**Методы численной оптимизации**

Градиентный спуск — простой метод оптимизации, позволяющий найти минимум целевой функции. Это жадный алгоритм, который делает шаг в сторону максимальной скорости убывания функции. Рассмотрим функцию нескольких переменных f(w), где . Чтобы найти w, при котором функция достигает своего локального минимума, градиентный спуск делает следующие шаги:

1. Выбирается случайное значение w.

2. Выбирается максимальное количество итераций T.

3. Выбирается значение скорости обучения (шаг сходимости)

4. Повторять следующие 2 шага пока значение f не станет приемлемым, или число итераций не превысит T.

a. Вычислить:

b. Перезаписать w:

Здесь

Рассмотрим пример с функцией параболоида в качестве функции нескольких переменных. На каждой итерации обновляются так:

Градиент можно рассматривать как вектор, указывающий направление, в котором функция возрастает быстрее всего. Следование отрицательному направлению градиента приведёт к точкам, в которых значение функции уменьшается с максимальной скоростью. Скорость обучения, также называемая размером шага, определяет, насколько быстро мы движемся в направлении от градиента.

При использовании градиентного спуска мы сталкиваемся со следующими проблемами:

1. Попадание в ловушку локального минимума, что является прямым следствием жадности этого алгоритма.
2. Превышение и отсутствие глобального оптимума — это прямой результат слишком быстрого движения по направлению градиента.
3. Осцилляция - это явление, которое возникает, когда значение функции не изменяется существенно независимо от направления, в котором оно движется (так называемое плато).

Чтобы решить эти проблемы, к выражению добавляется импульс α. Ниже формула для определённой итерации:

**Реализация Градиентного спуска на python**

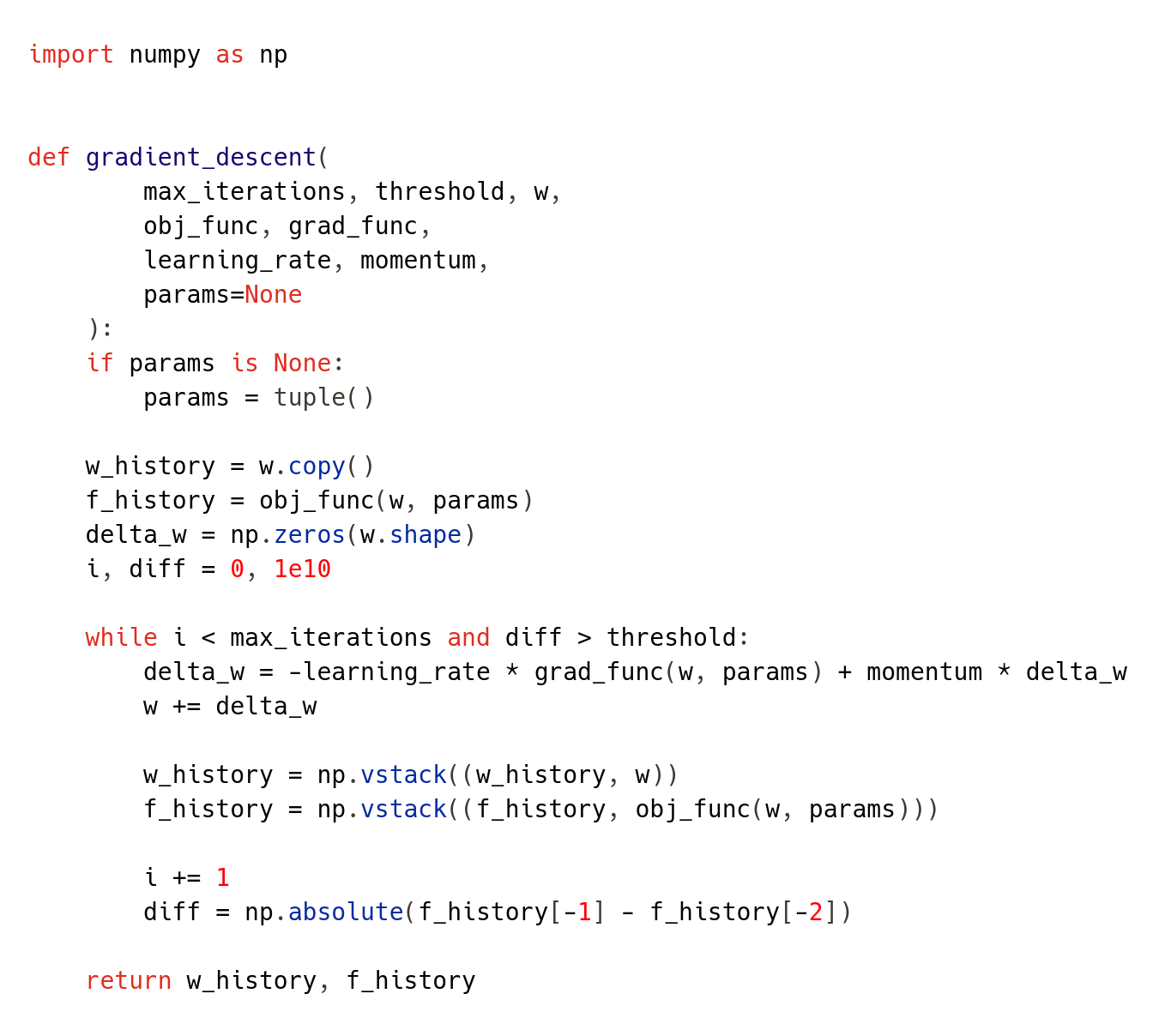
Для работы с матрицами используется библиотека numpy.

Для графического представления используется библиотека matplotlib.

Реализуем функцию градиентного спуска, которая будет принимать функцию, её градиент и возвращать посещённые точки со значениями в этих точках. (см. рис. 1)

Функция будет принимать следующие параметры:

