель аппроксимирует целевую функцию. Сам этот метод представляет из себя смесь сразу двух алгоритмов, решающих задачу:

- ▷ Линейный поиск используется для определения направления поиска и дальнейшего нахождения оптимального шага вдоль выбранного вектора пути.
- ▷ Сам по себе trust-region используется для определения области вокруг текущей итерации, в которой модель достаточно аппроксимирует целевую функцию. Причем, стоит заметить, что для поиска следующего радиуса рассматриваемого региона также будет использоваться линейный поиск.

В общем случае Trust-region на каждой итерации решает следующую квадратичную задачу:

$$\min_{p \in \mathbb{R}^n} m_k(p) = f_k + p^T g_k + \frac{1}{2} p^T B_k p,$$

где  $f_k = f(x_k)$ ,  $g_k = \nabla f_k$ ,  $B_k = \nabla^2 f_k$  и  $\nabla_k > 0$  — изменяющийся радиус региона, причем всё это, при условии, что  $|p| \leq \nabla_k$ . Заметим, что в таком простейшем виде

мы получаем безусловно почти бесполезный алгоритм: он чрезвычайно медленный из-за появления  $B_k$  – Гессиана функции. С другой стороны, если он положительно определен и  $|B_k^{-1}\nabla f_k| \leq \nabla_k$ , то решение легко определить:  $p_k^B = -B_k^{-1}\nabla_k$ . Но, опять же, высчитывать еще и обратную матрицу – дело долгое и медленное, поэтому, начиная отсюда и до конца все лабораторной работы, мы будем то и дело пытаться приближать наши значения к реальным/по настоящему посчитанным значениям Гессиан-функции.

Здесь мы рассмотрим один из методов оптимизации при аппроксимации квадратичной модели – *Powell Dog Leg*. Начнем, пожалуй, с определения радиуса рассматриваемого доверительного региона: в алгоритме *dogleg* обычно выбирают основываясь на сходстве функции  $m_k$  (та, что мы решаем изначально) и оригинальной функции  $f$  на предыдущей итерации. Зададим  $\rho_k$  следующим образом:

$$\rho_k = \frac{f_k - f_k^*}{m_k(0) - m_k(p_k)},$$

где  $f_k^* = f(x_k + p_k)$ . А теперь посмотрим на то, как именно лучше поменять шаг: в том случае, если  $\rho_k$  меньше нуля, то это значит, что наша модель далека от функции и нужно обязательно уменьшить радиус; в том случае, изменение функции почти не изменилось и мы попали на границу региона, то есть смысл увеличить радиус; в ином другом случае – остается неизменным.

$$\Delta_{k+1} = \begin{cases} \frac{1}{4}\Delta_k, & \rho_k < \frac{1}{4} \\ \min(2\Delta_k, \Delta_{\max}), & \rho_k > \frac{3}{4} \wedge \|p_k\| = \Delta_k \\ \Delta_k, & \text{в ином другом случае} \end{cases}$$

Наконец, начинается самое интересное со стороны *Powell Dog Leg*. Итак, мы находимся на некоторой точки нашей модели, есть подсчитанный  $\Delta$ -радиуса доверительного региона, и посмотрим на полный шаг  $p^B = -B^{-1}g$ . Если  $p^B$  лежит в окружности региона, то мы можем его взять и более закончить алгоритм. В ином случае, рассмотрим анти-градиент  $-g$  и попробуем вдоль нее поискать минимум квадратичной модели, то есть решить

$$\min_{\|-\tau g\| \leq \Delta} m(-\tau g)$$

Для её решения мы можем взять некую новую точку без каких-либо ограничений в направлении анти-градиента и найти минимум модели

$$p^U = -\frac{g^T g}{g^T B g} g$$

Здесь снова две ситуации, где может находиться т.  $p^U$ :

▷ Если она находится вне рассматриваемой области, то мы можем взять точку на границе и шагнуть туда.

▷ Если же она находится в окружности, то построим отрезок  $p^U p^B$  и начнем искать минимум вдоль этих двух линий  $\left( \text{текущая} \rightarrow p^U \text{ и } p^U \rightarrow p^B \right)$ .

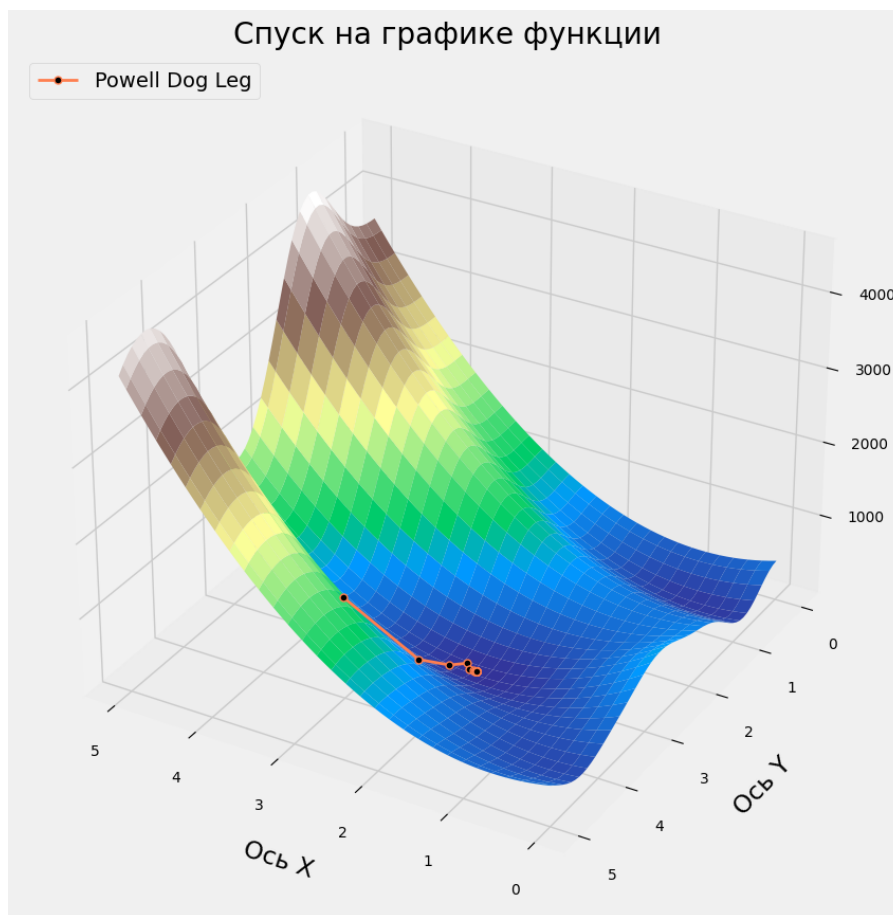
Наконец, вдоль пути мы рассматриваем траекторию  $\hat{p}(\tau)$

$$\hat{p}(\tau) = \begin{cases} \tau p^U, & 0 \leq \tau \leq 1 \\ p^U + (\tau - 1)(p^B - p^U), & 1 \leq \tau \leq 2 \end{cases}$$

Подытожим. Мы получили, на самом деле, в чем-то схожий на метод Гаусса-Ньютона алгоритм нахождения схождения, в частности, кстати, точка  $p^B$  – это то, куда бы шагнул метод Гаусса-Ньютона, но при этом, если эта точка удовлетворяет нашим потребностям, то мы действуем как Гаусс-Ньютон, в ином случае – чуть по другому. Причем под «немного другим» способом предполагается, на самом деле, хитрая комбинация Гаусса-Ньютона и градиентного спуска (так как при маленьком доверительном регионе мы пойдем по направлению, близкому градиентному спуску).

