

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

Методы оптимизации

Отчёт по лабораторной работе №2

Работу выполнили:
Щербина К. А.
Чеканова П. А.
Викулаев И.А.

Преподаватель:
Станислав Ким

Санкт-Петербург
2023г.

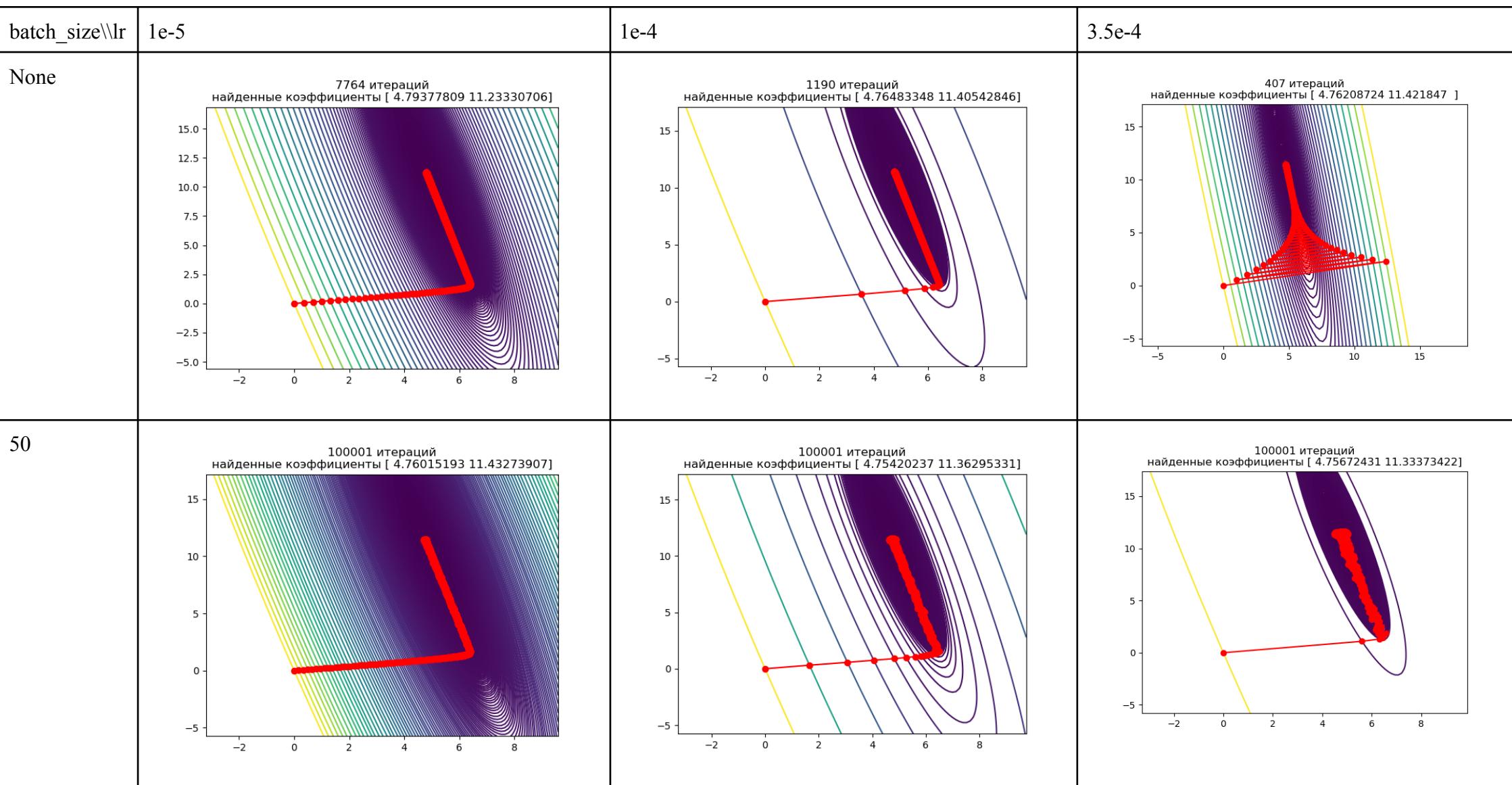
Задача 1.

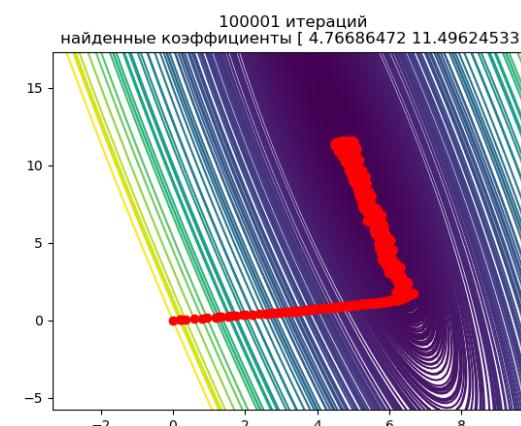
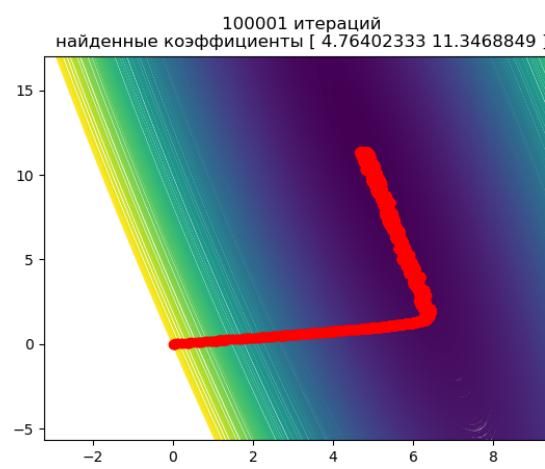
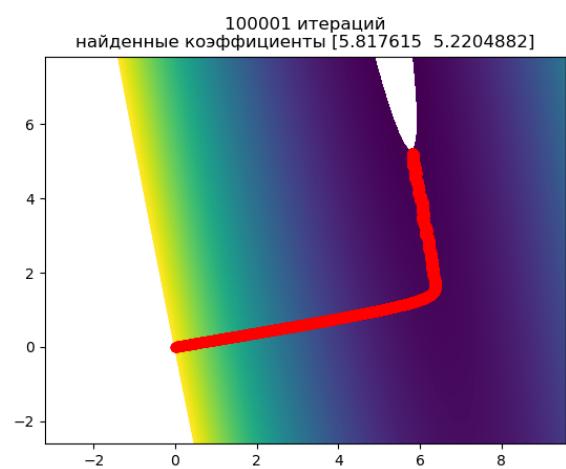
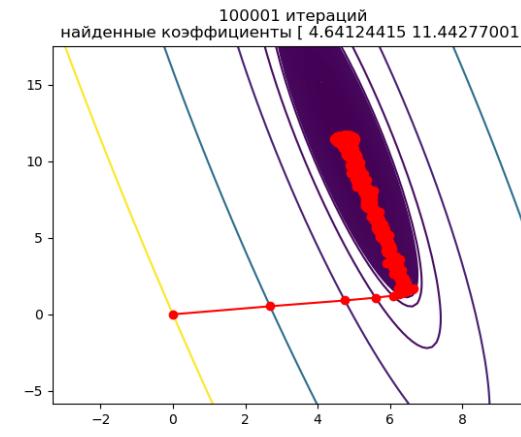
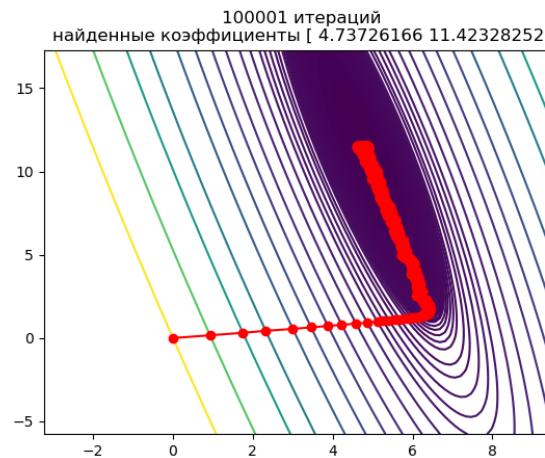
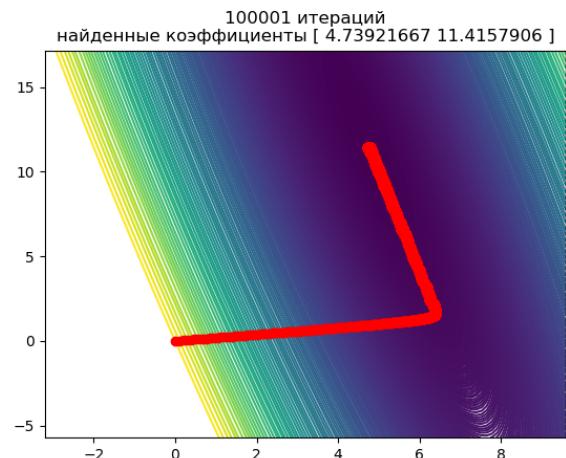
Линейная регрессия - один из методов построения функции прогнозирования числовых зависимостей данных. Данные должны представлять себе полностью заполненный список, каждая строка которого - n переменных. По сути каждую строчку списка можно представить как точку в n мерном пространстве. Наша задача состоит в том, чтобы построить функцию, которая сможет предсказывать расположение новых, не известных нам точек. линейная регрессия представляет собой следующую функцию:

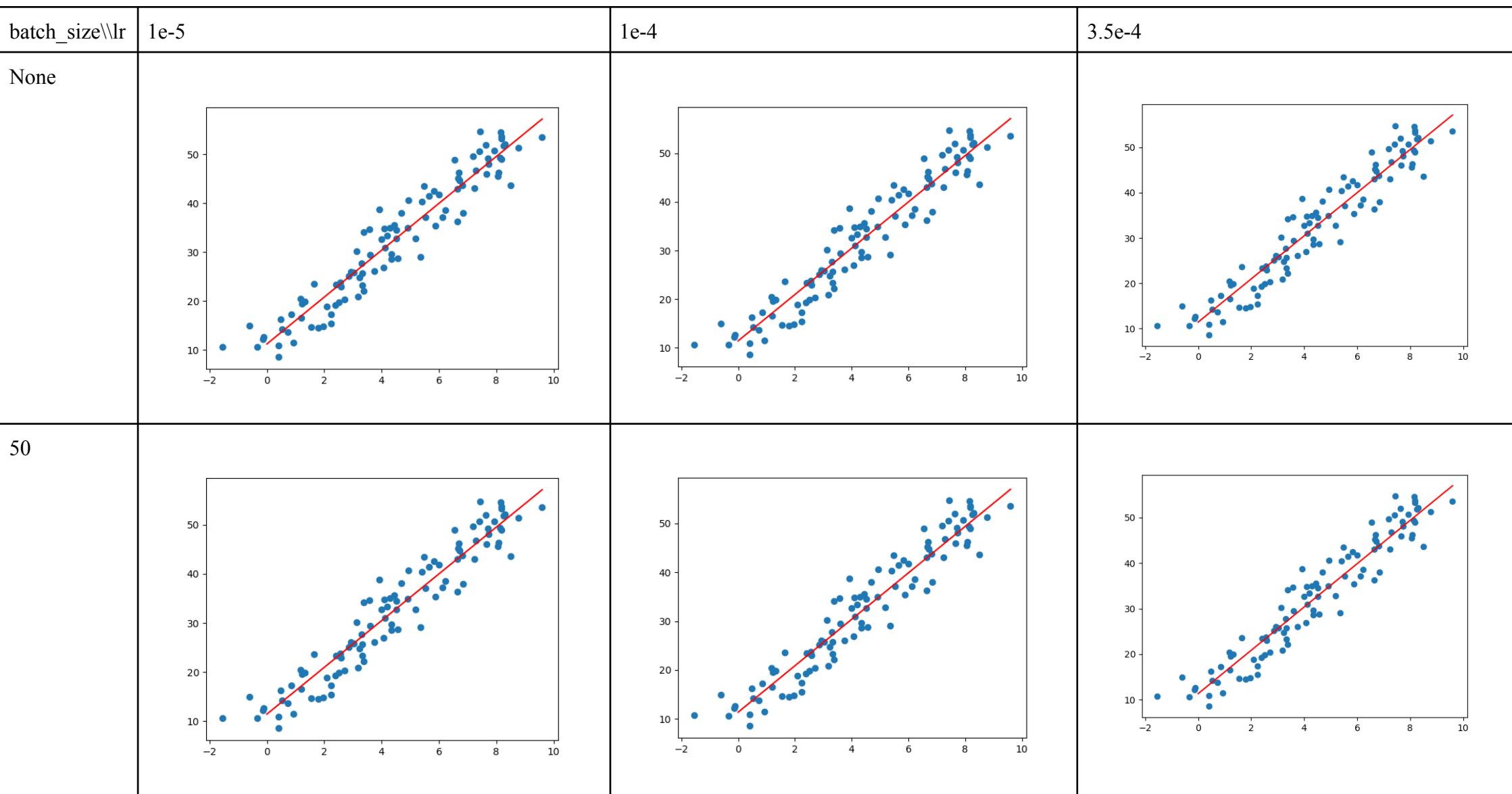
$$f(x) = \sum_{i=1}^{n-1} w_i x_i + w_0$$

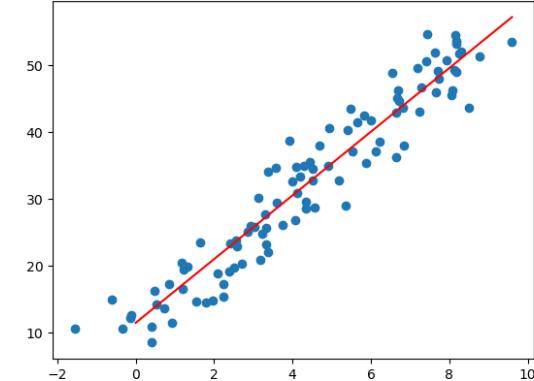
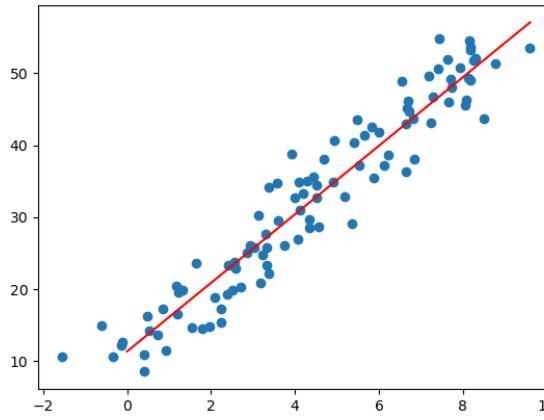
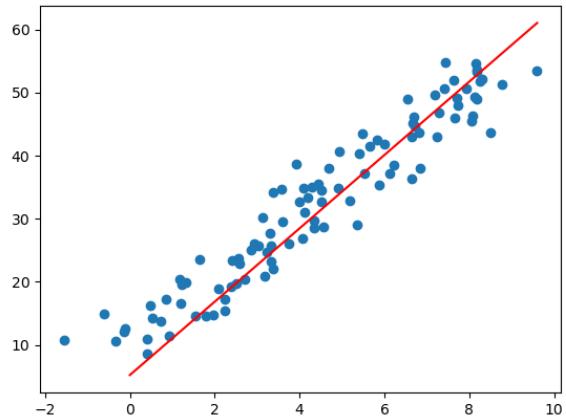
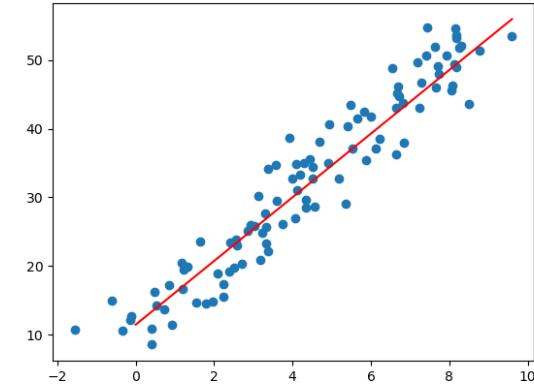
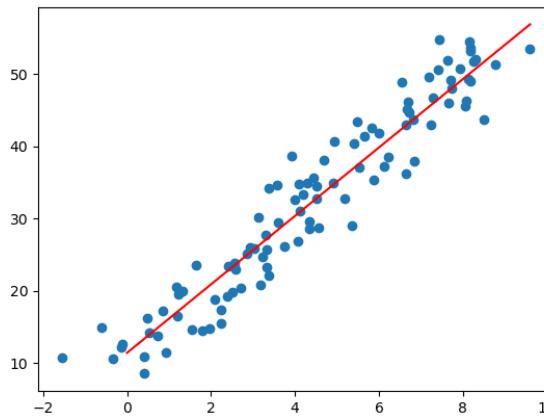
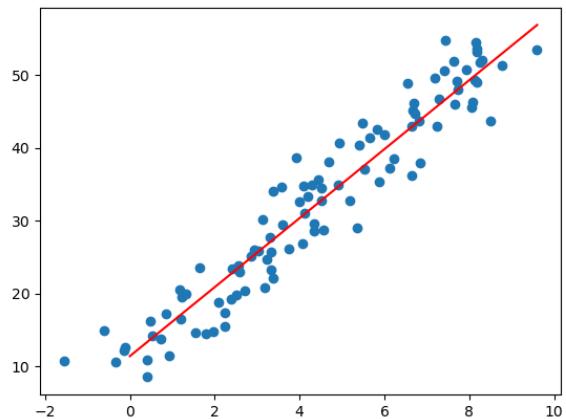
Задача состоит в том, чтобы подобрать такие коэффициенты w, что некоторая функция ошибки будет минимальной. В нашей имплементации метода мы выбрали среднеквадратичное отклонение.