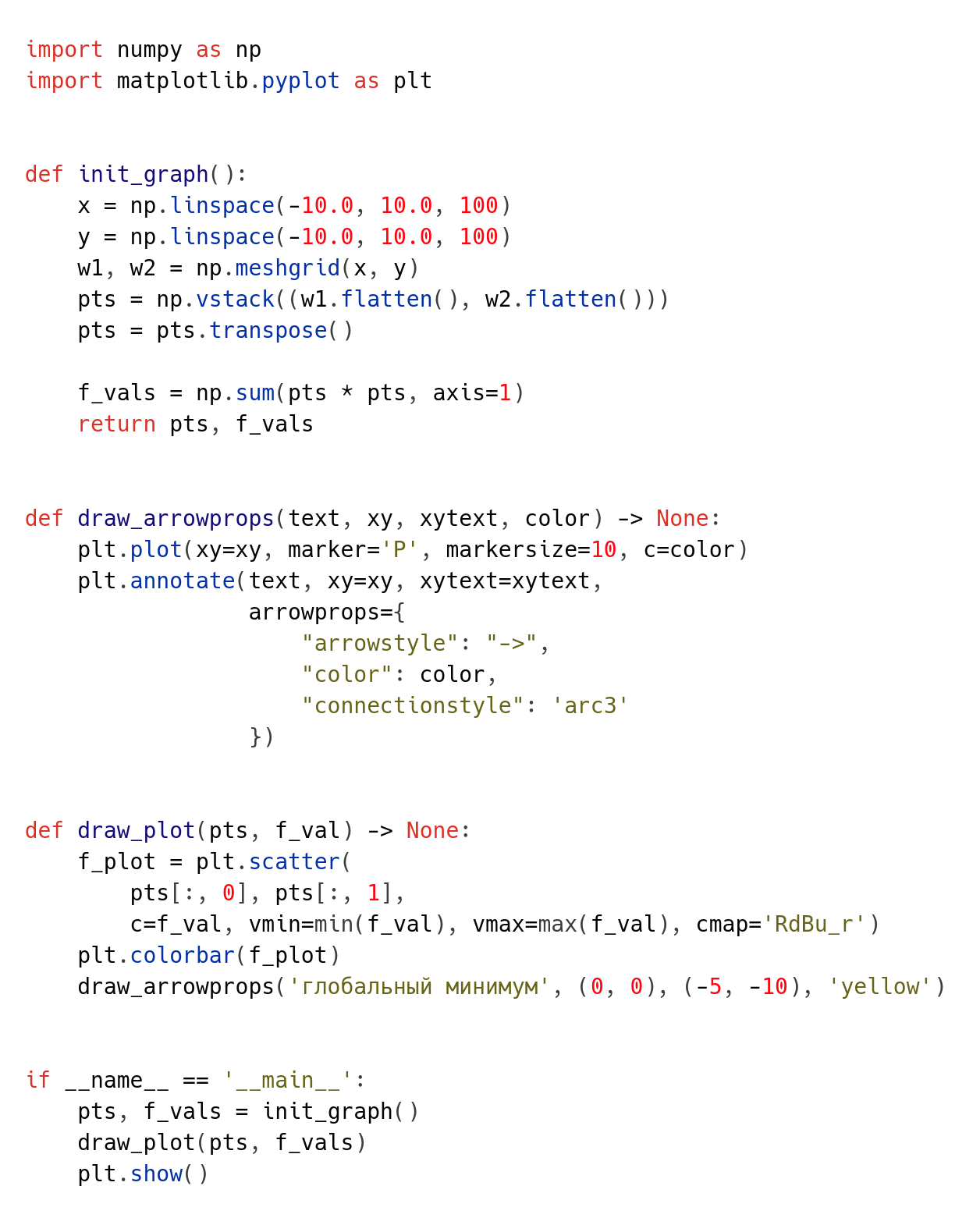
* max\_iterations — максимальное количество итераций.
* threshold — если разница между значениями функции падает ниже этого значения остонавливается вычисление.
* w - точка, с которой начинается вычисление градиентного спуска.
* obj\_func — функция, которая вычисляет значение.
* grad\_func — функция, которая вычисляет градиент.
* learning\_rate — значение скорости обучения (шаг сходимости).
* momentum — значение импульса.
* params — дополнительные параметры для функций вычисляющих градиент и значению (не является необходимым).

Рисунок 1. Функция градиентного спуска

Функция вернёт:

* w\_history — все точки, которые были посещены во время градиентного спуска
* f\_history — значения для всех точек, которые были посещены во время градиентного спуска

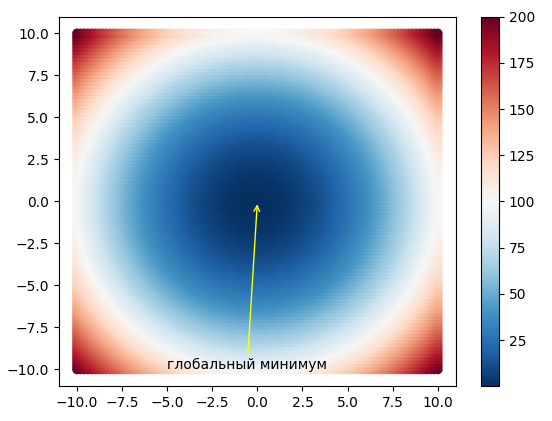
Рассмотрим пример с параболоидом, для начала изобразим график, с которым будем в дальнейшем работать (см. рис.2).

Рисунок 2. Отображение графика параболоида

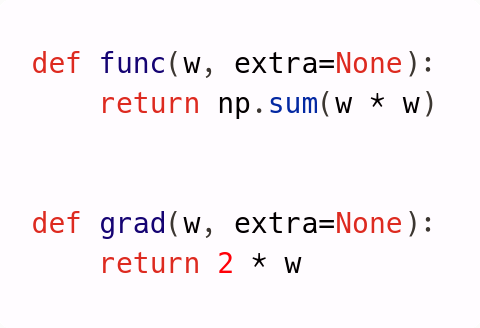
Функция init\_graph генерирует поверхность с 10.000 точками, распределёнными по поверхности.

Функция draw\_arrowprops отображает окрашенную стрелку.

Функция draw\_plot отображает график параболоида, где цвет — значение функции в определённой точке.

Рисунок 3. График параболоида

Далее определим нашу функцию параболоида, а так же её градиент. (см. рис. 4)

Рисунок 4. Функция параболоида и её градиент

Прежде, чем перейти к визуализации зависимости найденного локального минимума от значений изначальных параметров градиентного спуска рассмотрим ещё один метод поиска локального минимума — метод Ньютона.

**метода ньютона**

Метод Ньютона ищет такой *x,* что *f(x)=0* это не совсем то, что нужно, ведь функция не обязательно имеет экстремум в нуле. Поэтому применяется метод Ньютона для оптимизации [1]. Это два разных метода, для последнего необходимо считать вторые производные (матрицу Гессе). Далее под методом Ньютона будет подразумеваться именно метод Ньютона для оптимизации.

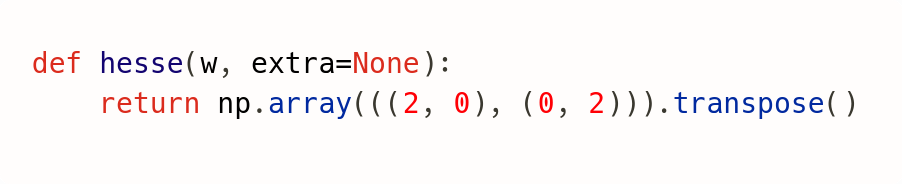
Метод Ньютона итеративный алгоритм, в одномерном случае выглядит следующим образом:

В многомерном случае первая производная заменяется на градиент, а вторая — на матрицу Гессе. Делить матрицы нельзя, поэтому происходит умножение на обратную матрицу.