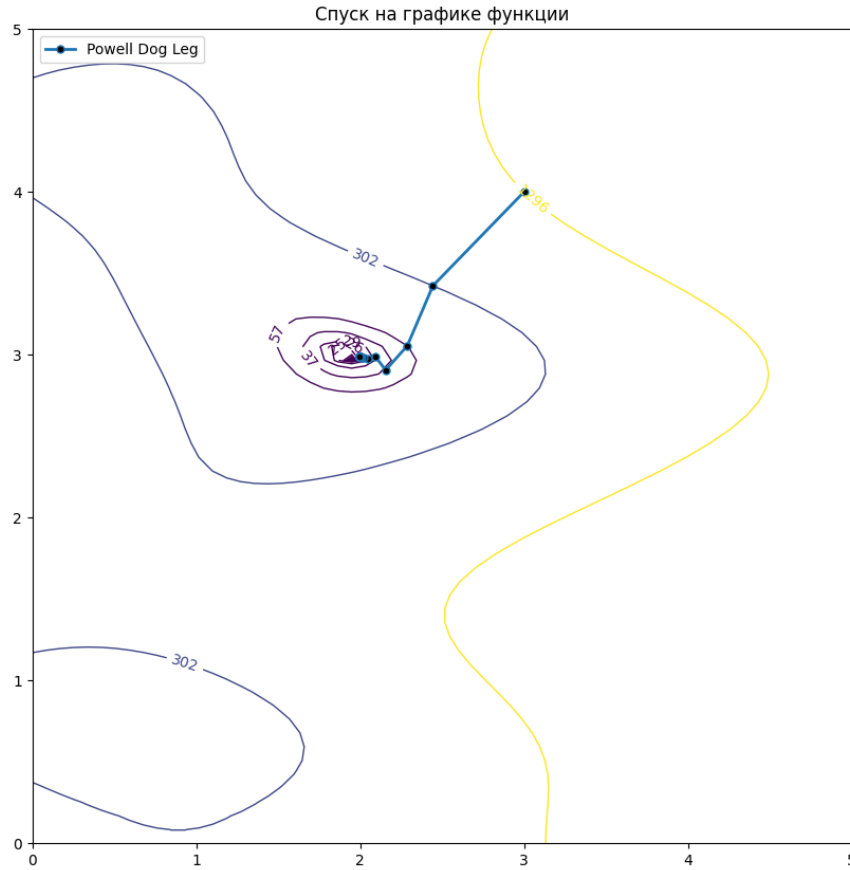
## Пример

Возьмем в качестве примера функцию  $f(x, w) = [\text{ДАННЫЕ УДАЛЕНЫ}]$ . В качестве начальной точки возьмем  $\langle 3, 4 \rangle$  и запустим зверя.



Example of Dog Leg

Метод сошелся за 38 шагов, при этом он нашел в качестве аргументов точку  $\langle 1.997871, 2.987030 \rangle$ , а – значения 24.179801. Тот же путь алгоритма в виде линий уровня:



Example of Dog Leg (lines)

## Исследования Gauss-Newton vs. Powell Dog Leg vs. Adam

В качестве исследования мы будем сравнивать сразу несколько методов в одной главе, так как именно так будет лучше видна разница между ними. Везде и всегда мы будем брать в качестве восстанавливаемой функции нашего старого друга функцию Розенброка:  $f(x, w) = (1 - x \cdot w_1)^2 + 100 \cdot (x \cdot w_2 - x^2)^2$ , где  $w_1 = 2$ ,  $w_2 = 3$ . Также здесь и везде будем применять следующие настройки генерации точек:

```
density = 8000 (количество точек на плоскости)
dots_count = 1000 (количество генерируемых точек для dataset)
radius = 0.001 (радиус генерации точек)
dist = 0.01 (разброс функции)
```

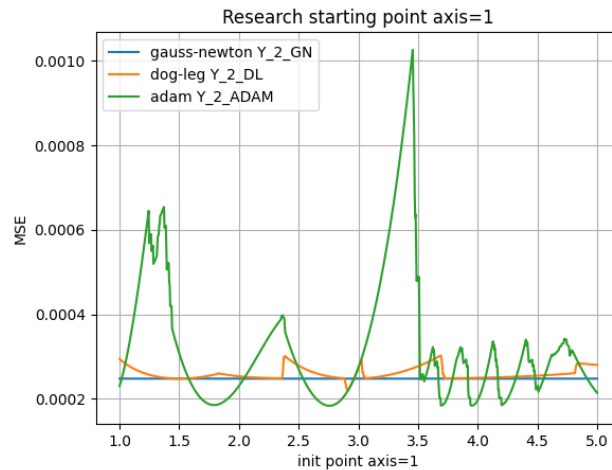
И, наконец, количество тестирований для усреднения результатов мы выберем `test_count = 10`. Положим точку  $m_0 = \langle x_0, y_0 \rangle$  как настоящий минимальный прообраз рассматриваемой функции и зададим пару  $P = \langle \text{axis}, \text{bias} \rangle$ , где `axis` – рассматриваемая ось смещения, `bias` – значение смещения. Проводить, наконец, исследование мы будем следующим образом: зададим некую пару  $P$  и в качестве `initial_w = m_0` и далее для заданного  $P.\text{axis}$  мы будем изменять начальную точку следующим образом:

$\langle m_0.(P.\text{axis}) - P.\text{bias} \rangle, \langle m_0.(P.\text{axis}) - P.\text{bias} + \text{size} \rangle, \dots, \langle m_0 \rangle, \langle m_0.(P.\text{axis}) + \text{size} \rangle, \dots$

То есть, например, если  $\text{axis} = Y$  и  $\text{bias} = 10$ , то мы начнем с точки  $x_0 = m_0.(x)$  и  $y_0 = m_0.(y) - \text{bias}$  и с некоторым шагом дойдем до настоящего минимум и далее до точки  $\langle x_0 = m_0.(x), y_0 = m_0.(y) + \text{bias} \rangle$ .

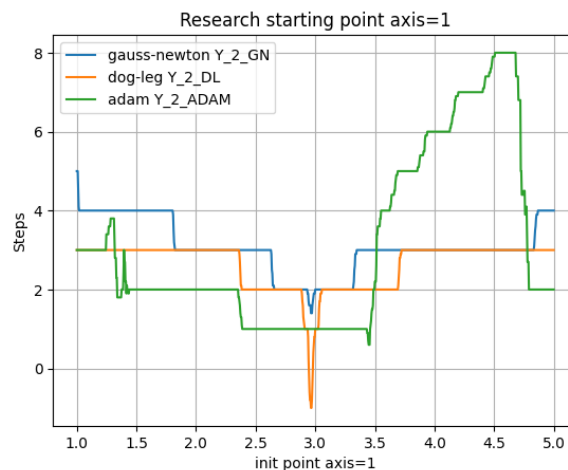
$$\mathbf{P} = \langle \mathbf{Y}, 2 \rangle$$

Итак, рассмотрим первый случай, с функциям минимизации потерь у всех трех методов. Получим график MSE-значения.



$\langle Y, 2 \rangle$  - MSE

Замечаем, что лучшим из них показал результат Gauss-Newton, из-за устойчивости метода. В то время как Dog Leg показывает результат хуже, из-за неправильности первого шага поиска оптимального радиуса доверительного региона. И тут без комментариев Adam проигрывает, из-за того, что вообще говоря функция Розенброка не представима в виде линейной регрессии. Теперь рассмотрим количество шагов у трех методов одновременно:



$\langle Y, 2 \rangle$  - Steps

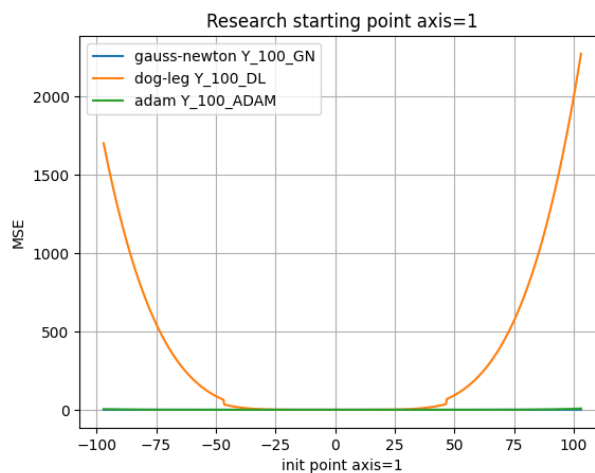
У Gauss-Newton мы видим самые стабильные изменения в количестве шагов - неудивительно, ведь мы подбирались все это время всё ближе и ближе к минимуму.



С Dog Leg ситуация аналогична, вот только вместо того, чтобы делать в минимум привычный шаг как Gauss-Newton, он делает первым шаг по поиску доверительной окружности, что сильно быстрее сходится начальную точку к минимуму. И, наконец, Adam. Снова проигрывает в этой гонке. Причем, заметим, что в он сильно увеличился уже именно на  $+0$  знатно – вероятнее всего, это связано уже с тем, как работает Adam с положительными числами.