Главные отличия GD и minibatch GD в том, что GD будет использовать весь массив данных для поиска минимума функции потерь, в то время как minibatch возьмут некоторое фиксированное по размеру подмножество этих данных. Если размер этого множества будет равен 1, то мы будем работать с стохастическим градиентным спуском.



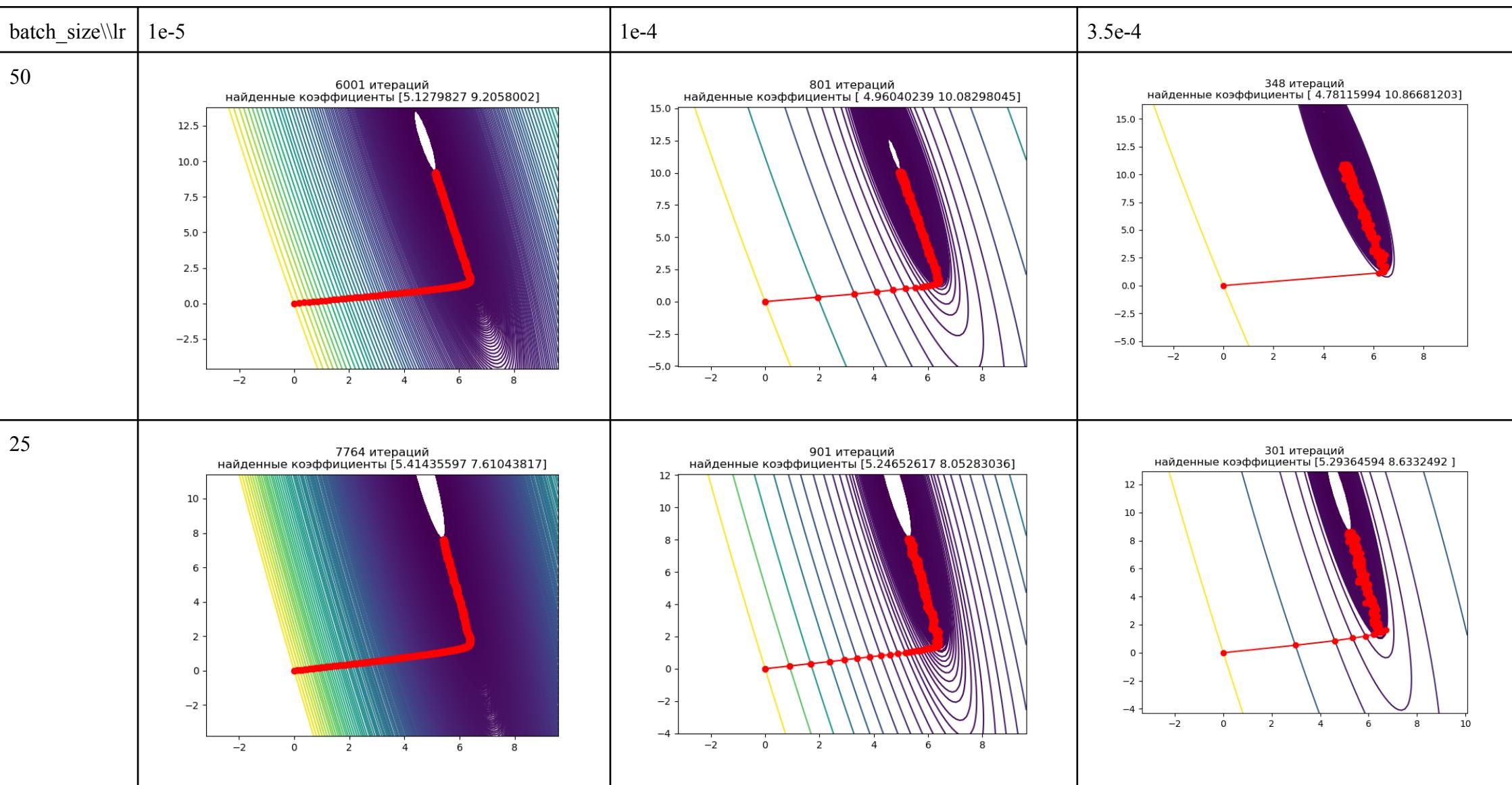


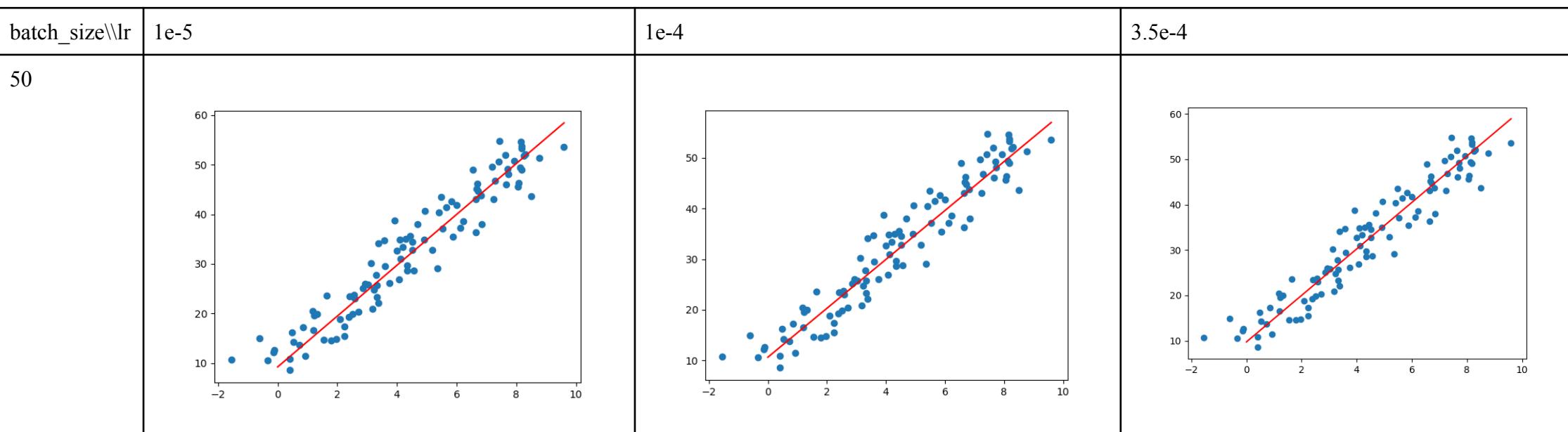
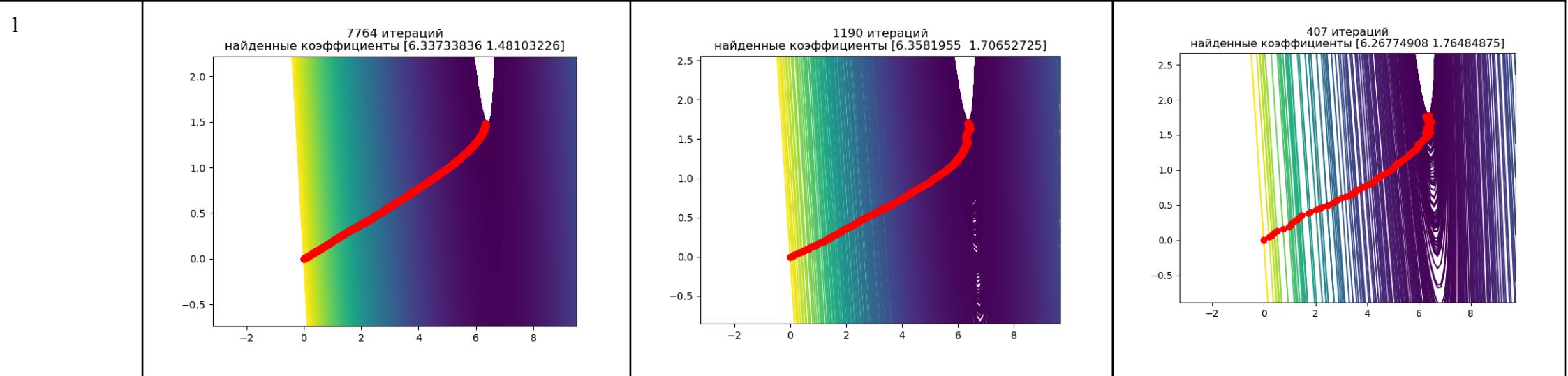


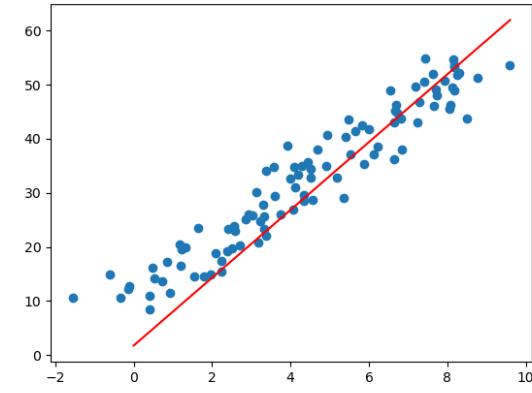
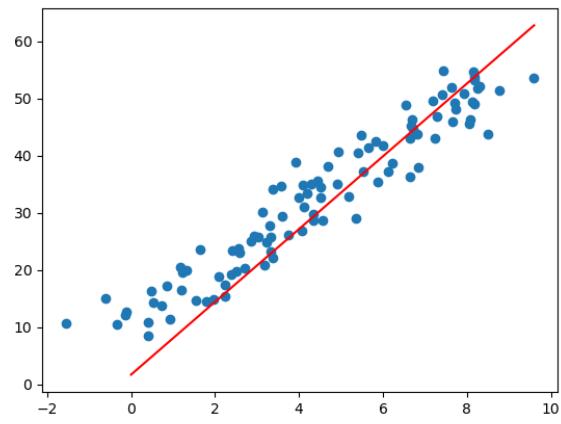
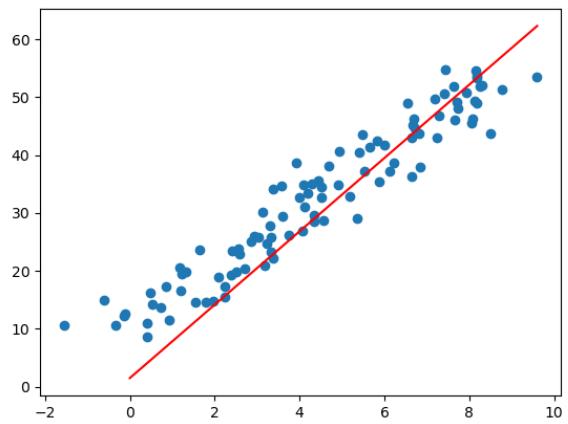
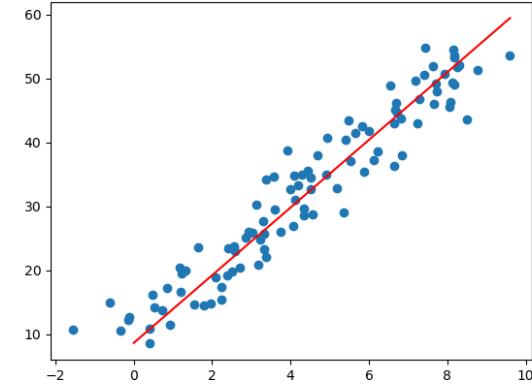
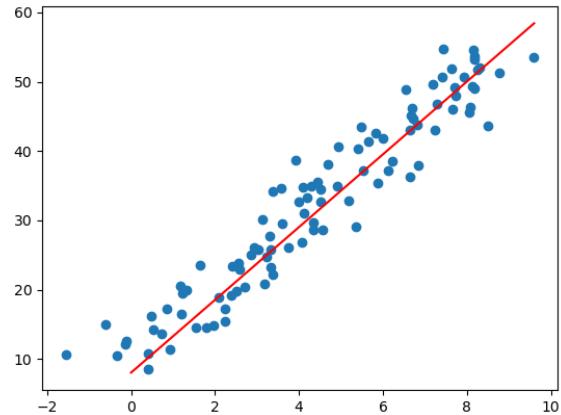
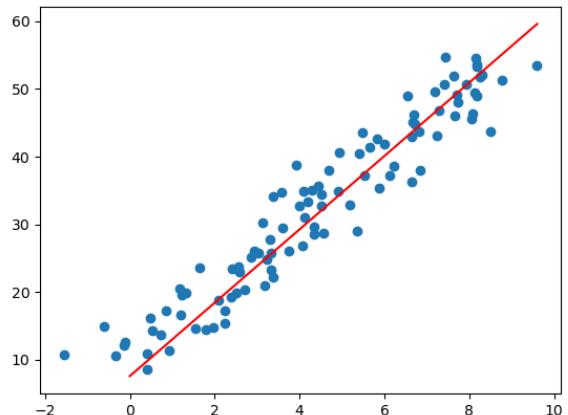


Несложно заметить, что SGD и Minibatch GD не сошлись, но получили ответ не сильно хуже, чем обычный градиентный спуск. Отметим, что при внимательном рассмотрении траекторий сходимости, градиентный спуск на меньших размерах батча начинает ходить в некоторой окрестности минимума. Поэтому выставление слишком большого параметра на `max_epochs` может быть не всегда хорошей идеей. Например, как уже сказано выше, в приведённых экспериментах над `data2.csv` мы видим достаточно небольшое отклонение результатов SGD и Minibatch GD от наиболее точного результата обычного градиентного спуска. Отчего мы можем допустить уменьшение максимального количества итераций на несколько порядков и получить достаточно неплохой результат.

Проведём серию экспериментов для подтверждения этого утверждения и поставим ограничение на `max_epochs` число меньшее или равное, чем количество итераций обычного градиентного спуска.