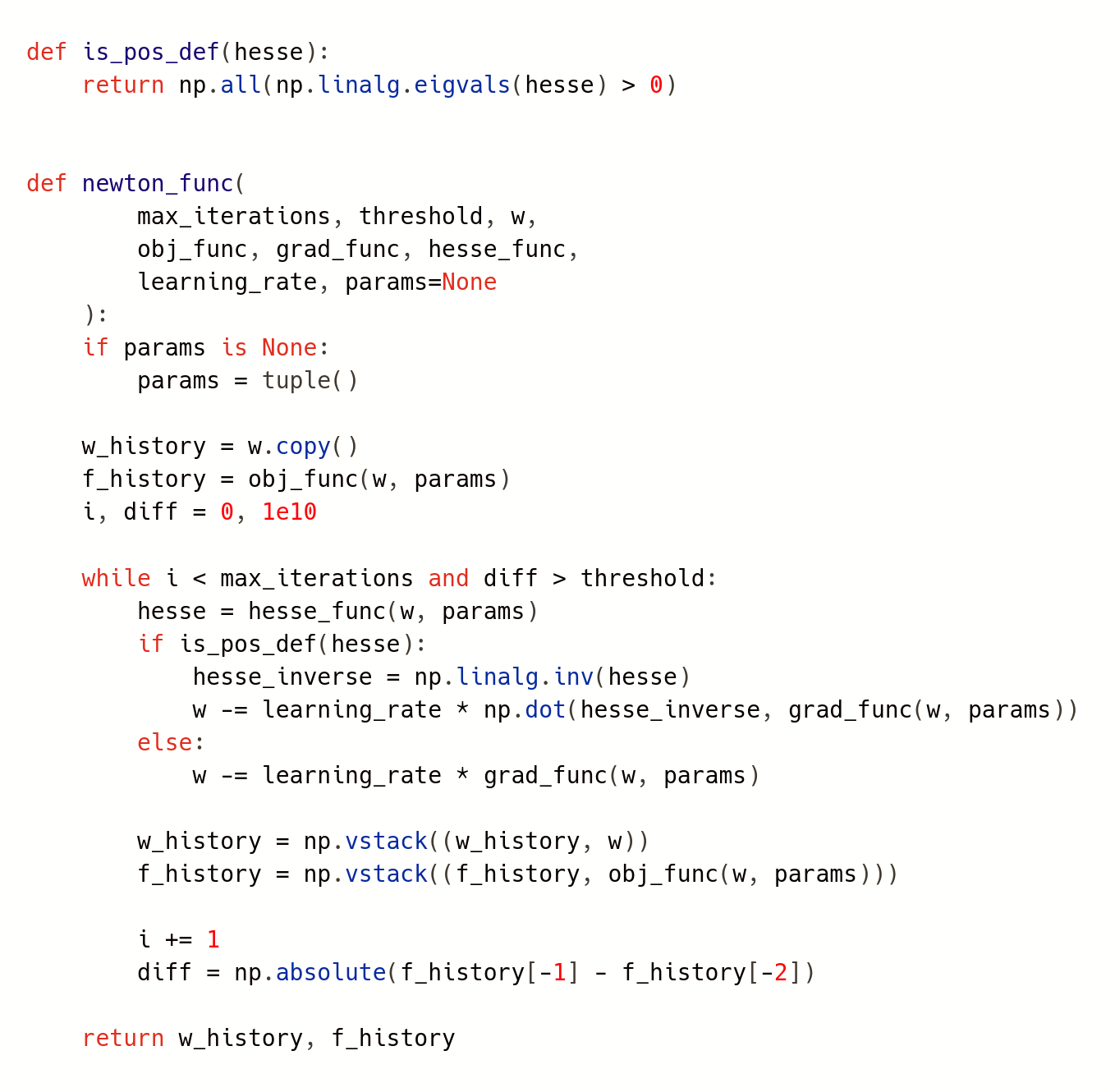
Так же нам необходима проверка в том ли мы направлении движемся. Поэтому будем проверять матрицу Гессе, если она положительно определена [2] — то мы движемся в правильном направлении, если нет — то будем использовать градиент для продвижения далее.

**Реализация Метода ньютона на python**

Для ранее рассмотренной функции параболоида напишем функцию матрицы Гессе. (см рис. 5)

Рисунок 5. Матрица Гессе для функции параболоида

После напишем сам метода Ньютона. (см. рис. 6)