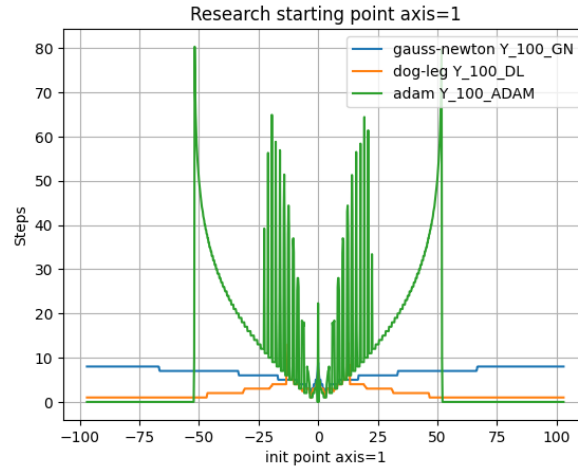
$$\mathbf{P} = \langle \mathbf{Y}, 100 \rangle$$

Итак, рассмотрим второй случай, с функциям минимизации потерь у всех трех методов. Получим график MSE-значения.



$\langle Y, 100 \rangle$  - MSE

Видим, что ситуация с Dog Leg'ом сильно пострадала. Утверждение следующее: как только Dog Leg делает неправильный шаг, он останавливается и прекращает какие-либо дальнейшие попытки сойтись, ибо у него есть ограничение на не увеличение радиуса доверительного круга и так получается, что при дальних расстояниях и маленькой рассматриваемой окружности. Ситуация с Adam и Gauss-Newton, что самое интересное, почти идеальное: они сошлись в одну асимптотически почти на-верное прямую. Но как мы увидим дальше, не все так просто, как кажется, а именно: давайте посмотрим на количество шагов до схождения всех трех методов:

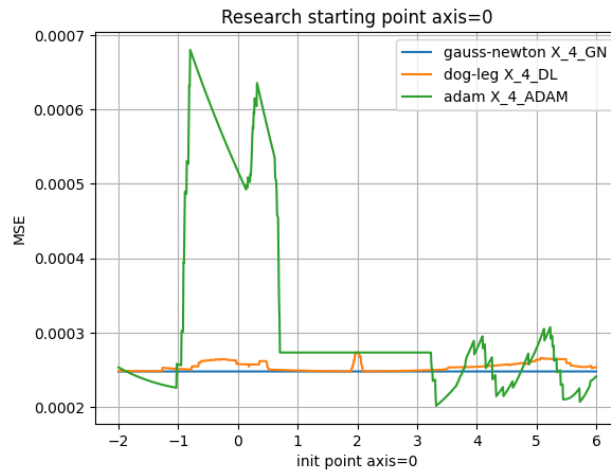


$\langle Y, 100 \rangle$  - Steps

Не смотрим на Dog Leg, ибо, как уже сказано выше, количество шагов здесь – это число до того, как он остановится от печали. В отличие от Gauss-Newton, Adam очень странно себя ведет: вероятнее всего, это связано с начальным `learning_rate = 2.5`, который остается неизменным на протяжении всего эксперимента, из-за чего на некоторых расстояниях Adam начинает вести себя ненормально. Gauss-Newton здесь и далее будет понемногу уменьшать количество шагов как по лестнице.

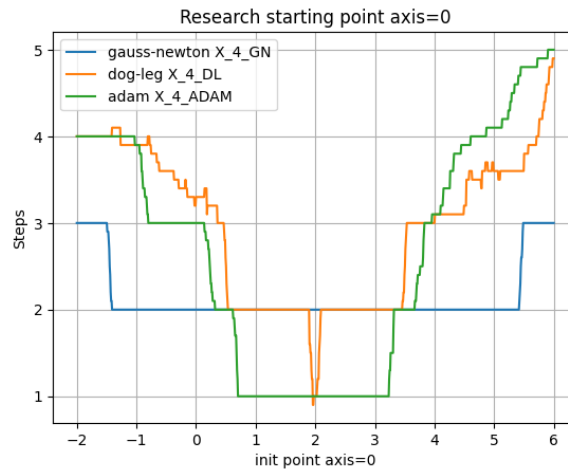
$$P = \langle X, 4 \rangle$$

Настало время поэкспериментировать, на точках, с изменяющимся осью  $X$ . Итак, рассмотрим первый случай, с функциям минимизации потерь у всех трех методов. Получим график MSE-значения.



$\langle X, 4 \rangle$  - MSE

Здесь мы получаем аналогичные результаты с случаем  $P = \langle Y, 2 \rangle$ , поэтому, долго задерживаться не будем, а лучше сразу посмотрим на количество затрачиваемых шагов:

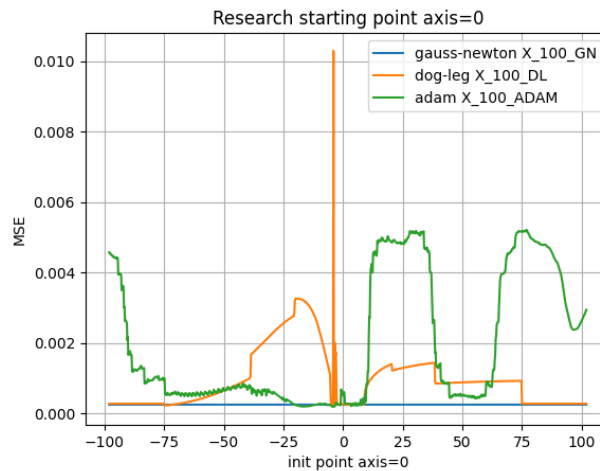


$\langle X, 4 \rangle$  - Steps

И здесь ровно такая же, аналогичная, ситуация с  $\langle Y, 2 \rangle$ .

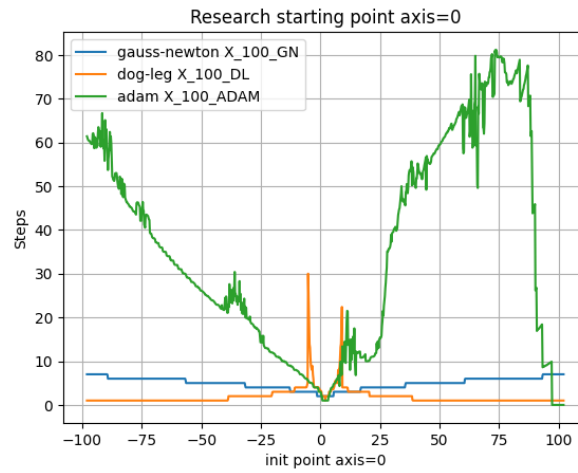
$\mathbf{P} = \langle \mathbf{X}, 100 \rangle$

Итак, рассмотрим второй случай, с функциям минимизации потерь у всех трех методов. Получим график MSE-значения.



$\langle X, 100 \rangle$  - MSE

А вот здесь результат интереснее: дело в том, что по каким-то причинам, Dog Leg в районе минимум все равно ошибается на какое-то предельно-маленькое, но незаметное значение для MSE. Gauss-Newton всё также хорош, а Adam иногда проигрывает, попадая под «неудобные» значения под длину шага. Теперь посмотрим, за сколько и кто сошелся:



$\langle X, 100 \rangle$  - Steps

Adam почти не сходится, если вбирать в анализ то, что у него у единственного максимальное число шагов среди всех остальных. Значит, мы взяли какой-то «неудобный» регион для исследования функции, а это значит, что в любом момент, при немного функции сложнее, он бы не сошел. Gauss-Newton всё также стабилен и силен. А с Dog Leg ситуация также интересна: опять же в районе нуля он делает в среднем количество до схождения больше, чем не в районе нуля - это связано с радиусом доверительного региона, только здесь он решил выбрать слишком маленьким, из-за чего столь большее число шагов.

## BFGS

*BFGS*, или Алгоритм Бройдена - Флетчера - Гольдфарба - Шанно – это тоже оптимизационный итерационный алгоритм для нахождения локального экстремума для не представимых данных или функций в линейном/полиномиальном виде.