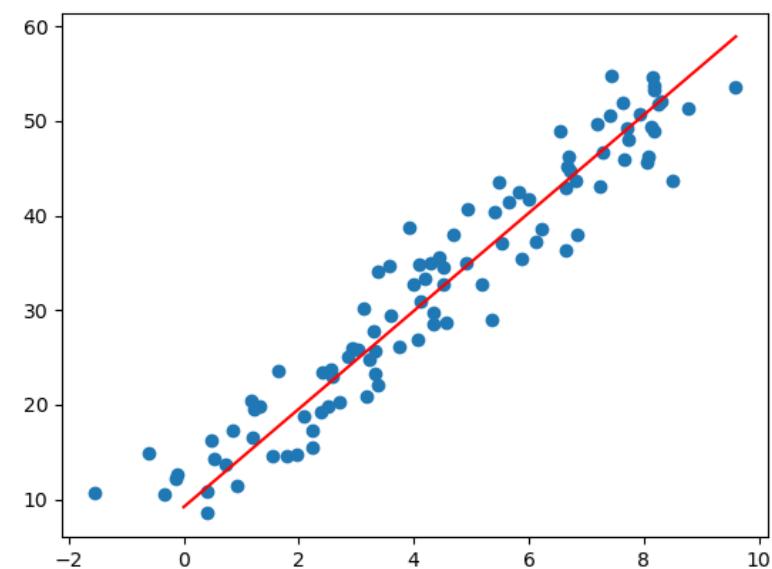
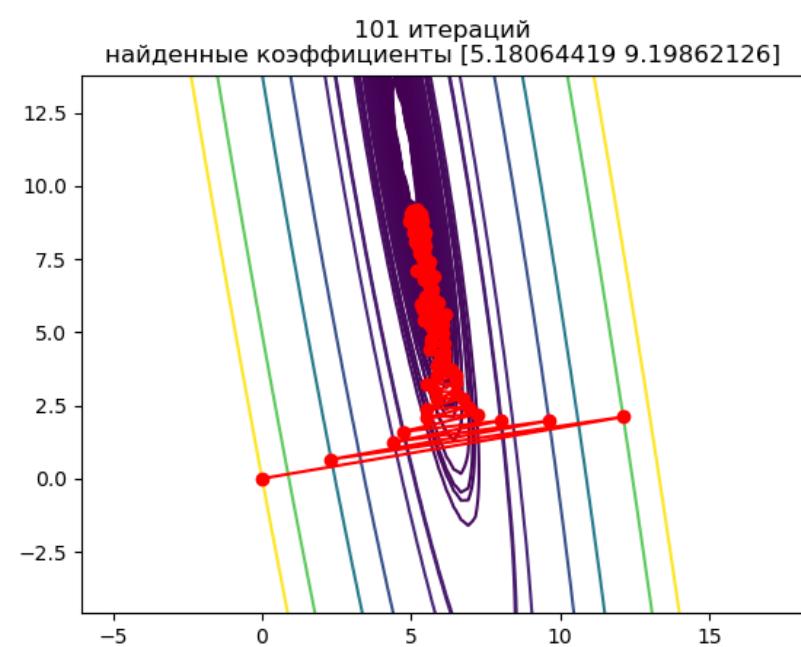


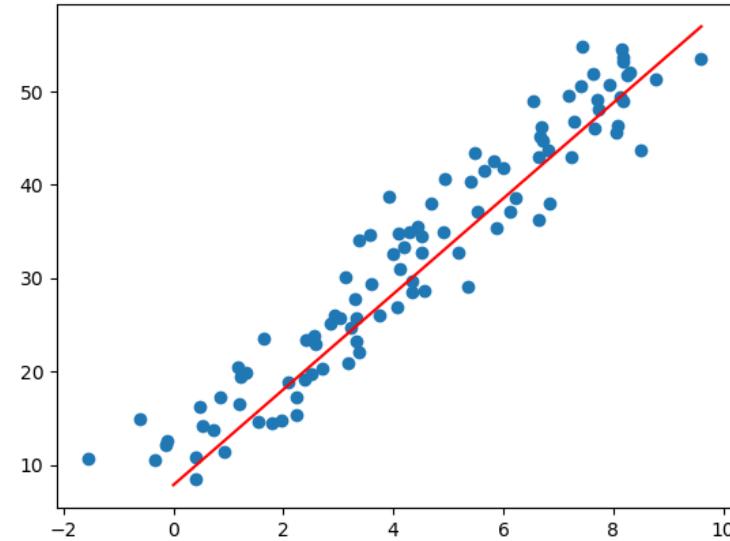
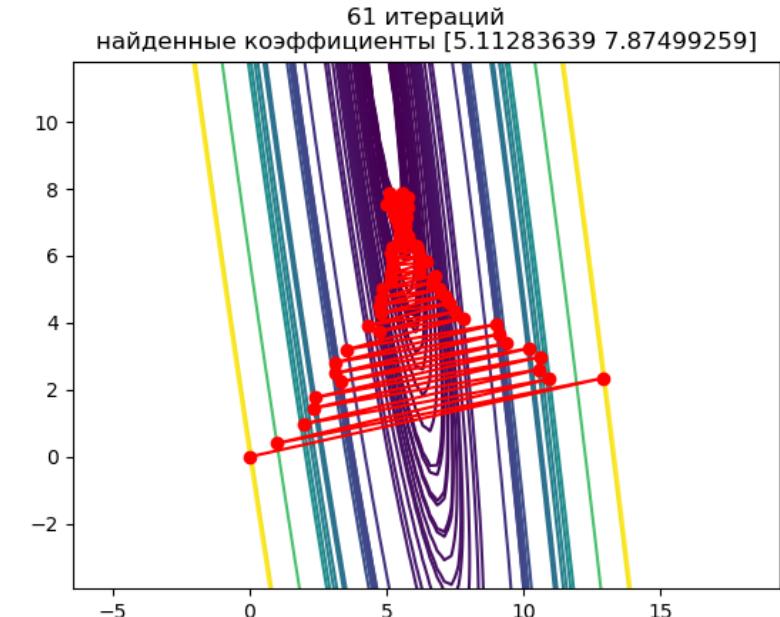
Заметим, что оптимальный размер батча находится выше 50, но меньше, чем n . В ходе экспериментов мы выставляли достаточно малые коэффициенты обучения, ввиду переполнения типов данных. Однако мы видим, что есть возможность для меньших размерах батча выставить коэффициент чуть выше, чтобы потенциально повысить эффективность. Для этого рассмотрим более высокий размер батча и коэффициент обучения.

batch_size\lr

6e-4

50





Таким образом получается, что при правильном выборе размера батча и ставя как предельное количество итераций - итерации градиентного спуска или меньше, мы можем в разы уменьшить количество итераций необходимых для определения весов линейной регрессии. Однако руками перебирать различные предельные значения итераций не хочется. Нужно как-то автоматизировать этот процесс. Для этого рассмотрим 2-ой пункт лабораторной работы.

Задача 2

Перед нами стоит задача реализации функции изменения шага (learning rate scheduling), чтобы улучшить сходимость.

Для улучшения сходимости стохастического градиентного спуска мы использовали экспоненциальную (exponential decay) функцию изменения шага, которая уменьшает шаг градиентного спуска экспоненциально с течением времени.

Идея заключается в том, что на начальных этапах обучения мы можем использовать более высокую скорость обучения, чтобы быстрее достигнуть локального минимума функции потерь. Однако с течением времени модель уже будет находиться ближе к этому минимуму, и использование высокой скорости обучения может привести к тому, что модель будет «скакать» между значениями параметров и не будет сходиться к оптимальным значениям. Поэтому вместо постоянной скорости обучения мы будем постепенно уменьшать ее со временем, чтобы уменьшить скорость обучения по мере того, как модель приближается к локальному минимуму.

Формула для этого метода выглядит следующим образом:

$$lr = lr_0 * e^{-kt}$$

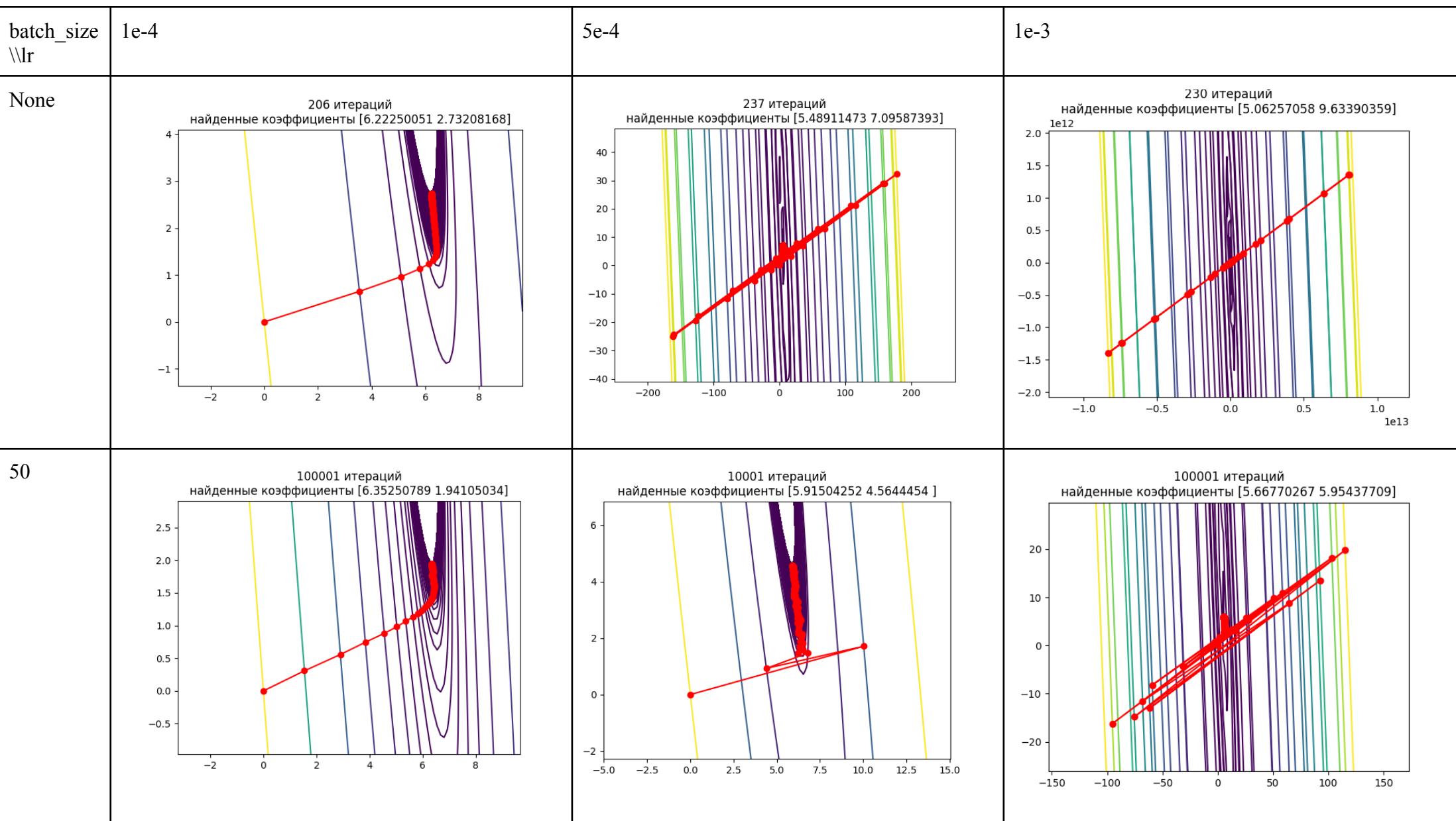
где lr - текущий шаг градиентного спуска, lr_0 - начальный шаг, t - номер итерации (текущая эпоха), k - коэффициент затухания.

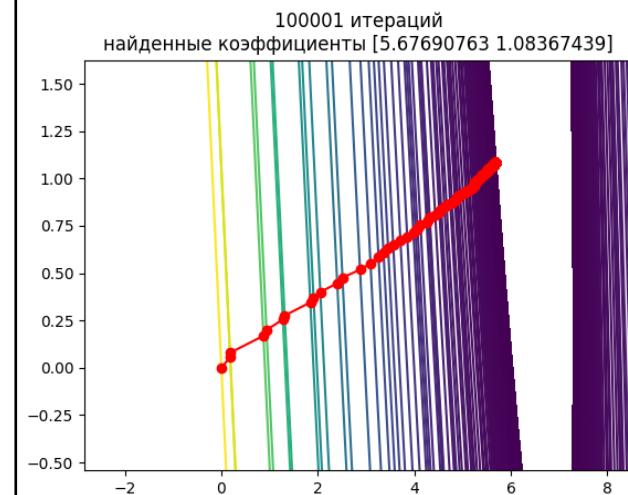
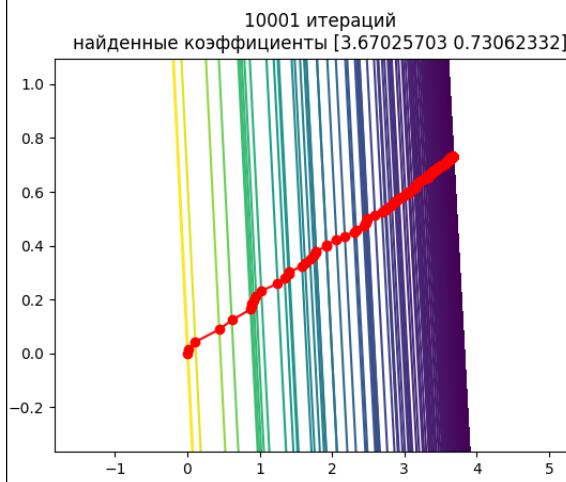
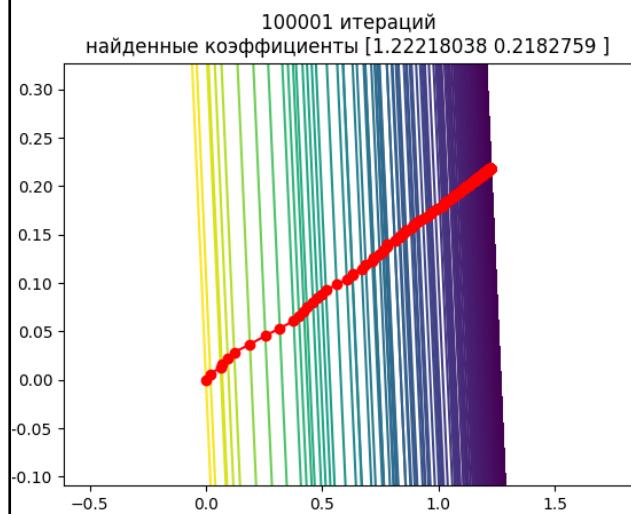
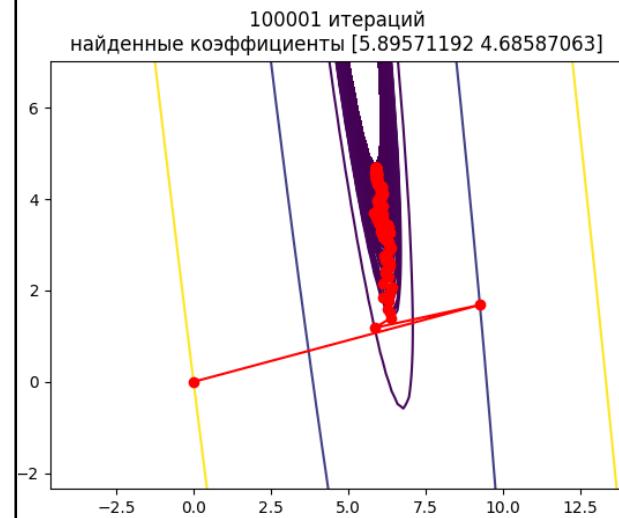
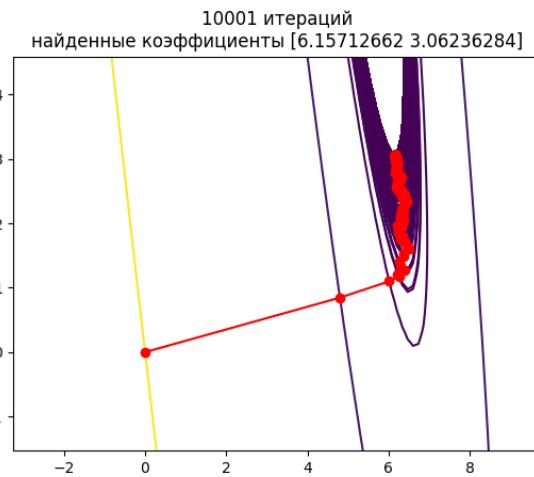
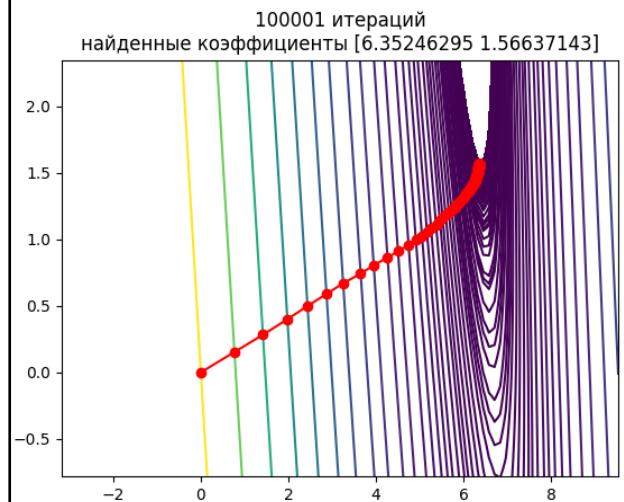
Коэффициент затухания k определяет, как быстро скорость обучения должна уменьшаться. Чем больше k , тем быстрее скорость обучения уменьшается со временем.