 Рисунок 6. Метод Ньютона

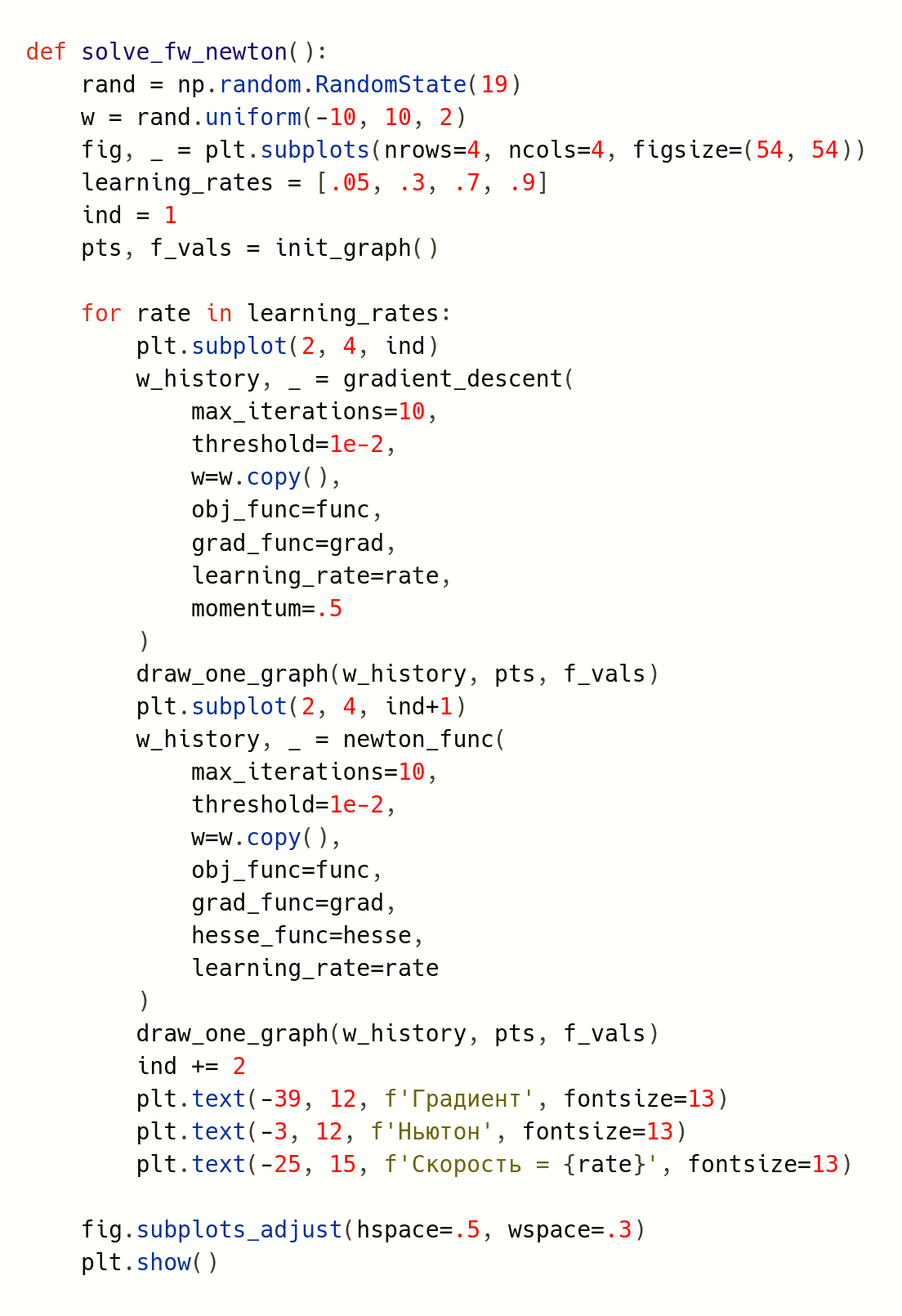
* is\_pos\_def – функция проверки определённости матрицы Гессе.
* newton\_func отличается от функции градиентного спуска тем, что принимается дополнительный параметр:

hesse\_func – функция матрицы Гессе

Возвращаемые значения такие же, как и у градиентного спуска.

**Визуализация методов**

Чтобы сравнить два этих метода поиска локального минимума напишем функцию визуализации. (см. рис. 7)

Рисунок 7. Визуализация методов

Для визуализации точки глобального минимума, а так же траектории движения на каждом графике напишем функцию отрисовки одного графика. (см. рис. 8)

Рисунок 8. Визуализация одного графика

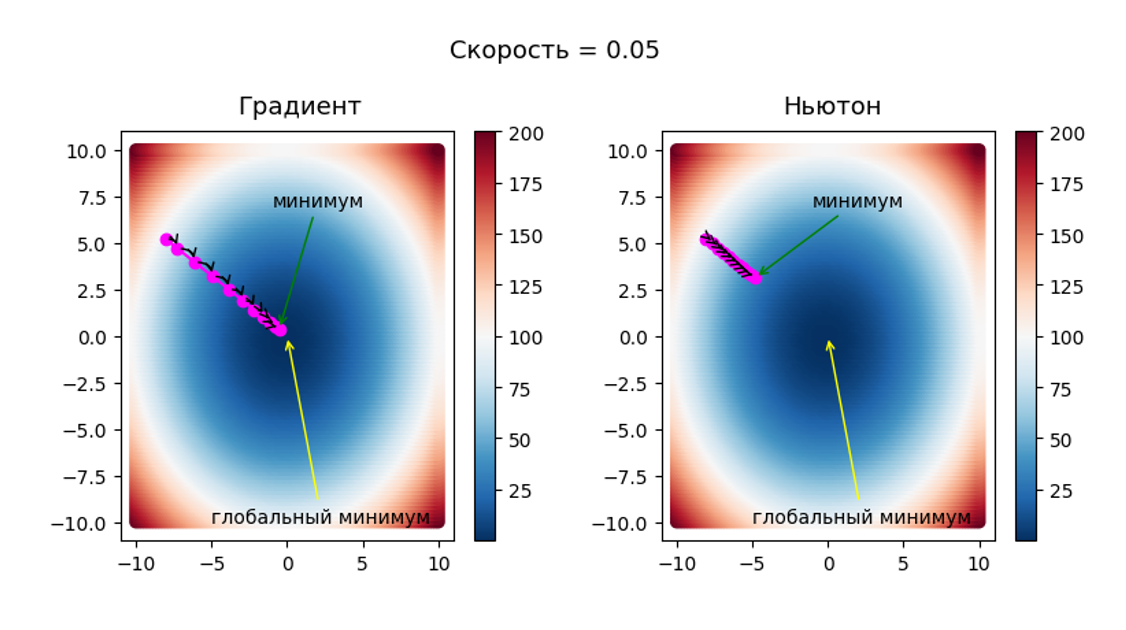
Рассмотрим влияние параметров методов на результат. Импульс для градиентного спуска выберем 0.5, скорость обучения будет меняться [0.05, 0.3, 0.7, 0.9]. (см. рис. 7)

Рисунок 9. Сравнение методов на примере параболоида.

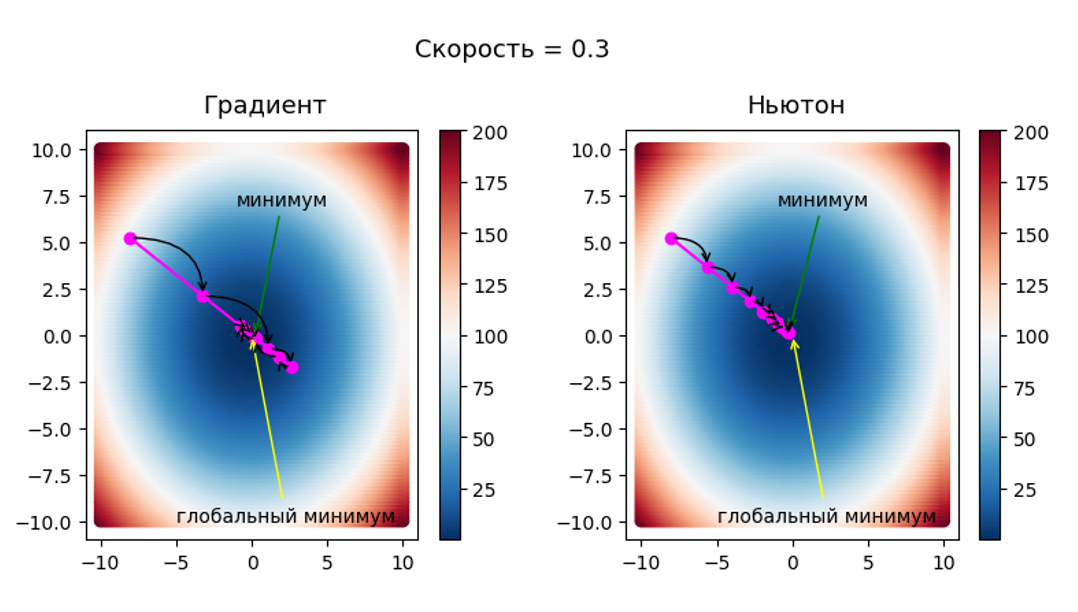


Рисунок 10. Сравнение методов на примере параболоида.