Один из известных квазиньютоновских методов (то есть, тех, которые основаны на получении информации о кривизне функции). Как и в случае с Powell Dog Leg, данный метод, в отличие от многих квазиньютоновских, использует аналог довольно медлительного постоянного переопределения Гессiana функции. Но если предыдущий механизм никак не взаимодействовал с явным Гессианом, то BFGS, наоборот, ускоряет работу на порядок: ибо он не явно каждый раз высчитывает матрицу, а лишь приближает к ней значения, при этом посчитав по честному Гессиан лишь один раз.

Рассмотрим идею этого алгоритма. Обозначим за  $x_i = \{x_i^0, x_i^1, \dots, x_i^{n-1}\}$  – координата в пространстве, где  $n$  – размерность соответствующего пространства. Пусть дана нам некоторая функция  $f(x)$  и, как обычно, решаем задачу оптимизации нахождения  $\arg\min_x f(x)$ . Тогда, зададим некую начальную точку  $x_0$  и  $H_0 = B_0^{-1}$  – начальное приближение, где  $B_0^{-1}$  – обратный Гессиан функции, который или может быть посчитан в точке  $x_0$ , или выбран как  $\mathbf{I}$  – обратная матрица. Наконец, сам алгоритм:

- 0) Пусть  $k$  – текущий номер итерации алгоритма.
- 1) Находим точку, в направлении которой будем производить поиск, она определяется следующим образом

$$p_k = -H_k \times \nabla f_k,$$

где здесь и далее  $f_k = f(x_k)$ .

- 2) Вычисляем  $x_{k+1}$  через рекуррентное соотношение следующего вида:

$$x_{k+1} = x_k + \alpha \cdot p_k,$$

где  $\alpha$  – коэффициент, удовлетворяющий условиям Вольфа, которые, напомним, выглядят вот так

$$\begin{aligned} f(x_k + \alpha \cdot p_k) &\leq f(x_k) + c_1 \cdot \alpha \cdot \nabla f_k^T p_k \\ \nabla f(x_k + \alpha \cdot p_k)^T p_k &\geq c_2 \cdot \nabla f_k^T p_k \end{aligned}$$

- 3) Теперь определим размер шага алгоритма после данной итерации и изменение градиента следующими соответствующими образами

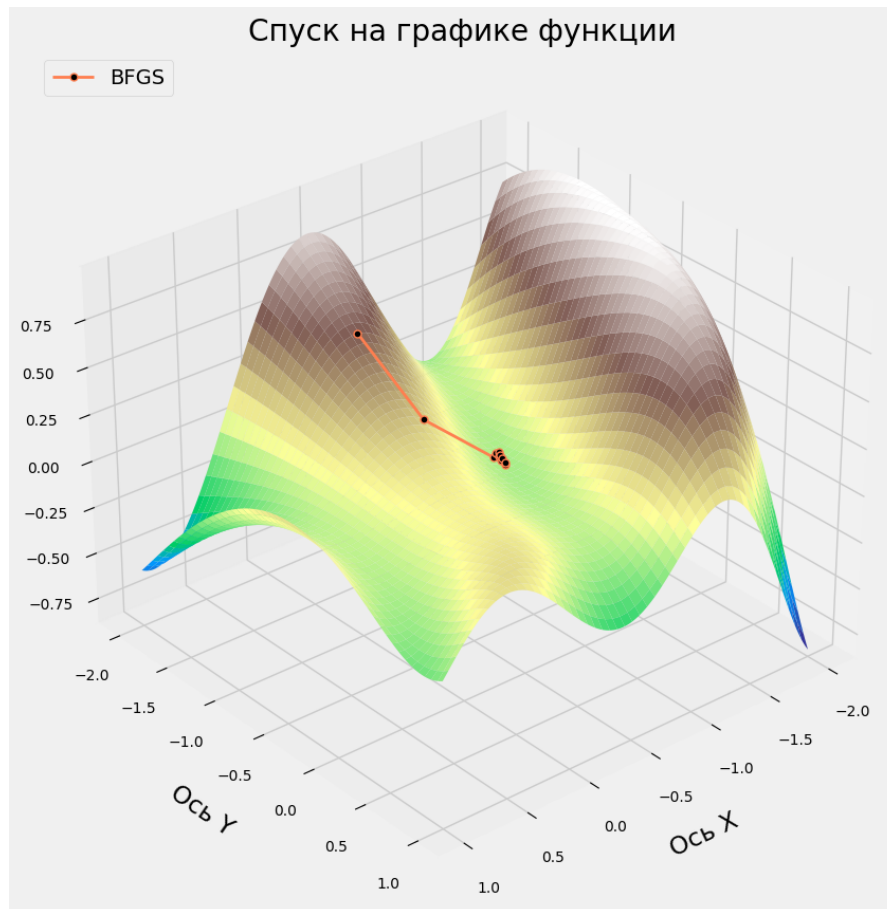
$$\begin{aligned} s_k &= x_{k+1} - x_k \\ y_k &= \nabla f_{k+1} - \nabla f_k \end{aligned}$$

- 4) Наконец, обновим Гессиан функции, зная, что  $\lambda = \frac{1}{y_k^T s_k} \in \mathbb{R}$

$$H_{k+1} = (\mathbf{I} - \lambda s_k y_k^T) H_k (\mathbf{I} - \lambda y_k s_k^T) + \lambda s_k s_k^T$$

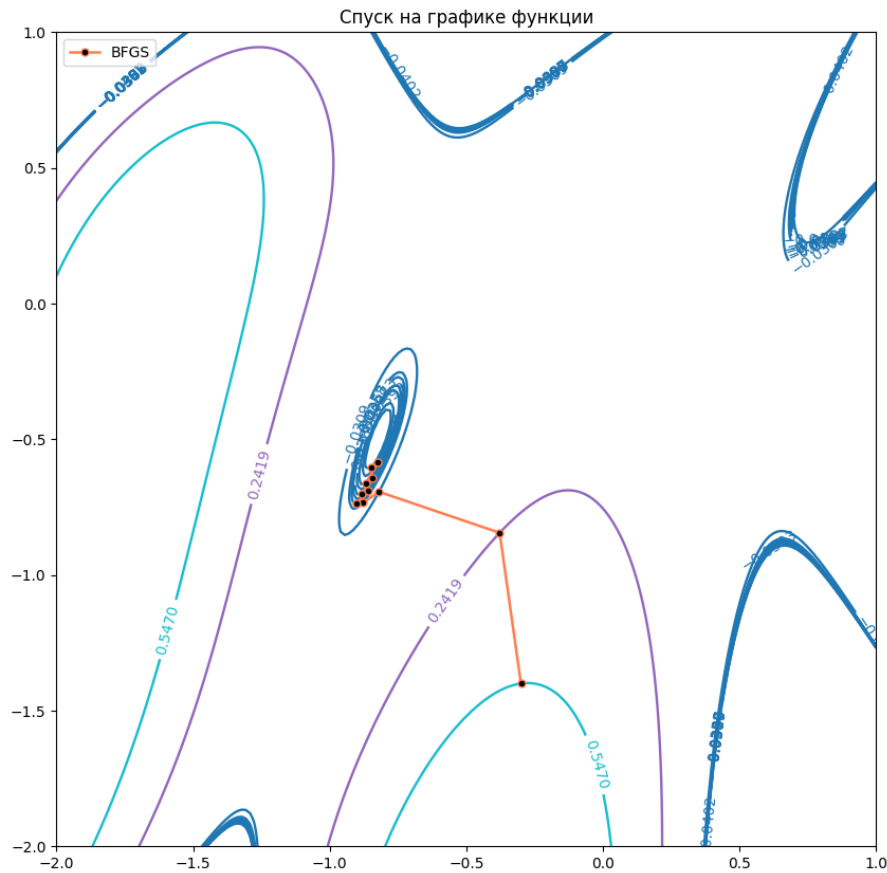
## Пример

Возьмем в качестве примера функцию  $f(x, y) = \sin(0.5x^2 - 0.25y^2 + 3) \cdot \cos(2x + 1 - e^y)$ . В качестве начальной точки возьмем  $\langle x_0, y_0 \rangle = \langle -0.3, -1.3 \rangle$  и запустим зверя.