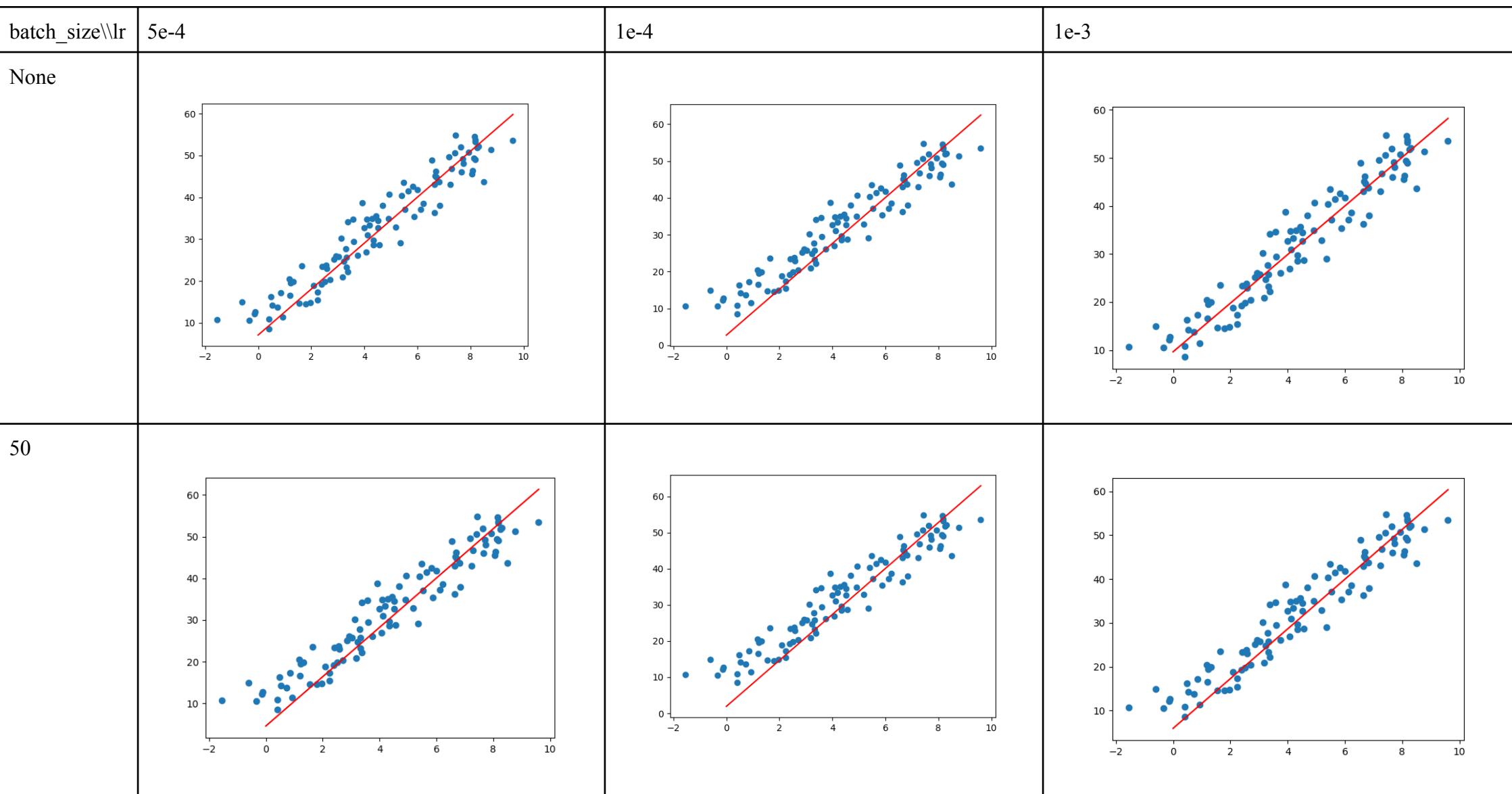
Экспоненциальное затухание позволяет медленно уменьшать шаг градиентного спуска по мере увеличения количества итераций. Это позволяет модели сходиться более плавно и уменьшает вероятность расхождения. Однако, если коэффициент затухания выбран неправильно, то шаг градиентного спуска может стать слишком маленьким и модель может сходиться очень медленно.

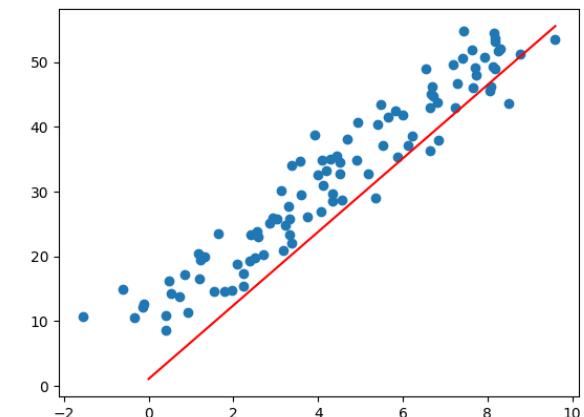
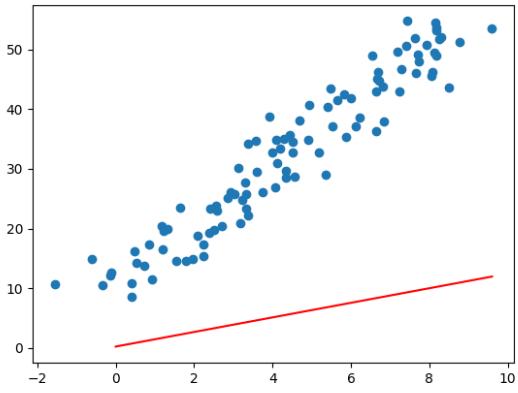
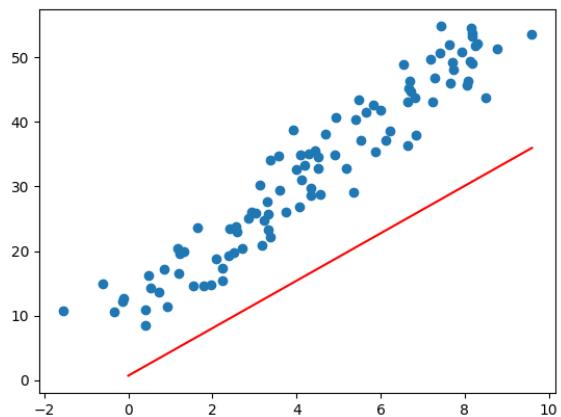
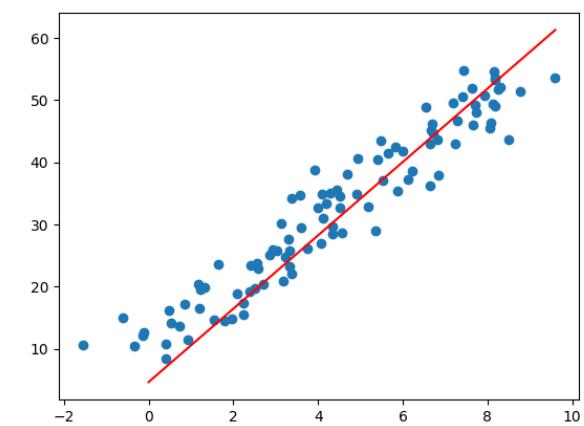
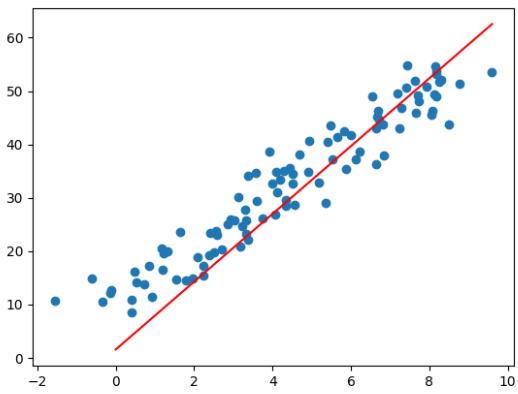
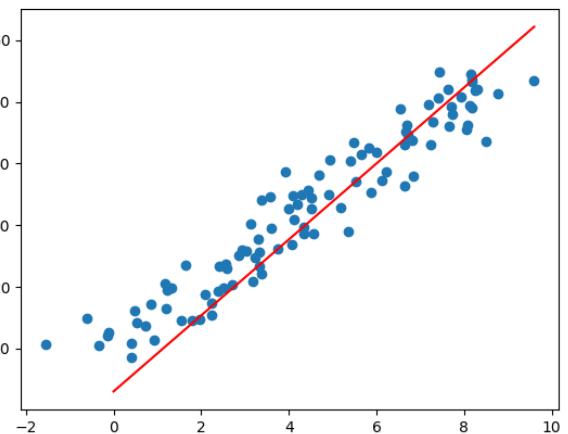
В общем, экспоненциальное затухание может быть удобнее в использовании, однако менее эффективным в более сложных задачах.

Проведём серию экспериментов, для иллюстрации работы нашей функции изменения шага. Возьмём $k = 0,03$.

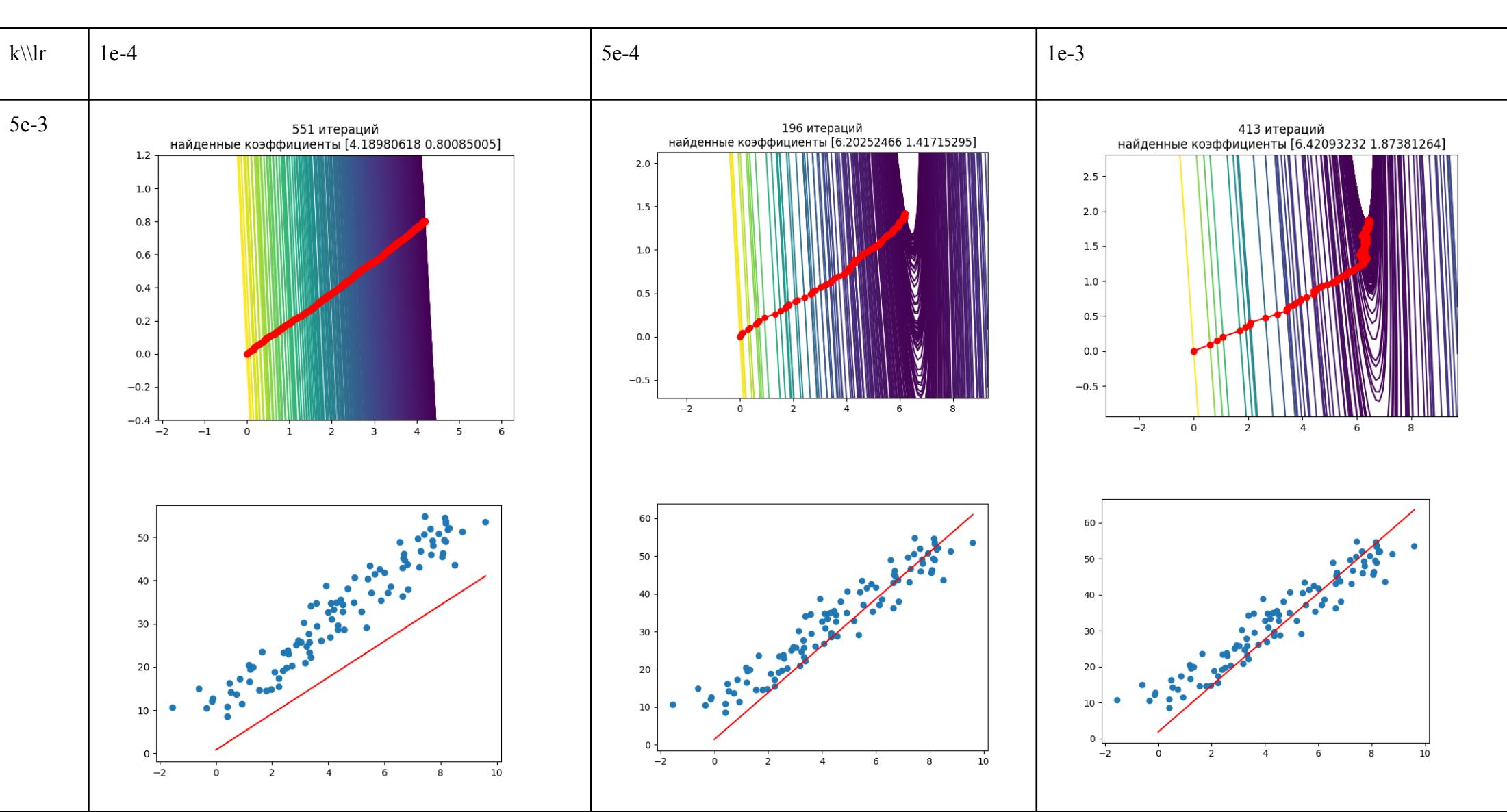


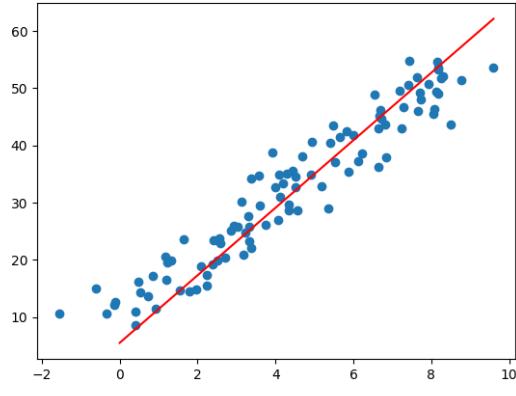
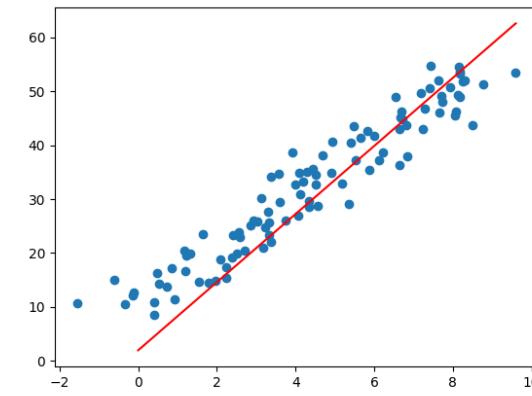
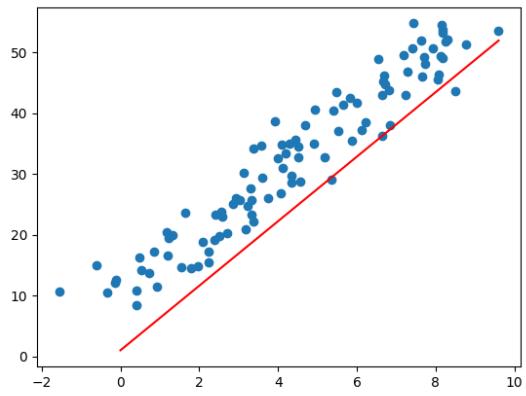
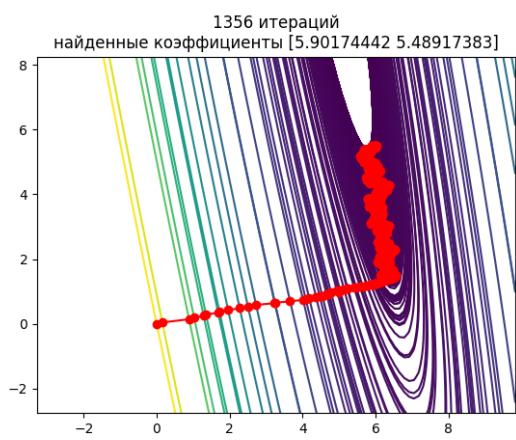
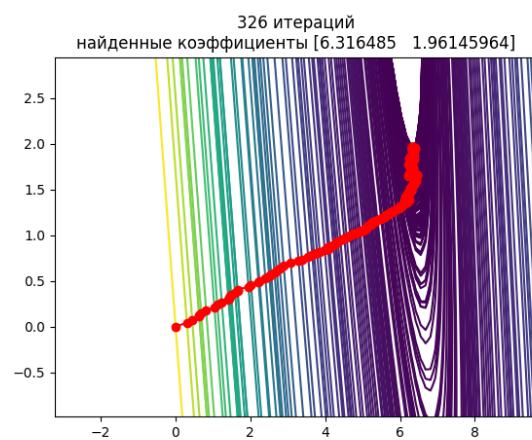
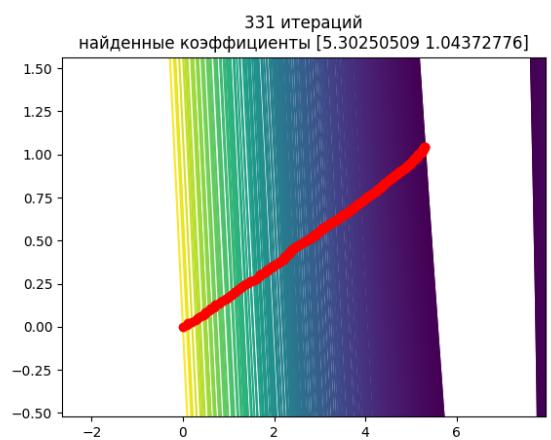