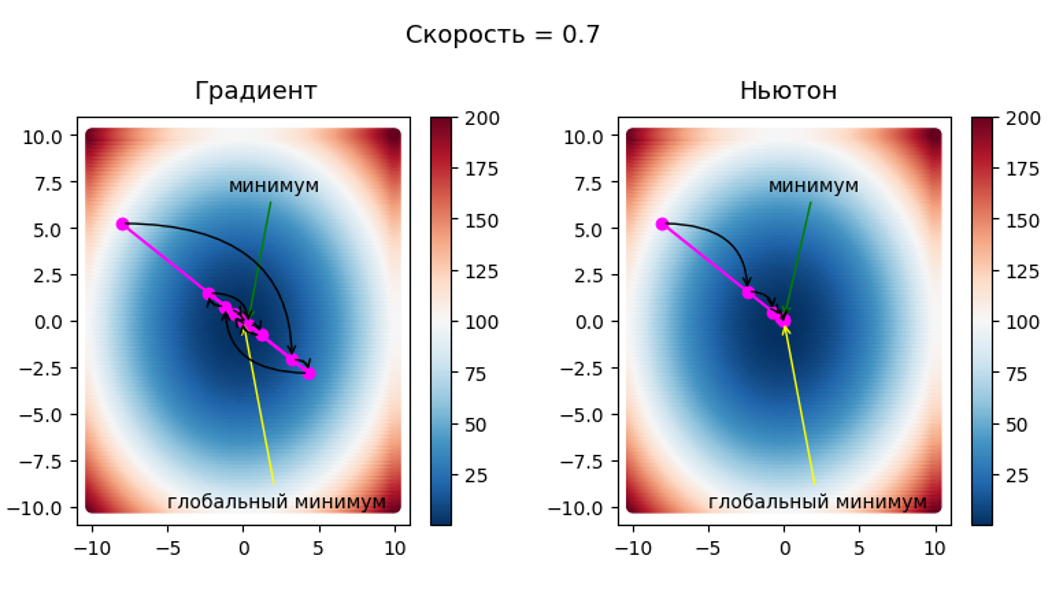
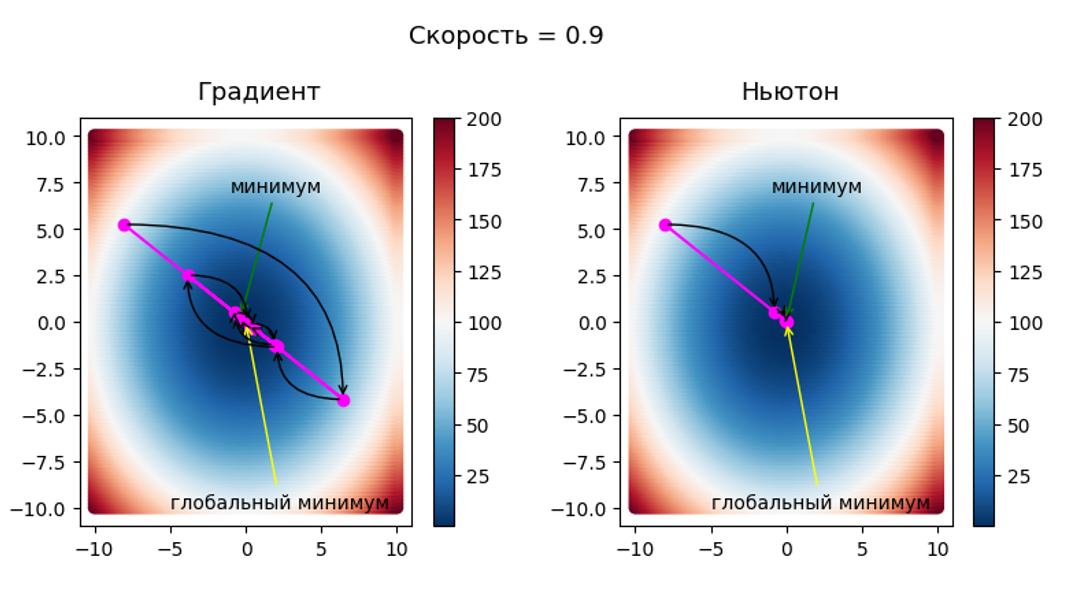


Рисунок 11. Сравнение методов на примере параболоида.