Example of BFGS

Полученные результаты столь небольшого примера с неприятным минимум следующие. Алгоритм сошелся к предполагаемой точке минимум за 11 шагов, после этого умер своей смертью (условие схождения). Полученная точка  $z_{\min} = -0.040723$ , а его прообразом служит пара из  $\langle -0.848456, -0.605206 \rangle$ . Тот же путь алгоритма в виде линий уровня:



Example of BFGS (lines)

## Исследования

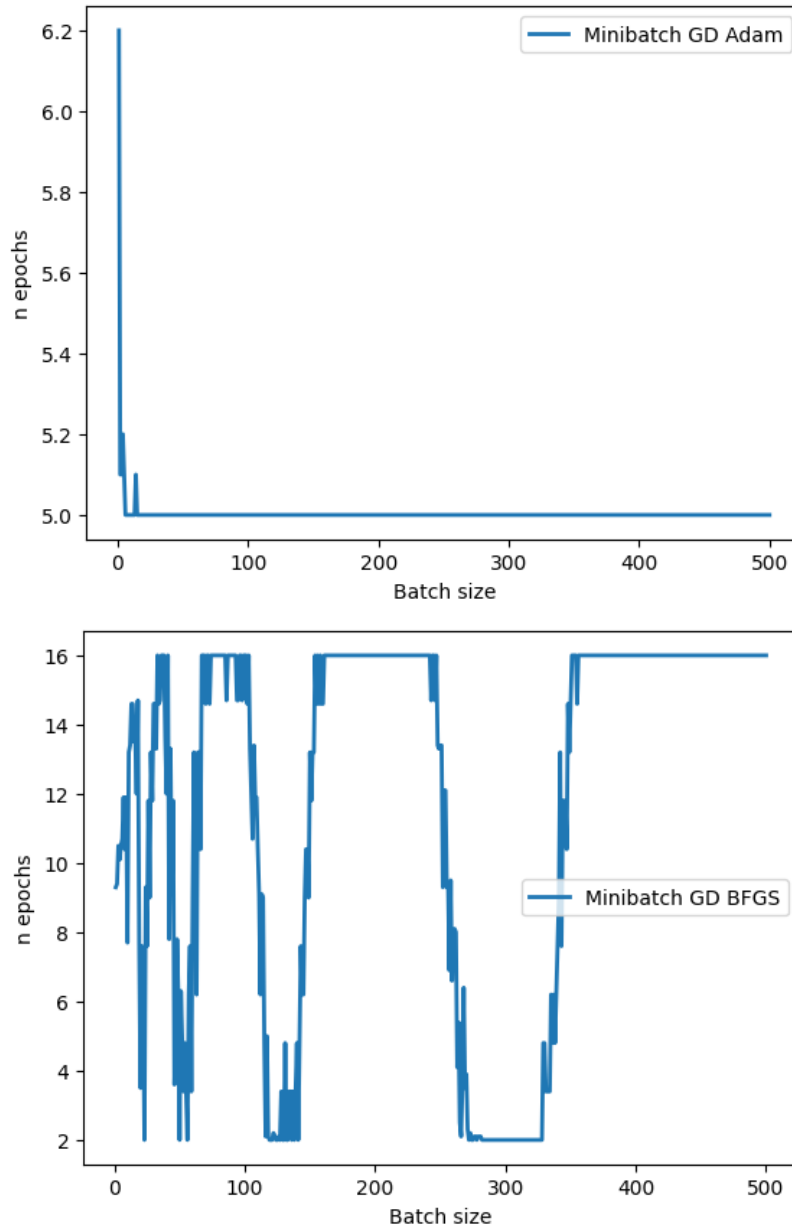
Заметим, что исследования чистой полилинейной регрессии с методом нахождения минимальной или максимальной точки функции выглядит задачей странной, так как там нам нужен сгенерированный dataset, а здесь начальное приближение. Поэтому, в качестве рейтинга  $H(\text{onest})$  за честность, мы воспользуемся BFGS как методом минимизации ошибки квадратичного уравнения. Его соперником выступит ранее известный нам Adam, а все-все исследования будем проводить на, внезапно, очень простой полилинейной функции  $f(x) = 2 \cdot x$ . Зададим следующие настройки генерации точек:

```

dots_count = 500 (количество точек)
variance = 0.5 (вариативность, константа, используемая в формуле генерации)
max_epoch = 16 (максимальное количество точек)
batch_min_size = 0 (минимальный размер батча)
batch_max_size = 500 (максимальный размер батча)

```

Будем проводить исследования, подобно предыдущей лабораторной работы, на раз-  
мере батча  $\text{batch\_size} \in [\text{batch\_min\_size}, \text{batch\_max\_size}]$ . В качестве Adam мы  
возьмем результаты предыдущей лабораторной работы. Итак, полученный график.

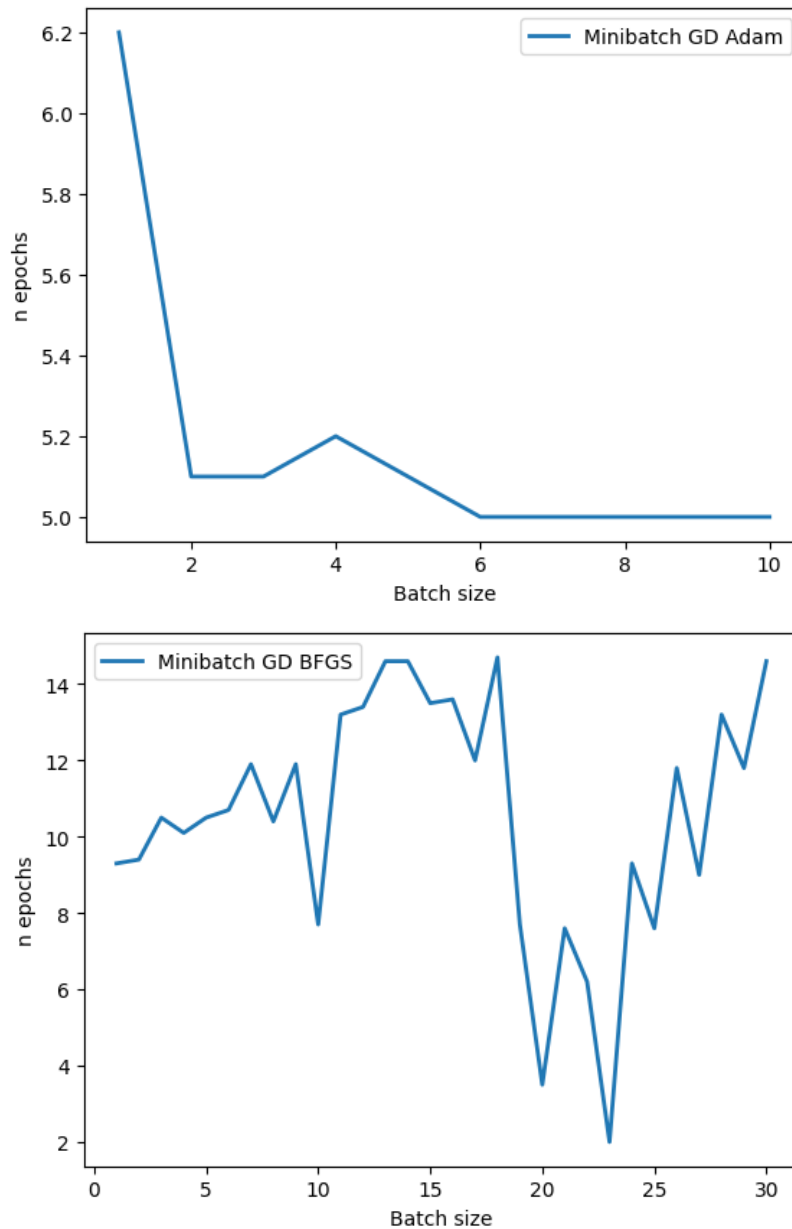


(Minibatch GD) Adam vs. BFGS

Заметим, что в большинстве случаев BFGS, по с течению обстоятельств, проигрывает великолепному Adam, у которого, по предыдущей работе, количество шагов почти никогда не изменялось и оставалось в среднем 5. Также, на некоторых промежутках размеров батча BFGS всё же выигрывает гонку у Adam. Почему это может происходить? На самом деле, проблема кроется в задаче, которую мы поставили перед методами, если Adam идеален в задачах линейной регрессии, то BFGS все-таки хорош во многих нелинейных функциях (смотри пример выше). На примере выше мы видели, что BFGS среди множества похожих окрестностей точек минимума находит тот самый настоящий и остаётся там.