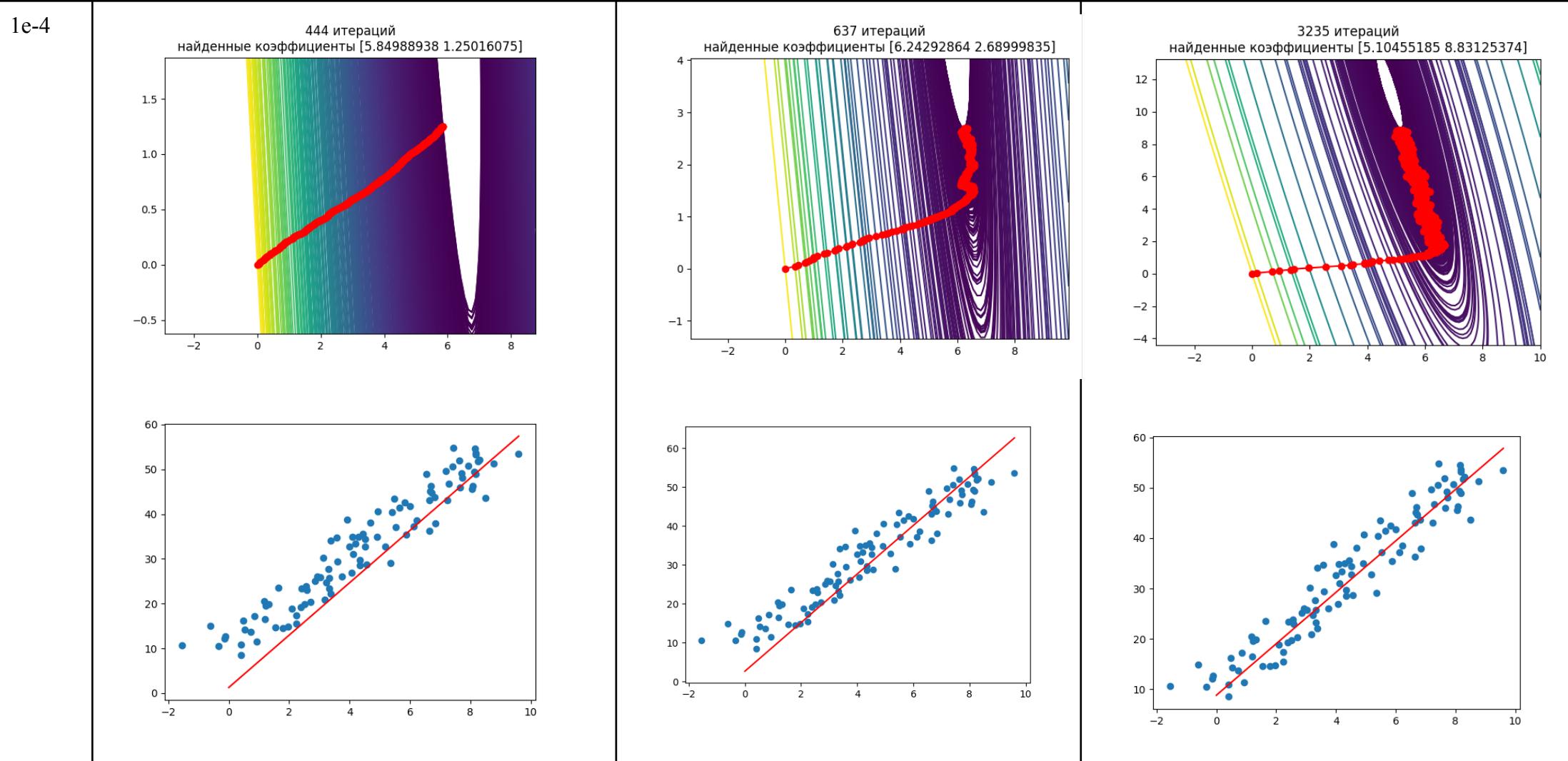




Несложно заметить, что SGD и Minibatch GD не сошлись. Данные графики наглядно иллюстрируют необходимость тщательного подбора коэффициента k . Рассмотрим сходимость SGD на данной функции с различными k .



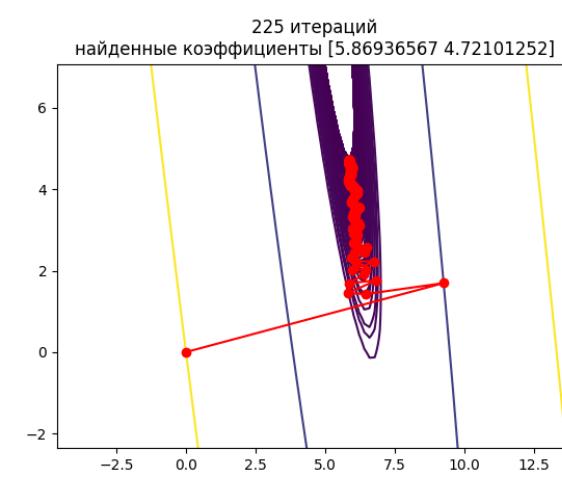
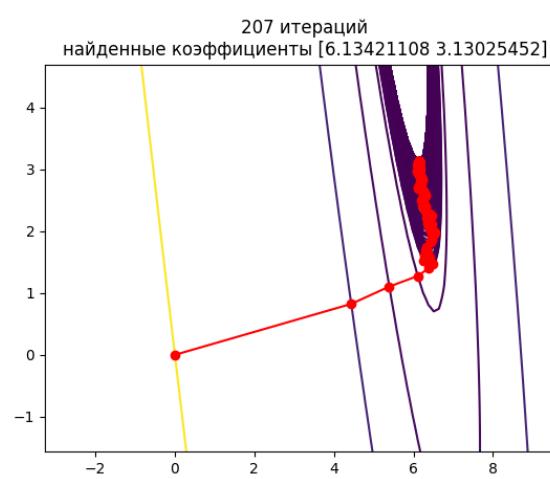
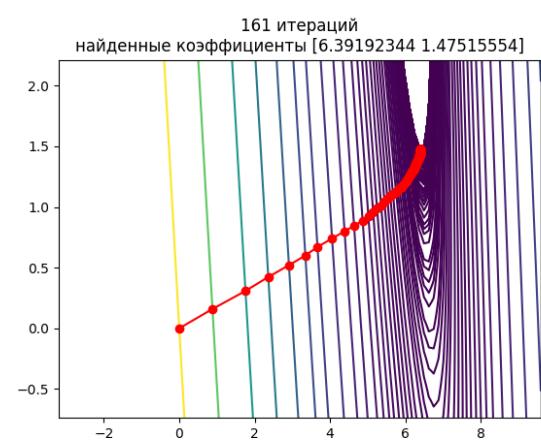
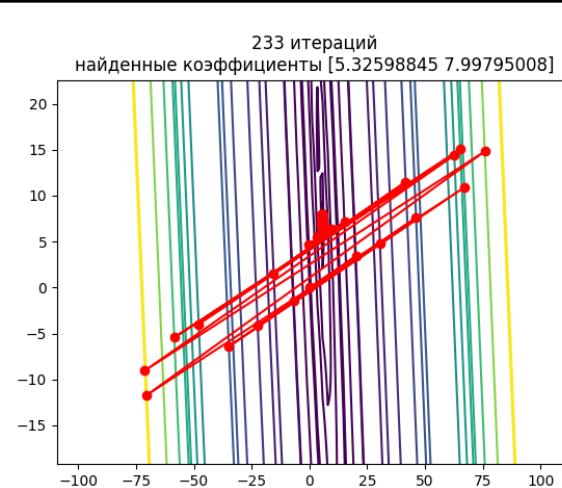
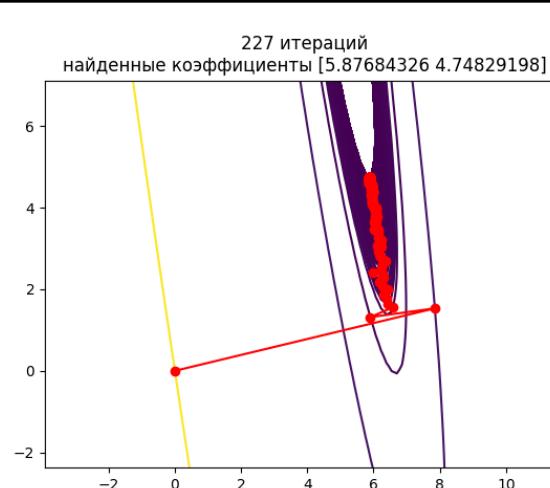
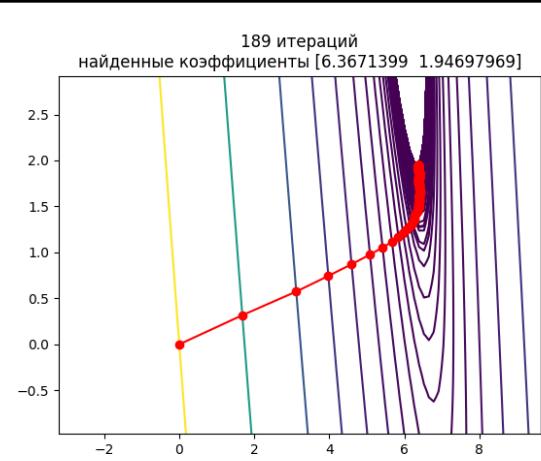


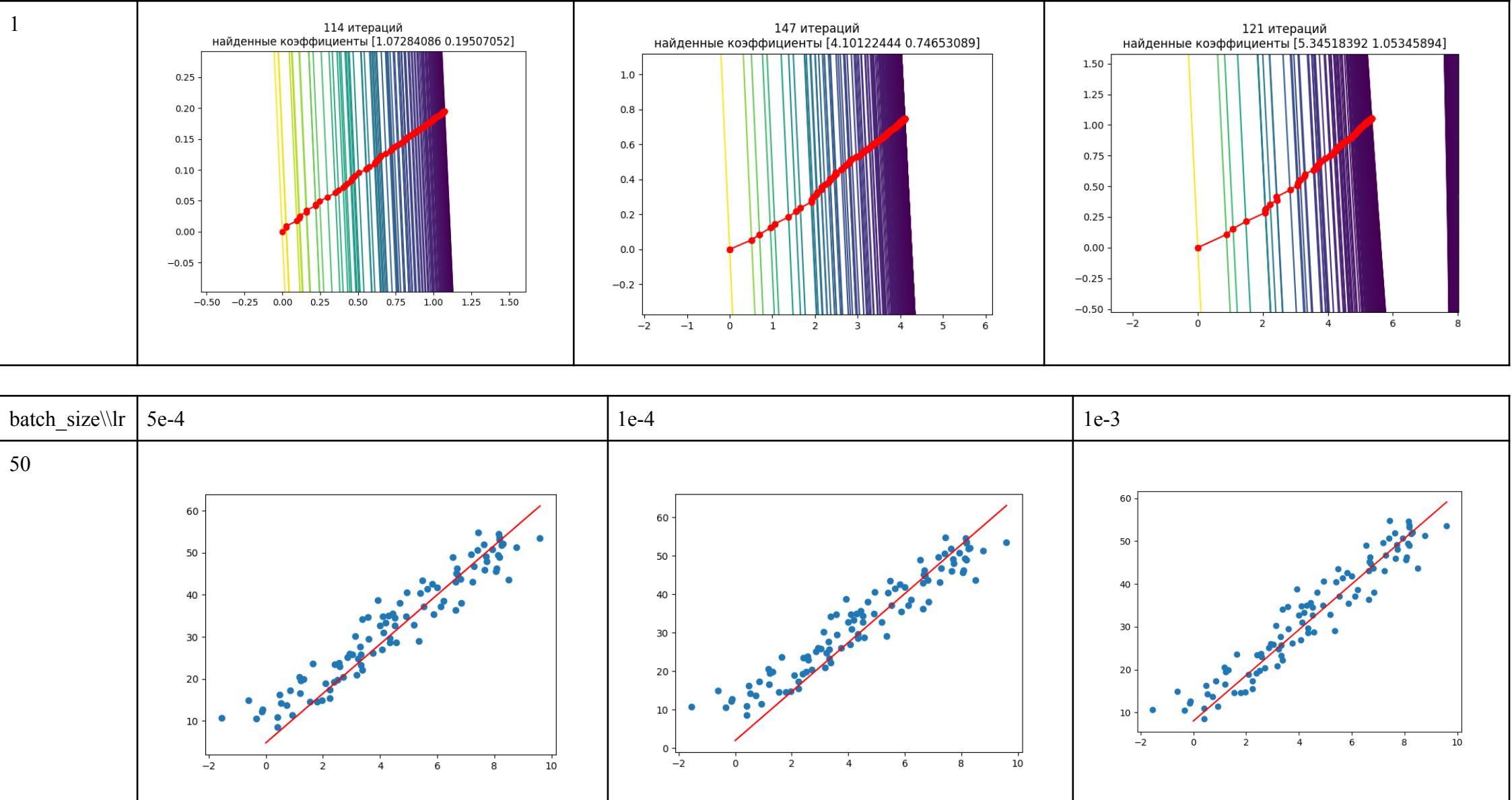


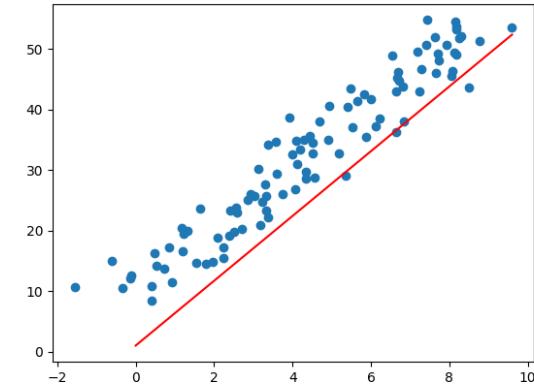
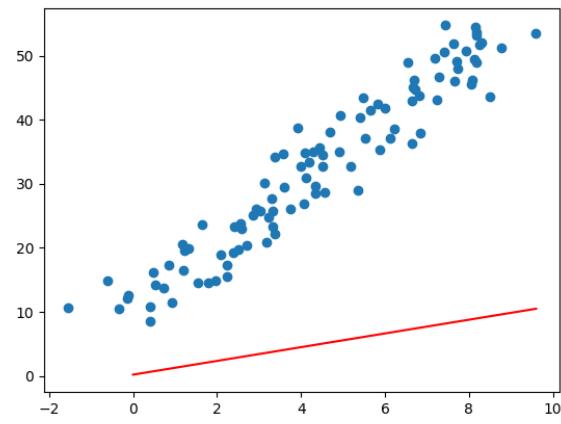
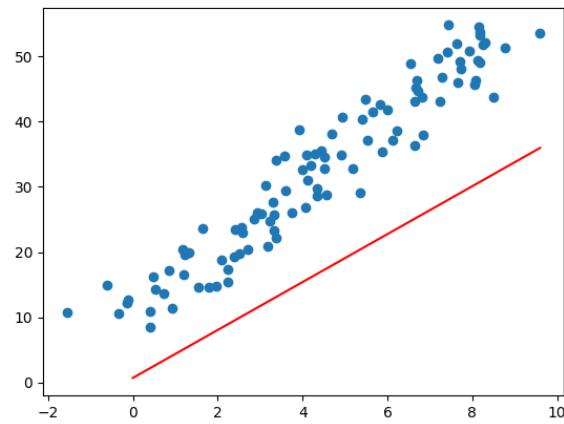
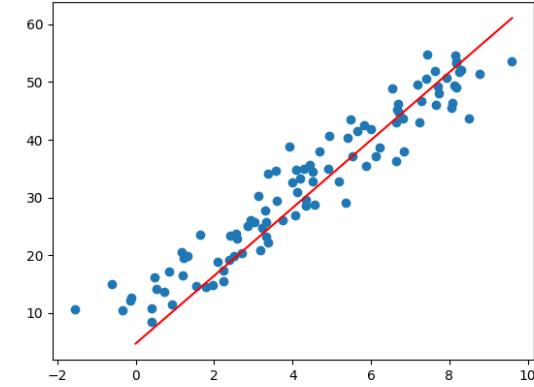
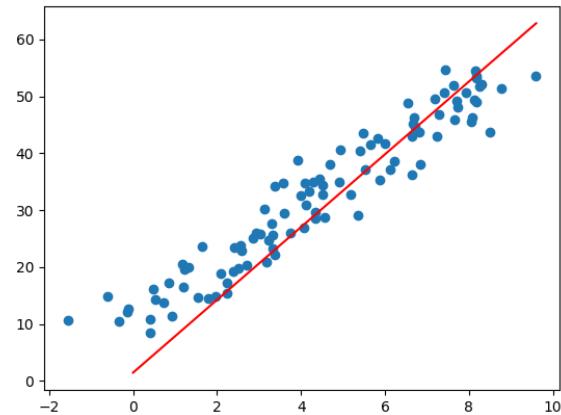
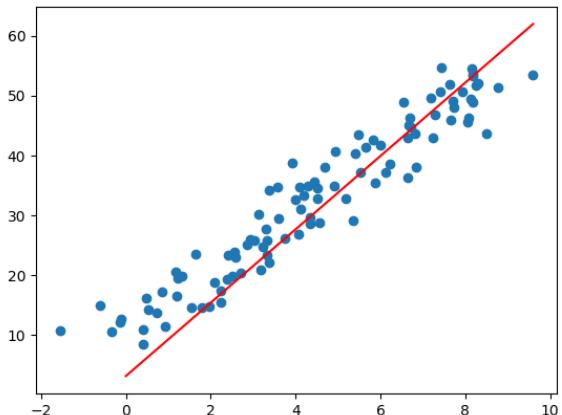
Наше предположение подтвердилось. Проведенная серия экспериментов показывает, что при правильно подобранном k даже неоптимальный batch_size начинает сходиться самостоятельно и лучше.