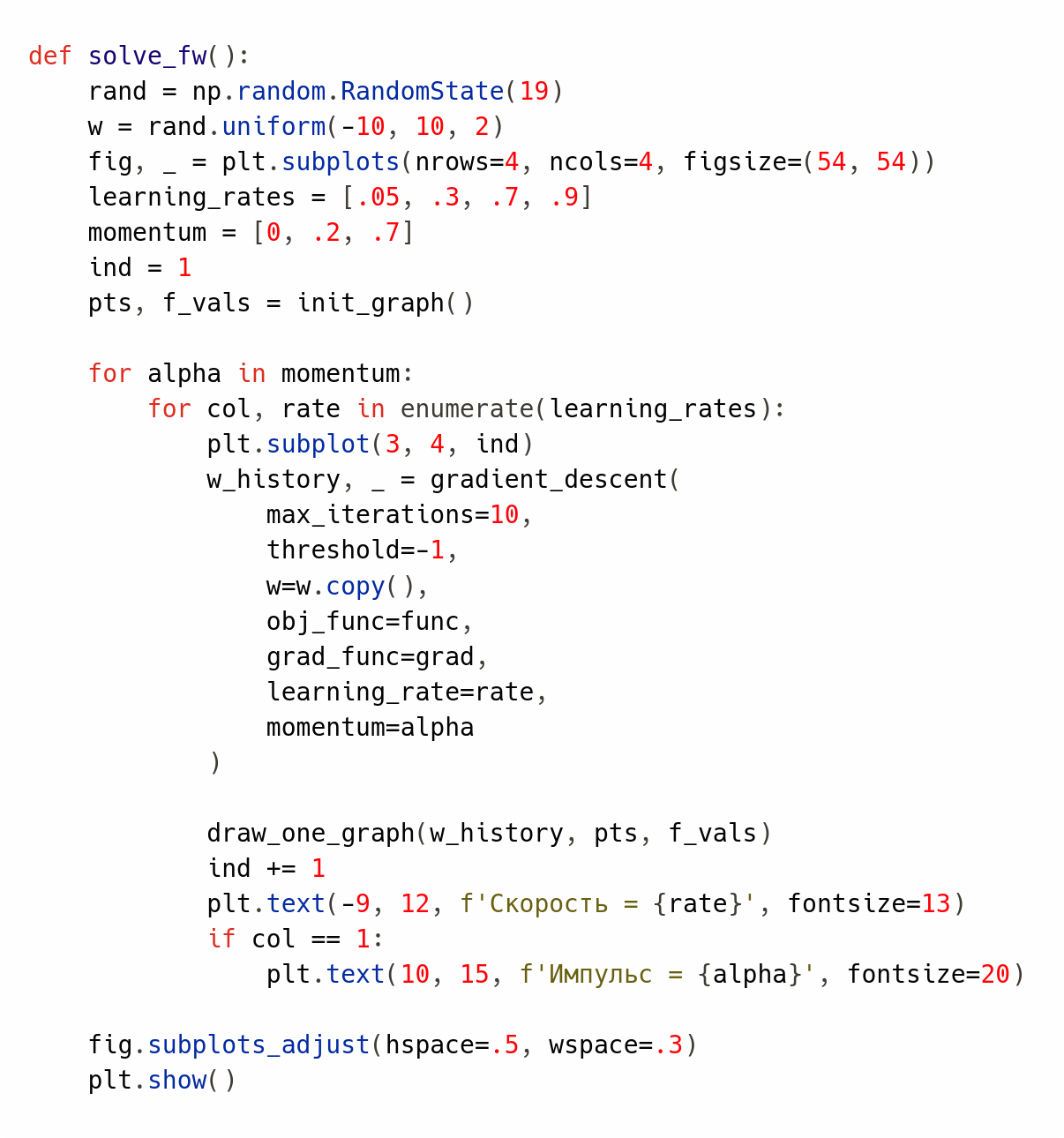
Рисунок 12. Сравнение методов на примере параболоида.

На рисунке 9 видно, что при маленьком значении скорости обучения ни один из методов не достиг глобального минимума, однако градиентный спуск оказался ближе, это связано с тем, что помимо скорости обучения он имеет импульс, который смещает точку к искомому значению.

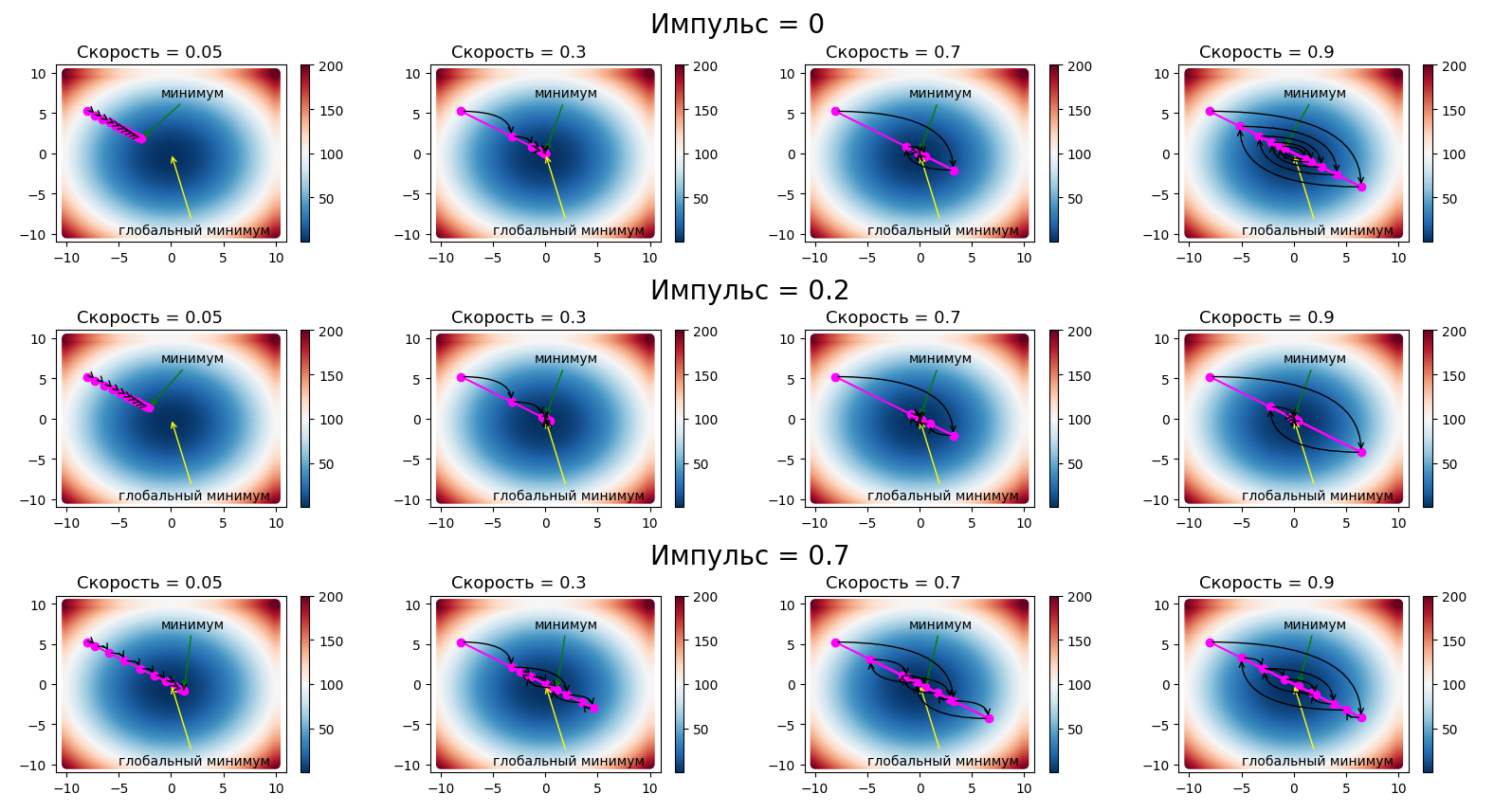
На рисунке 10 видно, что уже начиная с небольшого значения скорости обучения в 0.3 градиентный спуск начинает «перепрыгивать» через искомое значение, а метод Ньютона движется исключительно к искомому значению.

На рисунках 11 и 12 видно, что метод Ньютона работает тем лучше, чем больше значение скорости обучения. Градиентный спуск при больших значениях скорости обучения начинает часто перескакивать через искомую точку.