Теперь давайте немного «приблизимся» к батчу  $[0, 10]$  и  $[0, 30]$  и посмотрим, что там происходит. Полученный график.

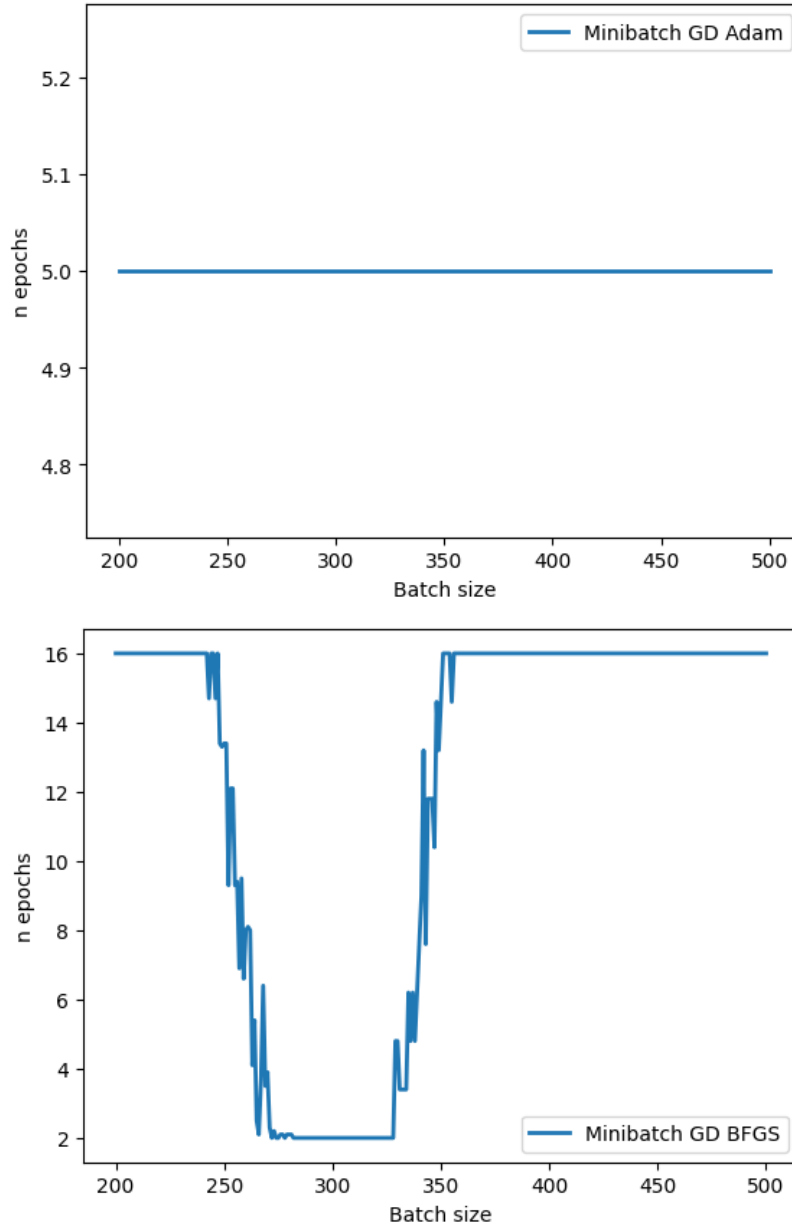




((Minibatch GD) Adam vs. BFGS)  $\in [0, 10]$  и  $\in [0, 30]$

Adam стремительно набирает скорость и падает вниз, в то время как BFGS, невооруженным взглядом заметное, что у него также скачкообразные шаги. При этом мы вновь видим, что при некоторых батчах также выигрывает Adam при двух или трех шагах, в отличие от пяти.

Наконец, рассмотрим повнимательнее отрезок  $[200, 500]$  у обоих методов и проанализируем. Получим графики.



((Minibatch GD) Adam vs. BFGS)  $\in [200, 500]$

Adam не желает перемен, в то время как BFGS пытается прогрессировать и скачкообразными шагами на промежутке от  $[250 + \varepsilon, 325 + \delta]$  и доходит до минимального количества шагов.

Подытожим результаты. Как мы видим, BFGS плохо справляется с минимизацией относительно столь простых функций как прямая  $f(x) = 2x$ . Однако, в тоже время, достаточно неплохо справляется нелинейными функциями, то есть теми, которые нельзя представить как конечный полином.

## L-BFGS

*L-BFGS*, или BFGS с ограниченной памятью – это оптимизационный алгоритм, который аппроксимирует оригинальный алгоритм BFGS с использованием заданного ограниченного объема памяти.

L-BFGS как и BFGS использует приближенную оценку Гесссиана, при этом в явном виде посчитав только один раз, а все остальные шаги лишь преобразовывая. Проблема: BFGS хранит всегда  $n \times n$  приближений к обратному Гесссиану. Решение: хранить несколько векторов, которые неявно представляют приближение, представляющие из себя историю последних  $m$  обновлений положения  $x$  и градиента  $\nabla f(x)$ . При этом,  $m$  обычно выбирается небольшим ( $m < 10$ ).

Рассмотрим идею этого алгоритма. Во многом она будет совпадать с предыдущим, поэтому пропустим обозначения и перейдем сразу алгоритму.

- 0) Пусть  $k$  – текущий номер итерации алгоритма, здесь и далее будем иметь ввиду  $g_k = \nabla f(x_k)$ .

- 1) Также как и в BFGS находим точку, в направлении которой будем производить поиск:

$$p_k = -H_k \times \nabla f_k,$$

здесь и далее  $f_k = f(x_k)$ .

- 2) Пусть мы сохранили  $m$  обновлений вида:

$$s_k = x_{k+1} - x_k$$

$$y_k = g_{k+1} - g_k$$

Заметим, что при последующих итерациях алгоритма  $k \geq m$  произведение из первого шага можно получить выполнив последовательность скалярных произведений и суммирования векторов, включающую  $\nabla f_k$  и пары  $\{s_i, y_i\}$ . После вычисления новой итерации самая старая пара векторов в наборе пар  $\{s_i, y_i\}$  заменяется новой парой, который получается из данного шага.

- 3) Наконец, пожалуй, самая идейная часть – обновление Гесссиана. На итерации  $k$  у нас определен  $x_k$  и пары  $\{s_i, y_i\} \forall i \in [k-m, k-m+1, \dots, k-1]$ . Выберем некоторое начальное Гесссианское приближение  $H_k^0$  (в отличие от стандартной итерации BFGS, это начальное приближение может меняться от итерации к итерации) и путем повторного применения формулы, заданной изначально в BFGS, получаем

$$\begin{aligned} H_k &= (V_{k-1}^T \cdot \dots \cdot V_{k-1}^T) H_k^0 (V_{k-m} \cdot \dots \cdot V_{k-1}) \\ &+ \rho_{k-m} (V_{k-1}^T \cdot \dots \cdot V_{k-m+1}^T) s_{k-m} s_{k-m}^T (V_{k-m+1} \cdot \dots \cdot V_{k-1}) \\ &+ \rho_{k-m+1} (V_{k-1}^T \cdot \dots \cdot V_{k-m+2}^T) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \cdot \dots \cdot V_{k-1}) \\ &+ \dots \\ &+ \rho_{k-1} s_{k-1} s_{k-1}^T, \end{aligned}$$

где  $\rho_k = \frac{1}{y_k^T s_k}$ ,  $V_k = \mathbf{I} - \rho_k y_k s_k^T$ . Из всего вышеописанного мы можем провести произведение  $H_k \times \nabla f_k$  более эффективно следующим образом

```

 $\alpha \leftarrow [0] * \text{sizeof } s;$ 
 $q \leftarrow \nabla f_k;$ 
 $\text{forall } i = k - 1, \dots, k - m \text{ do}$ 
     $\alpha_i \leftarrow \rho_i s_i^T q;$ 
     $q \leftarrow q - \alpha_i y_i;$ 
 $\text{end}$ 
 $r \leftarrow H_k^0 q;$ 
 $\text{forall } i = k - m, \dots, k - 1 \text{ do}$ 
     $\beta_i \leftarrow \rho_i y_i^T r;$ 
     $r \leftarrow r + s_i(\alpha_i - \beta_i);$ 
 $\text{end}$ 
 $\text{return } (-1) \cdot r \leftarrow \llbracket \equiv H_k \times \nabla f_k \rrbracket;$ 

```

Получение нового на данной итерации  $H_k^0$  мы также сильно ускорим, лишь приблизив наши значения, используя формулу  $H_k^0 = \gamma_k \mathbf{I}$ , где

$$\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$$

4) Все последующие шаги аналогичны с оригинальным BFGS.