Кроме того, проведём серию экспериментов с критерием остановки для градиентного спуска.

batch_size\\lr	1e-4	5e-4	1e-3
----------------	------	------	------







Таким образом, заметив, что градиентный спуск на меньших размерах батча начинает ходить в некоторой окрестности минимума. Мы допустили уменьшение максимального количества итераций на несколько порядков и получить достаточно неплохой результат.

Задача 3.

Градиентный спуск находит минимум функции, однако у него есть некоторые недостатки, такие как:

- Зависимость от выбора скорости обучения (шага) и начального приближения
- Медленная сходимость к оптимальному решению
- Чувствительность к зашумленным или негладким данным
- Склонность к попаданию в локальные минимумы

Для устранения этих недостатков были предложены различные модификации градиентного спуска, которые учитывают дополнительную информацию о функции или данных, такую как:

- Импульс (Momentum) – добавляет к градиенту часть предыдущего шага, чтобы ускорить движение вдоль основного направления и сгладить колебания. Сдвиг параметров расчитывается как взвешенная сумма сдвига на предыдущем шаге и нового на основе градиента:

$$g_{new} = \beta \times g_{prev} + (1 - \beta) \times direction$$

$$x_{new} = x_{prev} - lr \times g_{new}$$

- Nesterov – улучшает импульс, предугадывая следующее положение параметров и вычисляя градиент в этой точке, а не в текущей.

$$g_{new} = \beta \times g_{prev} + lr * grad(x - \beta * g_{prev})$$

$$x_{new} = x_{prev} - g_{new}$$

- AdaGrad – адаптирует скорость обучения для каждого параметра индивидуально в зависимости от того как сильно он менялся ранее. С этой целью будем хранить для каждого параметра сумму квадратов его обновлений. Данный метод борется с осцилляцией, не допуская ситуации, когда один параметр меняется слишком быстро.

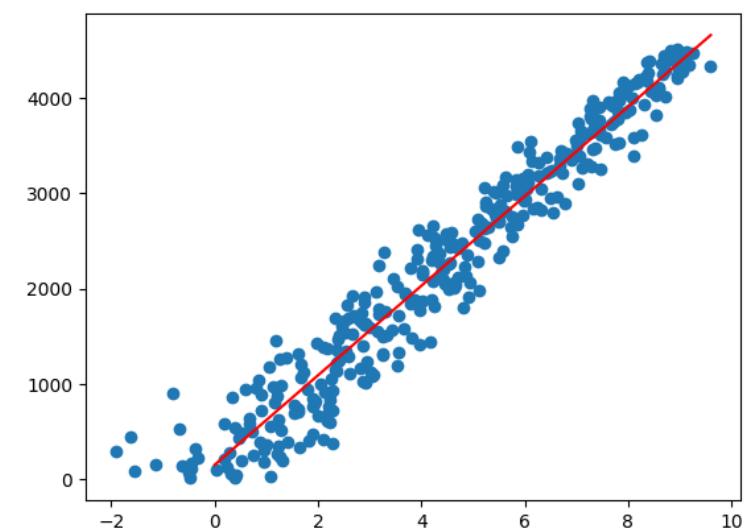
$$G_{new} = G + grad(x)^2$$

$$x_{new} = x_{prev} - grad(x) * lr \div (\sqrt{g_{new}} + \varepsilon)$$

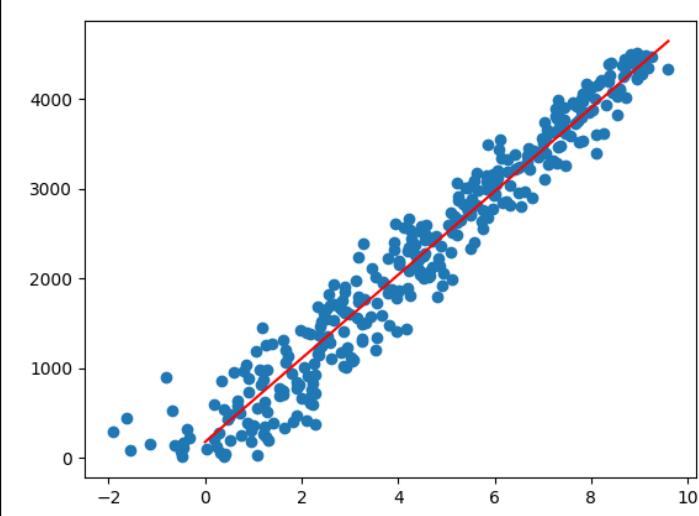
- RMSProp – модифицирует AdaGrad, чтобы избежать быстрого затухания скорости обучения, используя экспоненциальное скользящее среднее квадратов градиентов.
- Adam – комбинирует идеи импульса и RMSProp, сохраняя скользящие средние как градиентов, так и квадратов градиентов.

Эти модификации позволяют более эффективно и стабильно оптимизировать сложные функции.

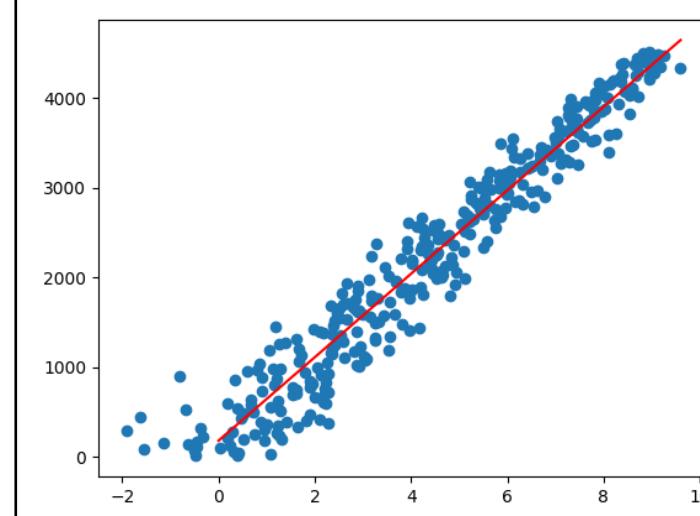
GD



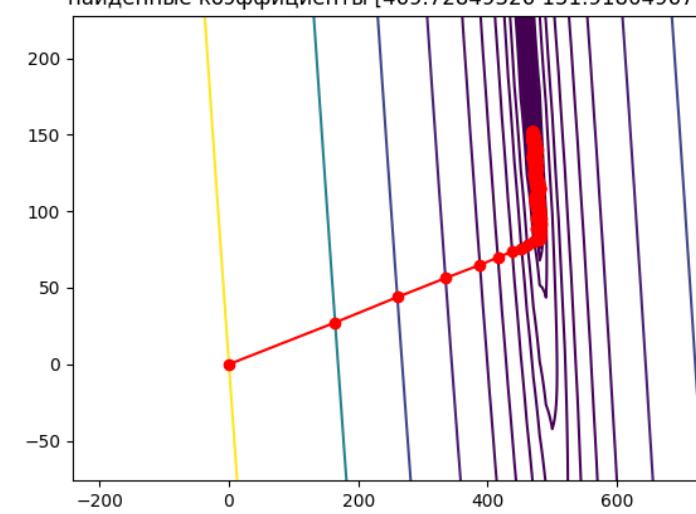
MomentumGD



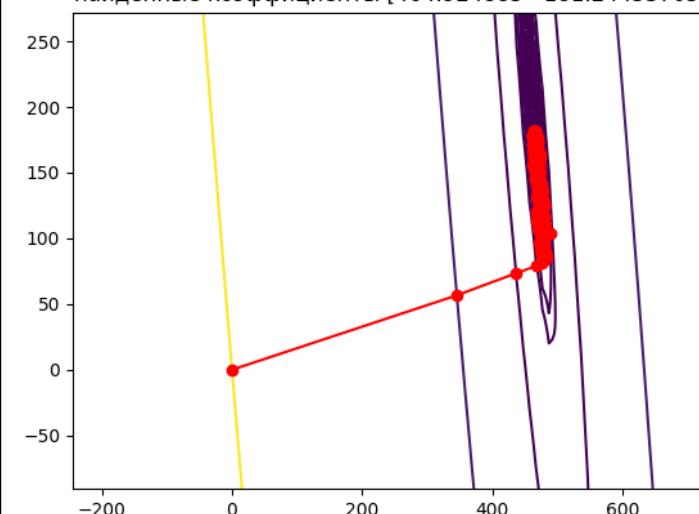
NesterovGD



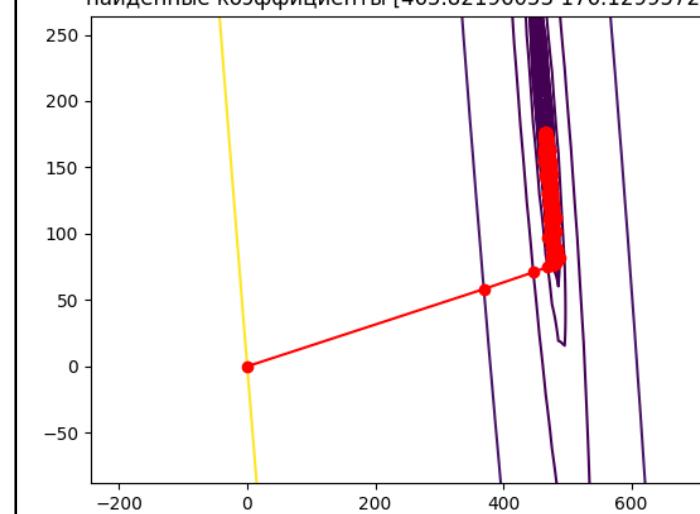
2240 итераций
найденные коэффициенты [469.72849326 151.91804967]



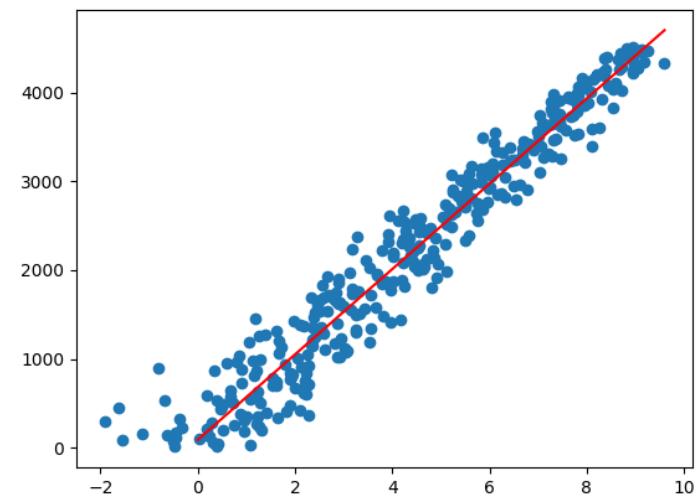
1645 итераций
найденные коэффициенты [464.924885 181.24453705]



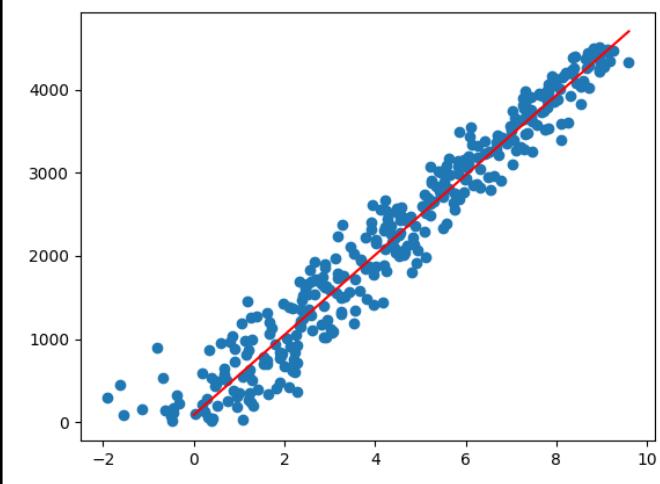
1561 итераций
найденные коэффициенты [465.82196053 176.12995721]



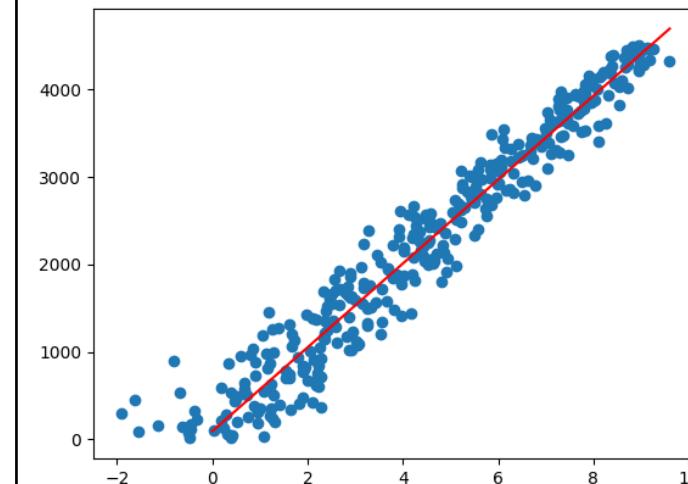
AdaGrad



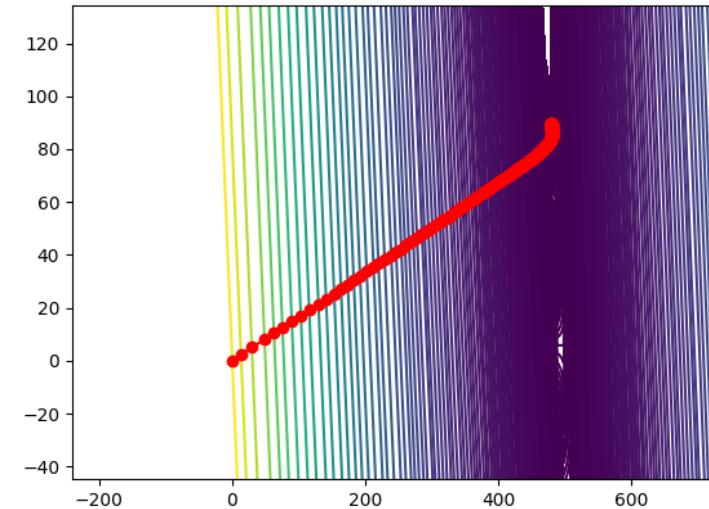
RMSProp



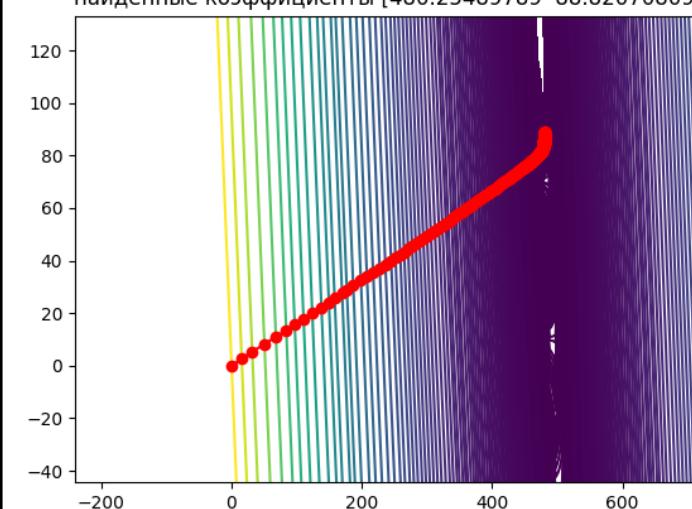
Adam



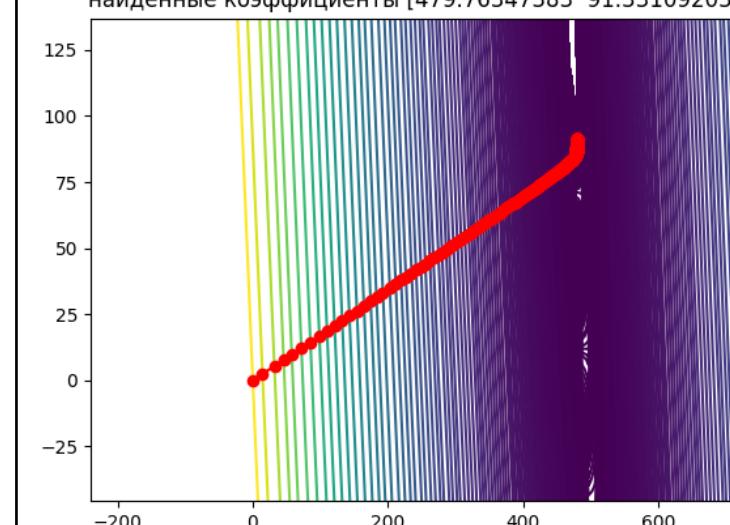
1483 итераций
найденные коэффициенты [480.18153813 89.69175692]



1233 итераций
найденные коэффициенты [480.25489789 88.82070809]



990 итераций
найденные коэффициенты [479.76347383 91.33109203]



Задача 4.