Рассмотрим как импульс и скорость обучения влияют на найденный локальный минимум. Для этого напишем функцию, которая будет визуализировать только градиентный спуск (см. рис. 13). Рассмотрим те же значения скорости, что и выше, а так же несколько значений импульсов: [0, 0.2, 0.7]

Рисунок 13. Визуализация градиентного спуска

Запустим данный код и увидим следующее (см. рис. 14):

Рисунок 14. Визуализация градиентного спуска

Рассмотрим график с импульсом = 0 и скоростью обучения = 0.05, видно, что алгоритм не достиг глобального минимума, увеличивая импульс (первый столбец) видно, что ситуация улучшается.

Так же рассмотрим график с импульсом = 0.7, и скоростью обучения = 0.9, видно, что алгоритм совершает колебания, это не является оптимальным вариантом.

Оптимально же использовать небольшое значение скорости и среднее значение импульса.

**Градиентный спуск для минимизации среднеквадратичной ошибки**

Градиентный спуск — это хороший и простой метод минимизации среднеквадратичной ошибки в контролируемой классификации или задаче регрессии.

Предположим, что нам дано m обучающих примеров , где каждый пример имеет n признаков. Если соответствующие целевое и выходное значения для каждого примера равны и соответственно, то функция среднеквадратичной ошибки определяется как:

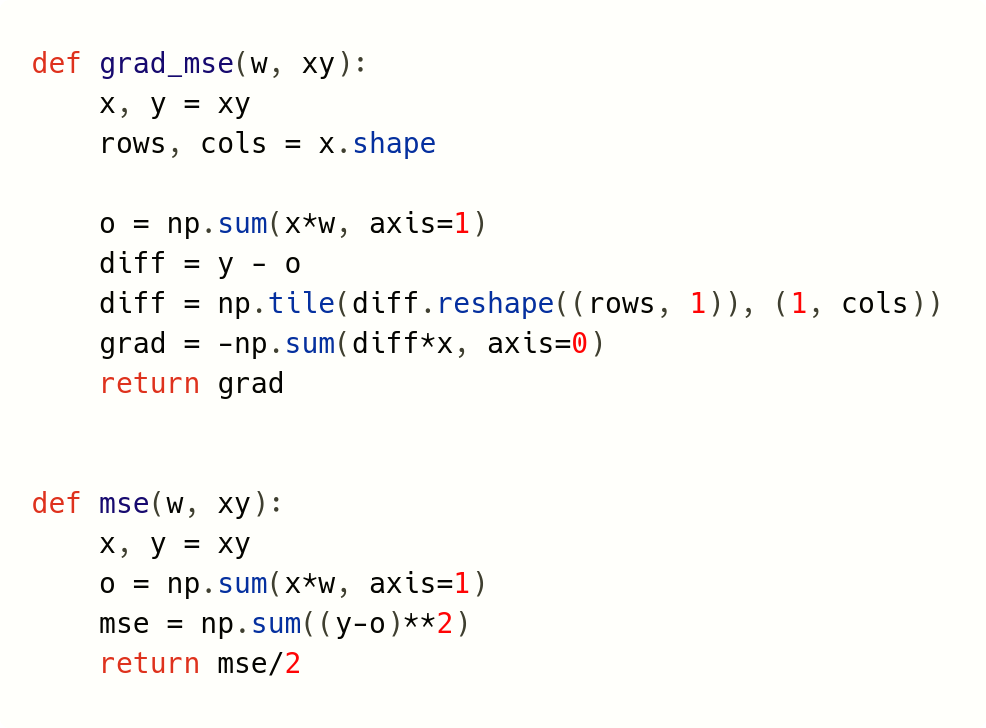
Где выход определяется взвешенной линейной комбинацией входов, определяемой как:

Неизвестным параметром в приведенном выше уравнении является весовой вектор

Целевая функция в этом случае представляет собой среднеквадратичную ошибку с градиентом, определяемым выражением:

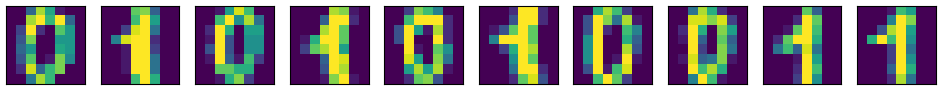
Где – i-й пример, или массив признаков размера n .

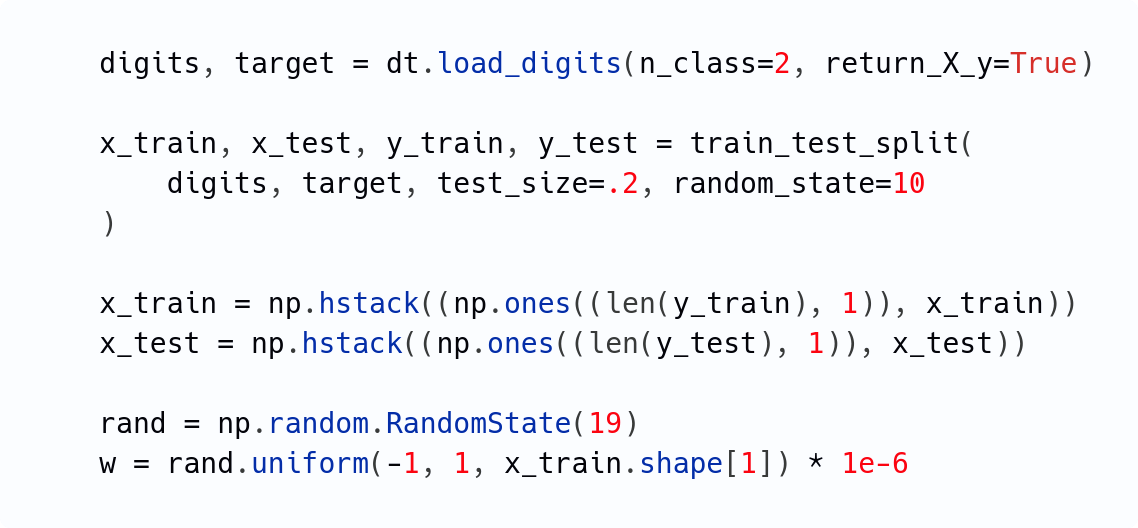
Определим функцию среднеквадратичной ошибки и её градиент. (см. рис. 15)