Итак, напомним идейный псевдокод алгоритма L-BFGS.

```

1 function is_convergence(f:  $\mathbb{R}^n \rightarrow \mathbb{R}$ ):
2     /*implementation defined*/
3
4 function Wolfe_coefficient(f:  $\mathbb{R}^n \rightarrow \mathbb{R}$ ,  $p_k$ ):
5     /*implementation defined*/
6
7 function get_prod(f:  $\mathbb{R}^n \rightarrow \mathbb{R}$ ):
8      $\alpha \leftarrow [0] * \text{sizeof } s;$ 
9      $q \leftarrow \nabla f_k;$ 
10     $\forall i = k - 1, \dots, k - m \text{ do}$ 
11         $\alpha_i \leftarrow \rho_i s_i^T q;$ 
12         $q \leftarrow q - \alpha_i y_i;$ 
13    end
14     $r \leftarrow H_k^0 q;$ 
15     $\forall i = k - m, \dots, k - 1 \text{ do}$ 
16         $\beta_i \leftarrow \rho_i y_i^T r;$ 
17         $r \leftarrow r + s_i(\alpha_i - \beta_i);$ 
18    end
19    return  $(-1) \cdot r \llbracket \equiv H_k \times \nabla f_k \rrbracket;$ 
20
21 function LBFGS():
22      $x_0 \leftarrow \text{INIT};$ 
23      $m \leftarrow i \in [5, 10];$ 
24      $q \leftarrow []$ 
25     while !is_convergence(f) do
26          $H_k^0 \leftarrow \left( \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}} \right) \mathbf{I};$ 
27          $p_k \leftarrow \text{get\_prod}(f);$ 
28          $\alpha_k \leftarrow \text{Wolfe\_coefficient}(f, p_k);$ 
29          $x_{k+1} \leftarrow x_k + \alpha_k p_k$ 

```

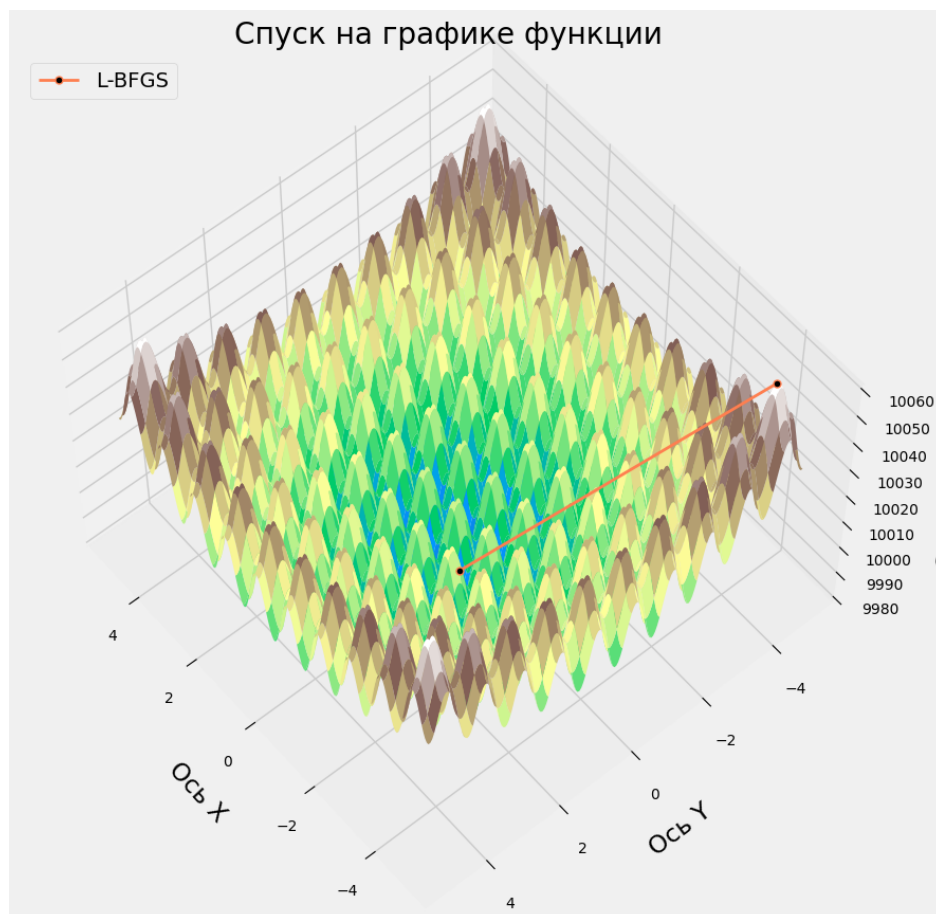
```

30     if k > m then
31         q.remove({sk-m, yk-m});
32     sk ← xk+1 - xk;
33     yk ← ∇fk+1 - ∇fk;
34     q.append({xk, yk});
35

```

## Пример

Возьмем в качестве примера функцию Растригина, описываемая как  $f(\vec{x}) = 10n + \sum_{i=1}^n (x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i))$ , где  $-5.12 \leq x_i \leq 5.12$ ,  $n$  – исследуемая размерность. Здесь глобальный минимум  $f(0, \dots, 0) = 0$ . В качестве начальной точки возьмем далеко от минимума и запустим зверя.

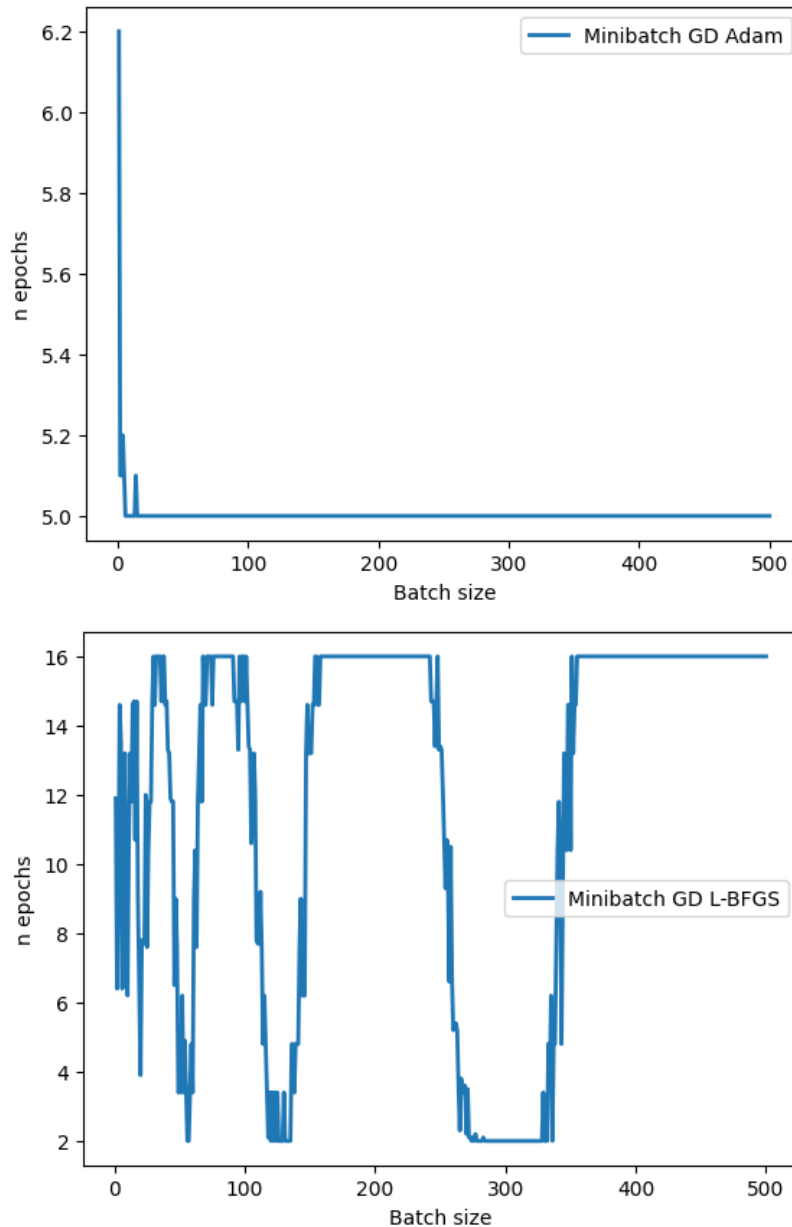


Example of L-BFGS

— "Туда нам надо — сказал L-BFGS и сделал один длинный шаг ровно в минимум. Полученная точка примерно такая же, что и заявленный минимум.