Исследуйте сходимость алгоритмов. Сравните различные методы по скорости сходимости, надежности, требуемым машинным ресурсам (объему оперативной памяти, количеству арифметических операций, времени выполнения).

1. Сходимость модификаций была представлена выше: все модификации в целом не сильно отличаются друг от друга, однако проведя множественный эксперимент и записывая лучшие результаты, удалось выяснить, что лучшим среди них является Adam.
2. Для эксперимента будем подбирать плохие lr, чтобы показать надёжность.. Самой надежной модификацией SGD является AdaGrad, поскольку даже при высоких значениях lr он показал хорошую сходимость. Следующими по этой характеристике будут методы, также учитывающие накопленную сумму, но умеющие ее забывать: это Adam и RMSProp. Все три метода накапливали сумму векторе, то есть учитывали отдельно каждую координату. Последними по надежности будут методы, использующие предугадывание (шаг вперед) и инерцию - это момент Нестерова и метод Momentum.
3. Сравнение требуемого объема оперативной памяти: каждая из рассмотренных модификаций требовала хранение вектора нецелых чисел с размером, равным размерности задачи. В компьютерах для хранения одного числа требуется 64 бита. Ровно так работает AdaGrad. Модификации SGD, такие как Momentum, Nesterov и RMSProp хранят также дополнительный double - коэффициент, который они используют для пересчета нового положения x. Замыкающим станет Adam, который требует $O(2^k)$ нецелых чисел в оперативной памяти, где k - размерность задачи.
4. Количество арифметических операций сильно варьируется: Momentum < Nesterov* (т.к Нестеров вызывает градиент дважды, не очень понятно как его оценивать) < AdaGrad < RMSProp < Adam
5. Время выполнения обратно пропорционально скорости сходимости и прямо пропорционально размеру датасета и сложности алгоритма.

Дополнительное задание

1. Перед нами стоит задача [реализации полиномиальной регрессии](#)

Полиномиальная регрессия - это метод регрессионного анализа, который используется для моделирования отношений между независимой переменной (X) и зависимой переменной (Y), где связь между ними предполагается нелинейной. Вместо прямой линии, которая используется в линейной регрессии, полиномиальная регрессия использует полиномиальную функцию более высокой степени для аппроксимации отношения между X и Y.

Обычно полиномиальная регрессия используется, когда имеется много данных, которые не могут быть линейно смоделированы. Модель полиномиальной регрессии может использовать полиномы второй, третьей, четвертой и т.д. степеней для аппроксимации данных.

Для нахождения оптимальных коэффициентов полиномиальной регрессии используется метод наименьших квадратов, который минимизирует сумму квадратов ошибок между моделью и фактическими данными. Как и в линейной регрессии, после нахождения коэффициентов модели, можно использовать ее для прогнозирования значений зависимой переменной на основе значений независимой переменной.

Формула для полиномиальной регрессии зависит от степени полинома, который используется в модели. Общая формула для полиномиальной регрессии степени n имеет вид:

$$y = \sum_{i=0}^n b_i x^i + \varepsilon$$

где:

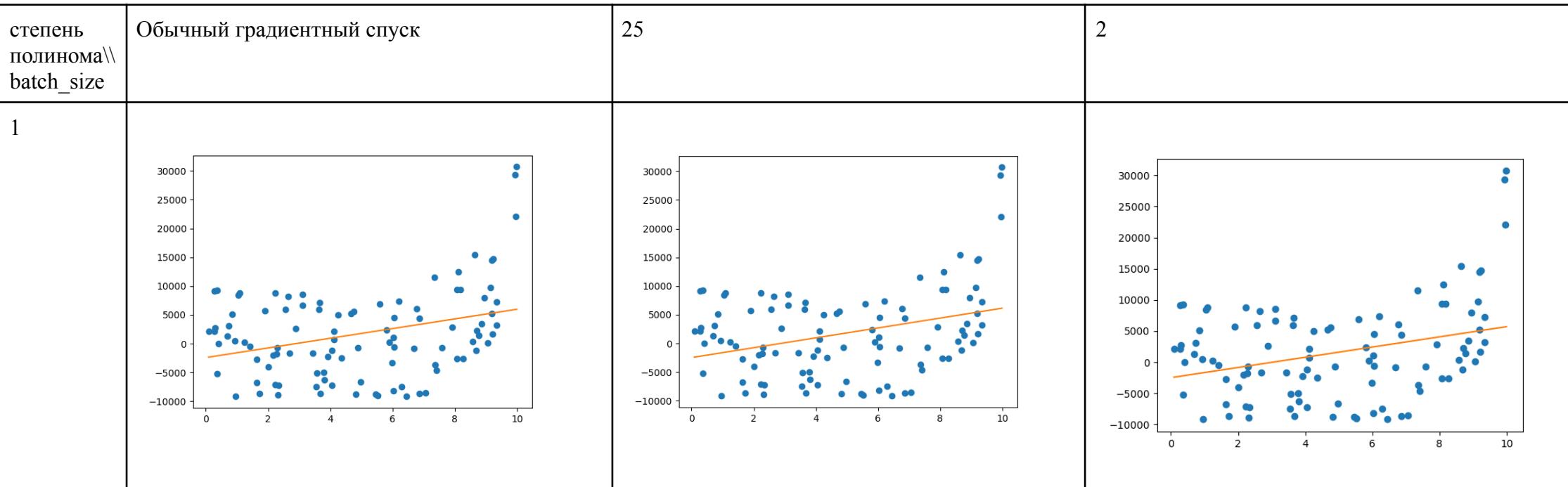
y - зависимая переменная

x - независимая переменная

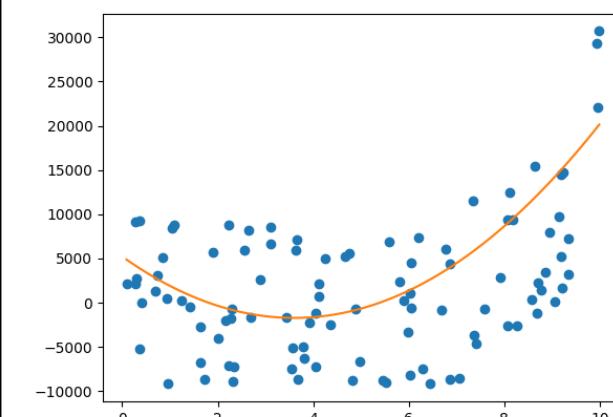
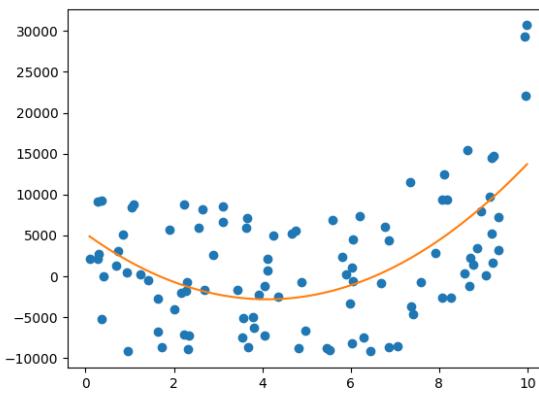
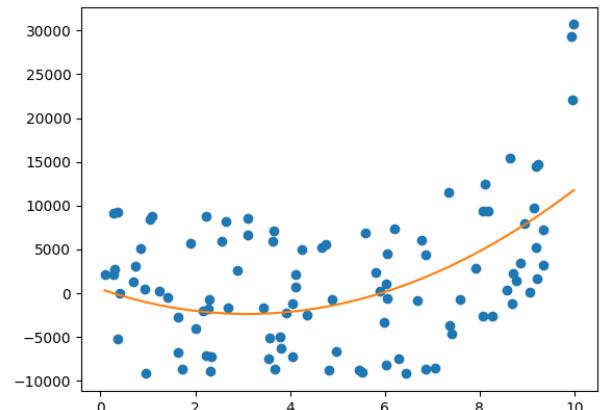
b_i - коэффициенты регрессии, которые определяют форму полинома

ε - случайная ошибка, которая представляет разницу между фактическими значениями y и предсказанными значениями модели

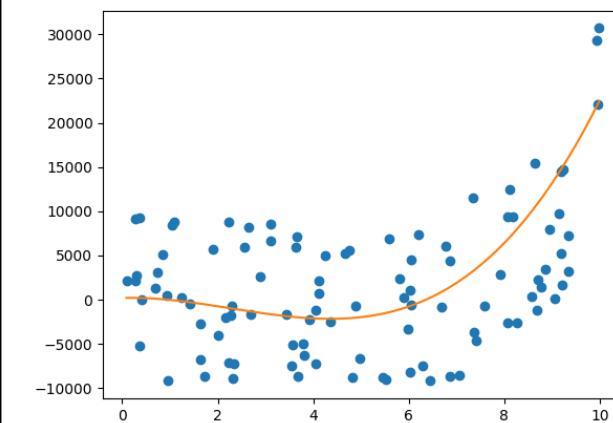
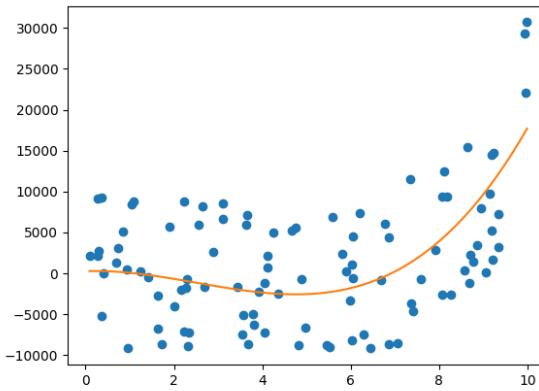
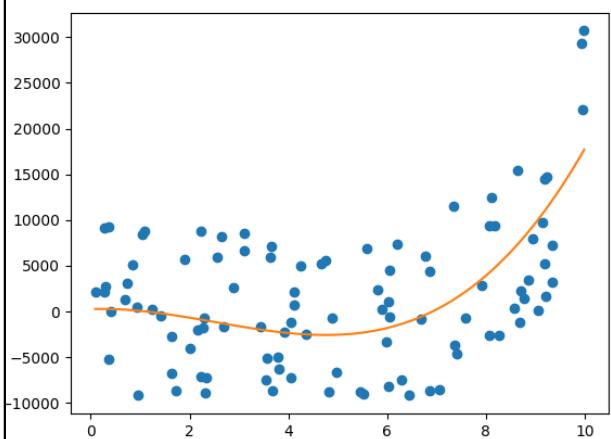
Построим графики восстановленной регрессии для полиномов разной степени.



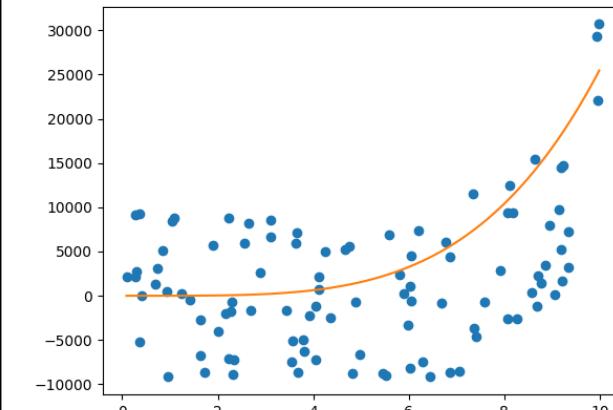
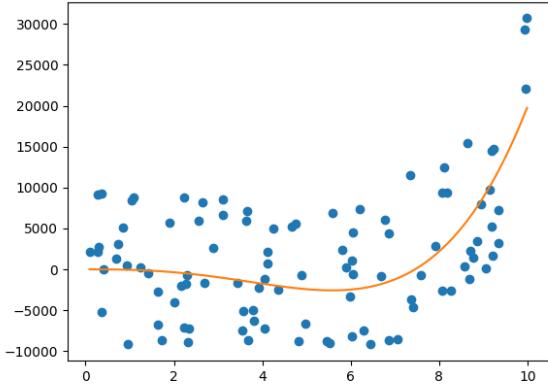
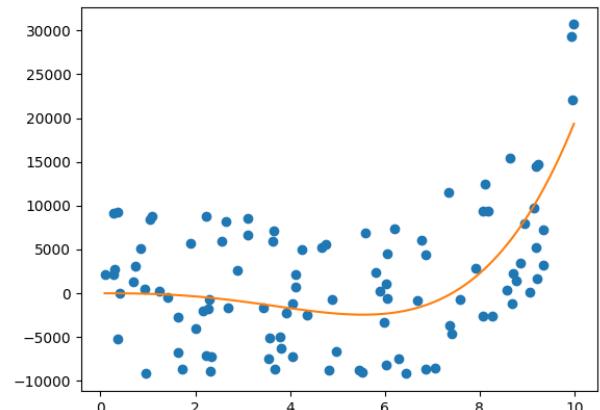
2



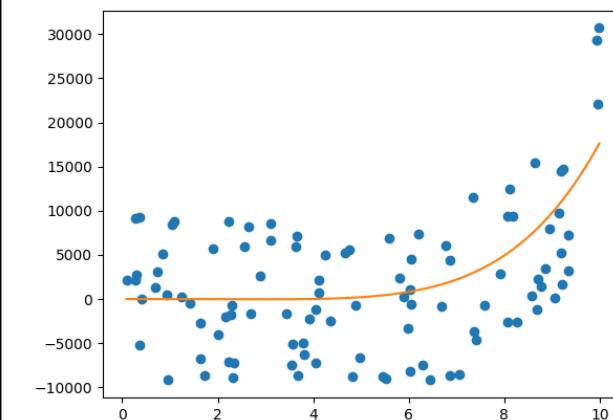
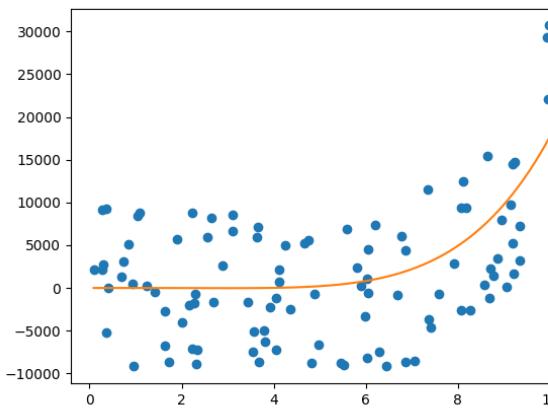
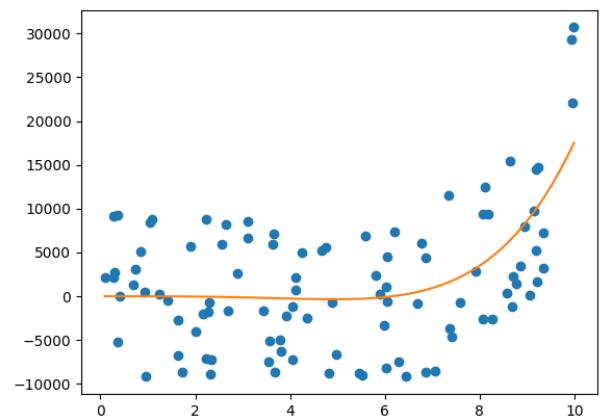
3



4



5



[2. Модифицируйте полиномиальную регрессию добавлением регуляризации в модель \(L1, L2, Elastic регуляризации\)](#)

Регуляризация (англ. regularization) в статистике, машинном обучении, теории обратных задач — метод добавления некоторых дополнительных ограничений к условию с целью решить некорректно поставленную задачу или предотвратить переобучение. Чаще всего эта информация имеет вид штрафа за сложность модели.