Рисунок 15. Функция среднеквадратичной ошибки и её градиент

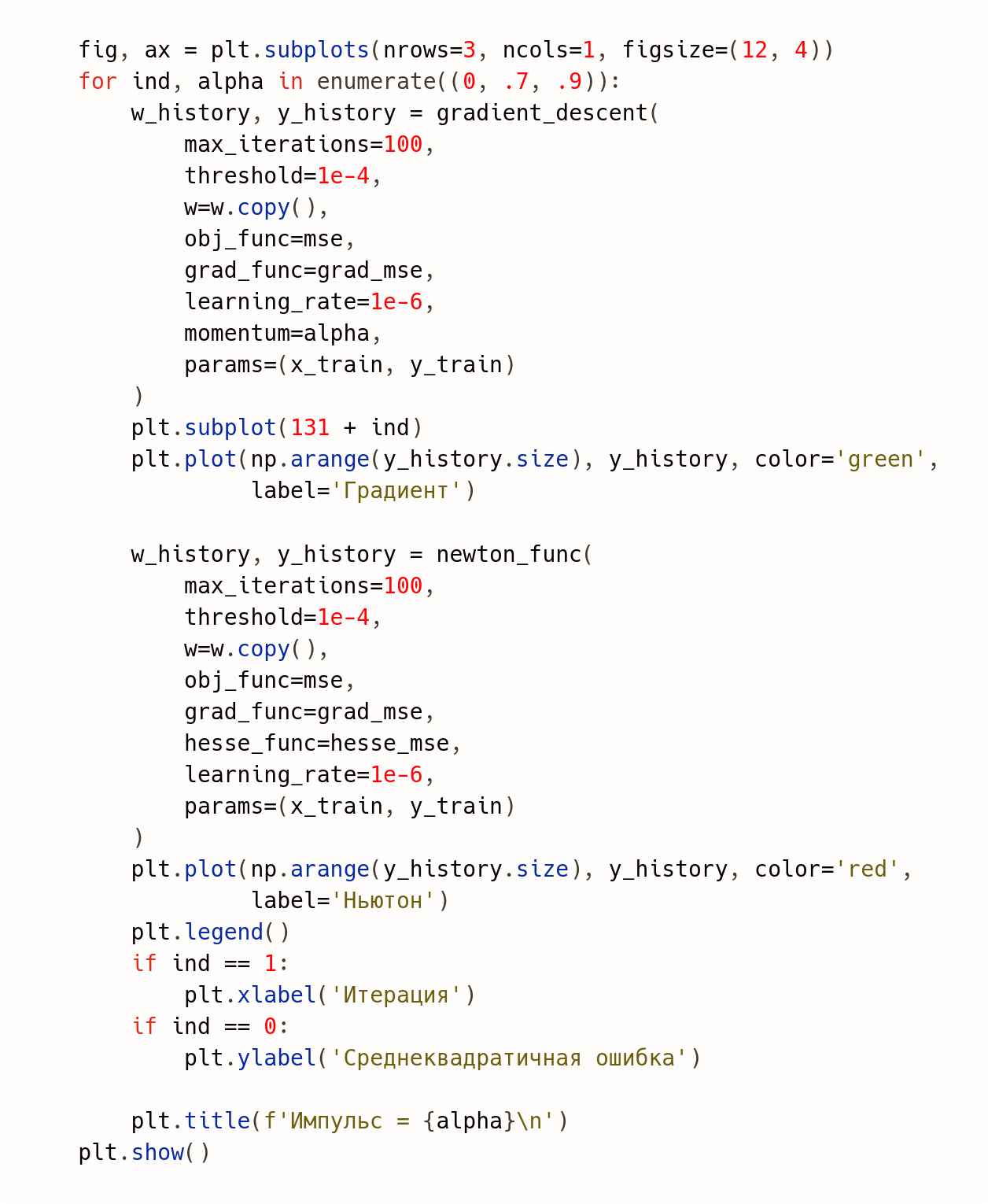
Чтобы проиллюстрировать градиентный спуск в задаче классификации, воспользуемся набором рукописных цифр включённых в sklearn.datasets.

Чтобы не усложнять задачу используем блок с 10 цифрами нулей и единиц. (см. рис. 16)

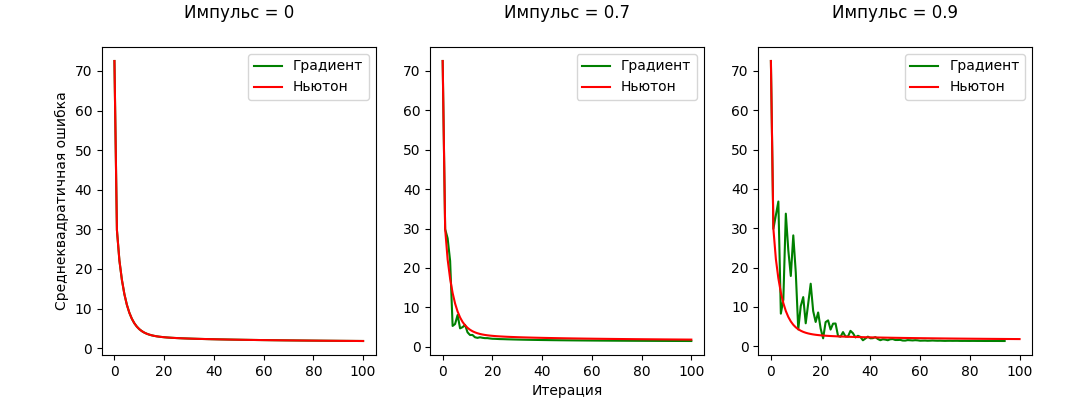
Рисунок 16. Рукописные цифры

Рисунок 17. Данные для классификации

Необходимо разделить данные на обучающие и тестовые. В приведенном ниже коде выполняется градиентный спуск на тренировочном наборе и строится среднеквадратичная ошибка с разными параметрами градиентного спуска. Шаг выбран тестовым путём и равен импульс изменяется по следующему набору: [0, 0.7, 0.9]. (см. рис. 18)

Рисунок 18. Визуализация двух методов минимизации среднеквадратичной ошибки

Запустим код и увидим:

Рисунок 19. Среднеквадратичная ошибка двух алгоритмов

В ходе анализа выяснилось, что незначительное изменение параметров градиентного спуска значительно влияет на результат. Нет какого-то универсального значения, и оно зависит от функции. Однако, зачастую маленькое значение шага и среднее значение импульса даёт хороший результат, что и было подтверждено в ходе исследования на функции параболоида, а так же в минимизации среднеквадратичной ошибки в задаче классификации.

**Список литературы:**

1. Аттетков, А. В., Галкин, С. В., & Зарубин, В. С. (2001). Методы оптимизации. Изд. МГТУ им. Баумана.
2. Yu. Nesterov. Efficiency of coordinate descent methods for huge-scale optimization problems // CORE discussion paper, 2010/2, 2010.
3. https://matplotlib.org/3.5.2/tutorials/introductory/usage.html