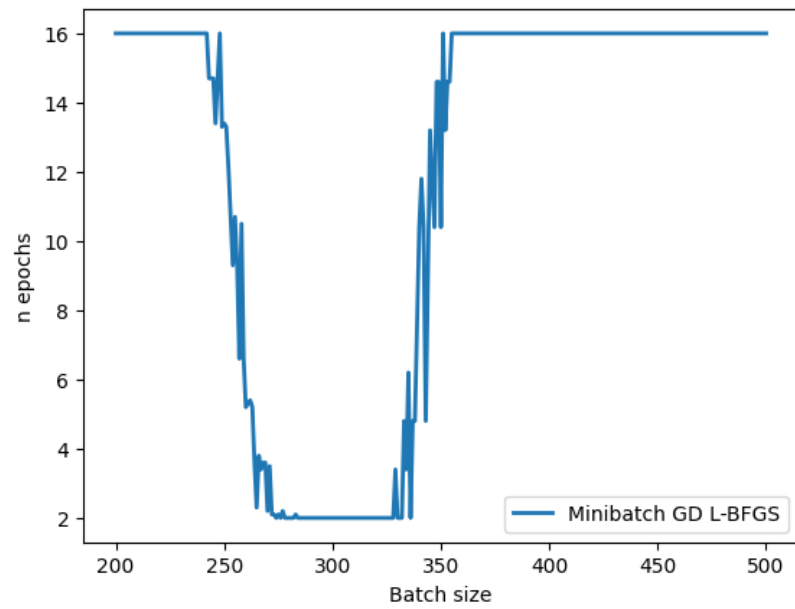
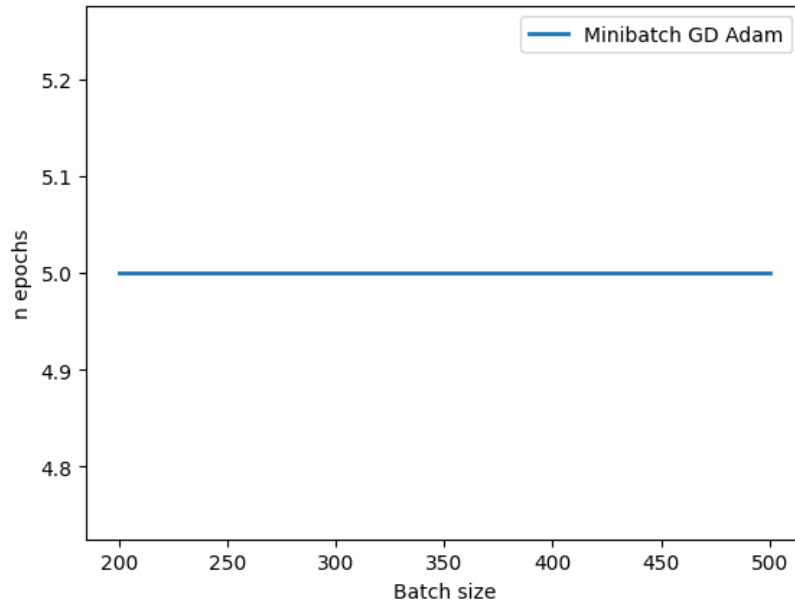
## Исследования

В качестве исследования здесь мы проведем аналогичные как и в BFGS. То есть, та же функция  $f(x) = 2x$ , и Adam как метод минимизации для линейной регрессии. Итак, зададим те же настройки и получим полный график по батчам  $[0, 500]$ .



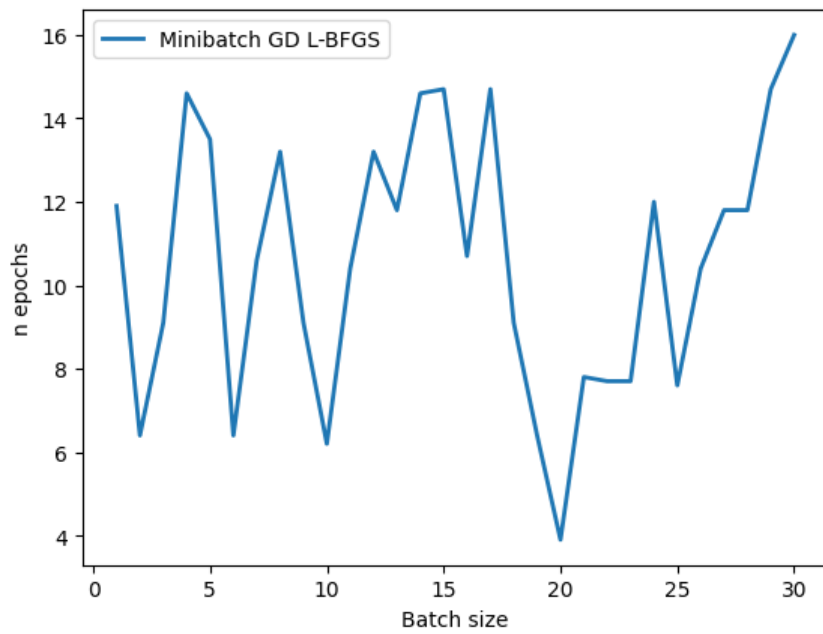
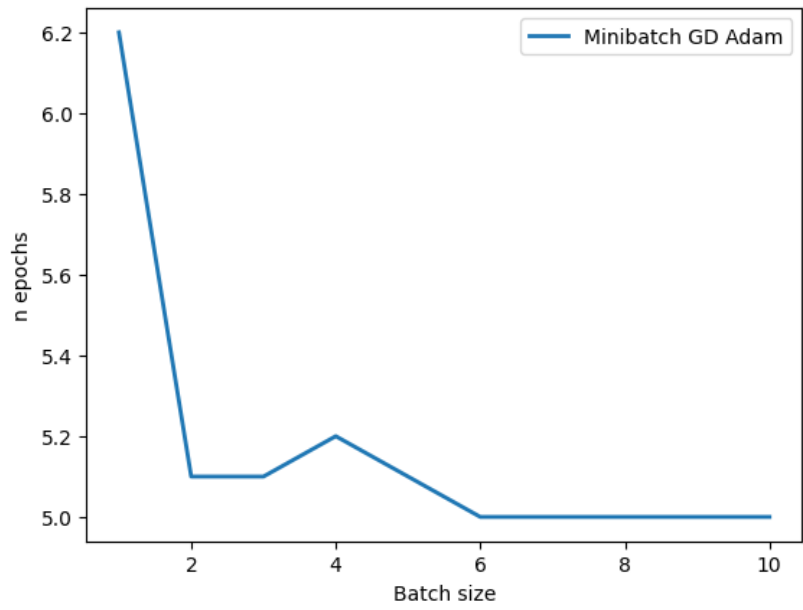
(Minibatch GD) Adam vs. L-BFGS

Получаем аналогичные результаты, что и предыдущий пациент. Связано это с тем, что там и здесь используется одна и та же идея, и, по сути, разница лишь в использовании памяти, которую BFGS жрет как не в себя. Аналогично рассмотрим наши результаты поближе на отрезке  $[200, 500]$ :



((Minibatch GD) Adam vs. L-BFGS)  $\in [200, 500]$

И, наконец, приближении в отрезок  $[0, 10]$  и  $[0, 30]$  в случае с LBFGS:



$((\text{Minibatch GD}) \text{ Adam vs. L-BFGS}) \in [0, 10] \text{ и } \in [0, 30]$