Переобучение в большинстве случаев проявляется в том, что итоговые модели имеют слишком большие значения параметров. Соответственно, необходимо добавить в целевую функцию штраф за это. Наиболее часто используемые виды регуляризации — L_1 и L_2 а также их линейная комбинация — эластичная сеть.

В представленных ниже формулах для эмпирического риска

$\mathbf{Q} : \mathcal{L}$ является функцией потерь, а β — вектором параметров из модели алгоритма, а λ — неотрицательный гиперпараметр, являющийся коэффициентом регуляризации.

L_2 -регуляризация

L_2 регуляризация, или регуляризация Тихонова (англ. ridge regularization или Tikhonov regularization):

$$Q(\beta, X^l) = \sum_{i=1}^l \mathcal{L}(y_i, g(x_i, \beta)) + \lambda \sum_{j=1}^n \beta_j^2$$

Минимизация регуляризованного соответствующим образом эмпирического риска приводит к выбору такого вектора параметров β , которое не слишком сильно отклоняется от нуля. В линейных классификаторах это позволяет избежать проблем мультиколлинеарности и переобучения.

L_1 -регуляризация

L_1 регуляризация, или регуляризация Тихонова (англ. ridge regularization или Tikhonov regularization):

$$Q(\beta, X^l) = \sum_{i=1}^l \mathcal{L}(y_i, g(x_i, \beta)) + \lambda \sum_{j=1}^n |\beta_j|$$

Данный вид регуляризации также позволяет ограничить значения вектора β . Однако, к тому же он обладает интересным и полезным на практике свойством — обнуляет значения некоторых параметров, что в случае с линейными моделями приводит к отбору признаков.

Запишем задачу настройки вектора параметров β :

$$Q(\beta) = \sum_{i=1}^l \mathcal{L}_i(\beta) + \lambda \sum_{j=1}^n |\beta_j|,$$

где $\mathcal{L}_i(\beta) = \mathcal{L}(y_i, g(x_i, \beta))$ — некоторая ограниченная гладкая функция потерь.

Сделаем замену переменных, чтобы функционал стал гладким. Каждой переменной β_j поставим в соответствие две новые неотрицательные переменные:

$$\begin{cases} u_j = \frac{1}{2}(|\beta_j| + \beta_j) \\ v_j = \frac{1}{2}(|\beta_j| - \beta_j) \end{cases}$$

Тогда:

$$\begin{cases} \beta_j = u_j - v_j \\ |\beta_j| = u_j + v_j \end{cases}$$

В новых переменных функционал становится гладким, но добавляются ограничения-неравенства:

$$\begin{cases} Q(u, v) = \sum_{i=1}^l \mathcal{L}_i(u - v) + \lambda \sum_{j=1}^n (u_j + v_j) \rightarrow \min_{u, v} \\ u_j \geq 0, v_j \geq 0, j = 1, \dots, n \end{cases}$$

Для любого j хотя бы одно из ограничений

$u_j \geq 0$ и $v_j \geq 0$ обращается в равенство, иначе второе слагаемое в $Q(u, v)$ можно было бы уменьшить, не изменив первое. Если гиперпараметр λ устремить к ∞ , в какой-то момент все $2n$ ограничений обратятся в равенство. Постепенное увеличение гиперпараметра λ приводит к увеличению числа таких j , для которых $u_j = v_j = 0$ откуда следует, что $\beta_j = 0$. Как говорилось ранее, в линейных моделях это означает, что значения j -го признака игнорируются, и его можно исключить из модели.

Эластичная сеть

Эластичная сеть (англ. elastic net regularization):

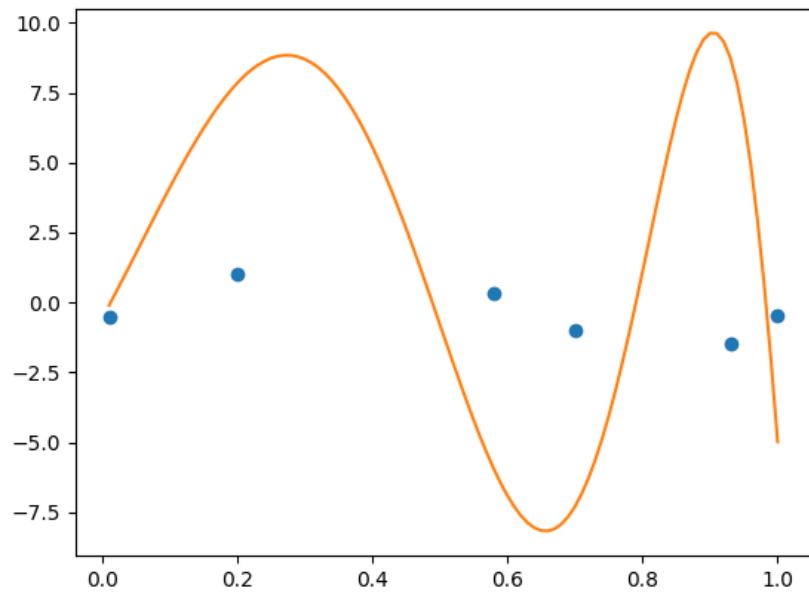
$$Q(\beta, X^l) = \sum_{i=1}^l \mathcal{L}(y_i, g(x_i, \beta)) + \lambda_1 \sum_{j=1}^n |\beta_j| + \lambda_2 \sum_{j=1}^n \beta_j^2$$

Приведенная регуляризация использует как

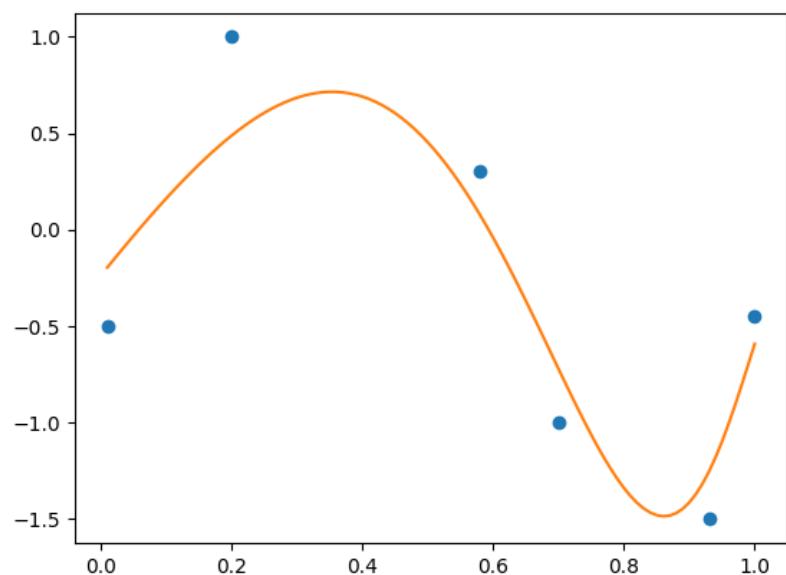
L_1 , так и L_2 регуляризации, учитывая эффективность обоих методов. Ее полезной особенностью является то, что она создает условия для группового эффекта при высокой корреляции переменных, а не обнуляет некоторые из них, как в случае с L_1 -регуляризацией.

Исследуем влияние регуляризации на восстановление регрессии на примере полинома 16 степени:

Без регуляризации

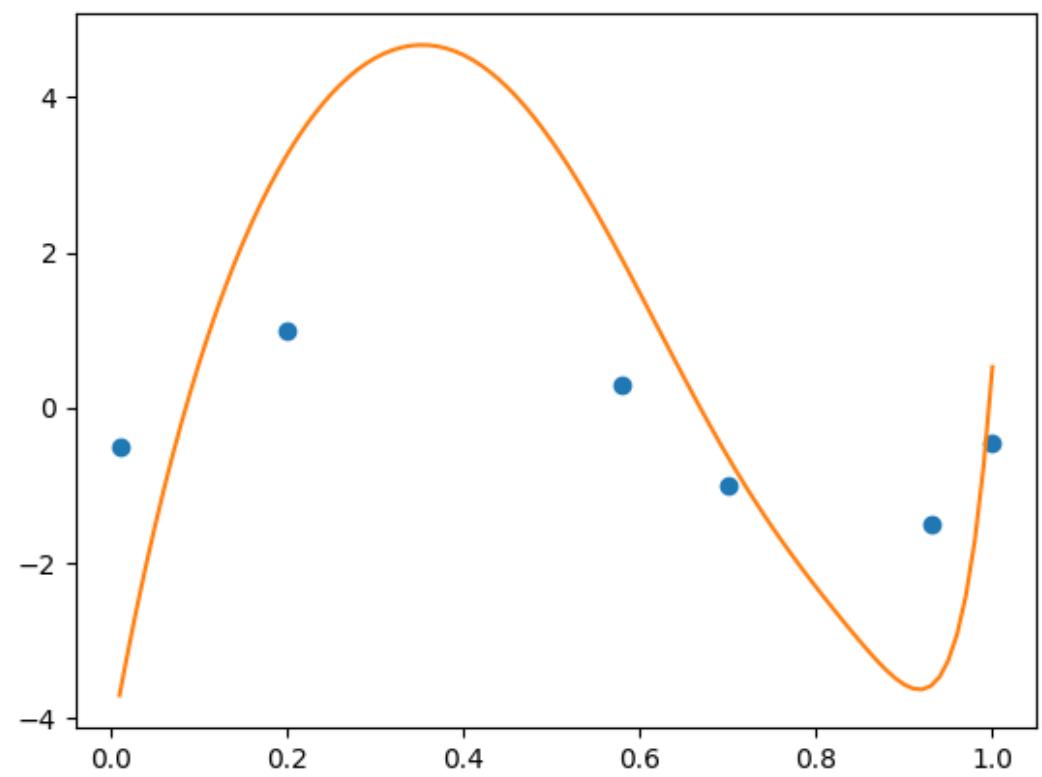


L2 - регуляризация
 $\lambda_2 = 0.199$



L1 - регуляризация

$$\lambda_1 = 0.125$$

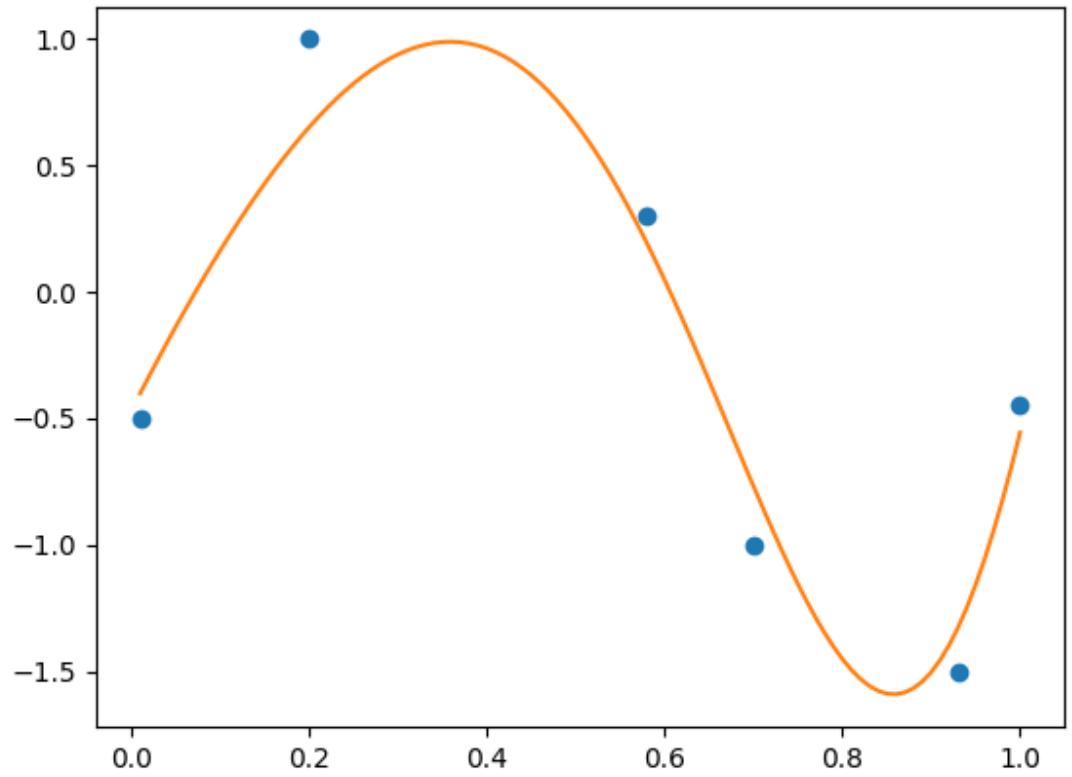


Elastic

регуляризация

$$\lambda_1 = 0.0002$$

$$\lambda_2 = 0.199$$



Изменение значения λ позволяет настраивать уровень регуляризации, который применяется к модели. При более высоких значениях λ веса модели будут более строго ограничены, что приведет к более простым моделям с меньшей вариантностью. С другой стороны, при более низких значениях λ регуляризационный член будет иметь меньшее влияние на модель, что приведет к более сложным моделям с более высокой вариантностью.

Итак, L2-регуляризация позволяет контролировать уровень вариантности модели путем уменьшения весов модели и упрощения модели, что может привести к лучшей способности модели обобщать на новых данных. Однако слишком высокий уровень регуляризации может привести к слишком простой модели, которая не может корректно обобщаться на новые данные.

В целом, применение L2-регуляризации может помочь уменьшить переобучение модели и улучшить ее способность к обобщению. Однако необходимо выбирать правильное значение коэффициента регуляризации, чтобы достичь наилучшей производительности модели.

L1 и L2-регуляризации ограничивают размер коэффициентов модели и, следовательно, помогают уменьшить переобучение. Однако L1-регуляризация имеет свойство устранять некоторые признаки, устанавливая соответствующие коэффициенты в ноль. Это может быть полезно в случае, когда у нас есть множество признаков, некоторые из которых не являются информативными для предсказания целевой переменной. С другой стороны, L2-регуляризация уменьшает все коэффициенты модели, но не обнуляет их. L1-регуляризация может быть полезна для отбора признаков и уменьшения размерности модели, но может ухудшить ее предсказательную способность.

Elastic регуляризация позволяет настраивать баланс между сокращением весов и отбором признаков. Как и L2 регуляризация, она может помочь уменьшить переобучение модели и улучшить ее способность к обобщению. Однако в отличие от L2 регуляризации, elastic регуляризация может также выполнять отбор признаков, устанавливая некоторые веса признаков точно равными нулю.

Исследования показывают, что elastic регуляризация может приводить к более точным предсказаниям, чем L1 или L2 регуляризация по отдельности, особенно когда в данных присутствуют множественные коррелирующие признаки. Также elastic регуляризация может быть полезна в задачах, где необходимо выполнить отбор признаков и уменьшить размерность данных.

В целом, elastic регуляризация является эффективным методом регуляризации, который может помочь улучшить производительность модели, особенно в случаях, когда данные содержат множественные коррелирующие признаки и/или необходимо выполнить отбор признаков. Однако как и в случае с L2 регуляризацией, необходимо подобрать правильное значение коэффициента регуляризации, чтобы достичь наилучшей производительности модели.